Core Test user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This guide describes the architecture, configuration, and use of Core Test. It will help you to understand the functionality of the driver, and provide a reference to the driver API.

The installation, build process, and general information about use of EB tresos Studio are not within the scope of this document. See the *EB tresos Studio for ACG8 user's guide* [8] for a detailed discussion of these topics.

Intended audience

This document is intended for anyone who uses the CORTST software of the TRAVEO™ T2G family.

Document structure

Chapter 1 General overview gives a brief introduction to Core Test, explains how it is implemented in the AUTOSAR environment, and describes the supported hardware and development environment.

Chapter 2 Using Core Test provides detailed steps that are required to use Core Test in your application.

Chapter 3 Structure and dependencies describes the file structure and dependencies for Core Test.

Chapter 4 EB tresos Studio configuration interface describes the driver's configuration with EB tresos Studio.

Chapter 5 Functional description provides a functional description of all services offered by Core Test.

Chapter 6 Hardware resources describes hardware resources used by Core Test.

The Appendix provides a complete API reference and access register table.

Abbreviations and definitions

Abbreviations	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
ASIL	Automotive Safety Integrity Level
Basic Software	Standardized part of software which does not fulfill a vehicle functional job.
DET	Default Error Tracer
DEM	Diagnostic Event Manager
uC	Microcontroller
EB tresos ECU AUTOSAR Suite	A bundle of AUTOSAR Basic Software modules and a Runtime Environment integrated in a common configuration and build environment
EB tresos Studio	Elektrobit Automotive configuration framework
TCM	Tightly Coupled Memory

About this document

Abbreviations	Description
DTCM	Data Tightly Coupled Memory
ITCM	Instruction Tightly Coupled Memory

Related documents

AUTOSAR requirements and specifications

Bibliography

[1] General specification of basic software modules, AUTOSAR release 4.2.2.

[2] Requirements on Core Test, AUTOSAR release 4.2.1.

[3] Specification of Core Test, AUTOSAR release 4.2.2.

[4] Specification of ECU configuration parameters, AUTOSAR release 4.2.2.

[5] Specification of default error tracer, AUTOSAR release 4.2.2.

[6] Specification of standard types, AUTOSAR release 4.2.2.

[7] Specification of diagnostic event manager, AUTOSAR release 4.2.2.

Elektrobit automotive documentation

Bibliography

[8] EB tresos Studio for ACG8 user’s guide.

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

[9] Layered software architecture, AUTOSAR release 4.2.2.

Table of contents

Table of contents

1 General overview

1.1 Introduction to Core Test

Core Test is a self-diagnostic program intended to detect failures of the processor core of a microcontroller.

It includes the following functions:

- A function to initialize all Core Test-relevant data structures, global variables, and registers with appropriate values used for Core Test.
- A function to start the test cycle from the beginning, cyclically called by the scheduler to perform the processing of Core Test, and finish it.
- A function to report the test result to the application.

Core Test is implemented according to AUTOSAR Standards and AUTOSAR Core Test SWS. See *Requirements on Core Test* [2].

In addition, Core Test is delivered as a plugin for EB tresos Studio, which allows you to statically configure the driver.

Core Test is provided with EB tresos Studio included so that you can configure the test suites.

1.2 User profile

This guide is intended for users having at least a basic knowledge of the following:

- Automotive embedded systems
- The C programming language
- The AUTOSAR standard
- The target hardware architecture

1.3 Embedding in the AUTOSAR environment

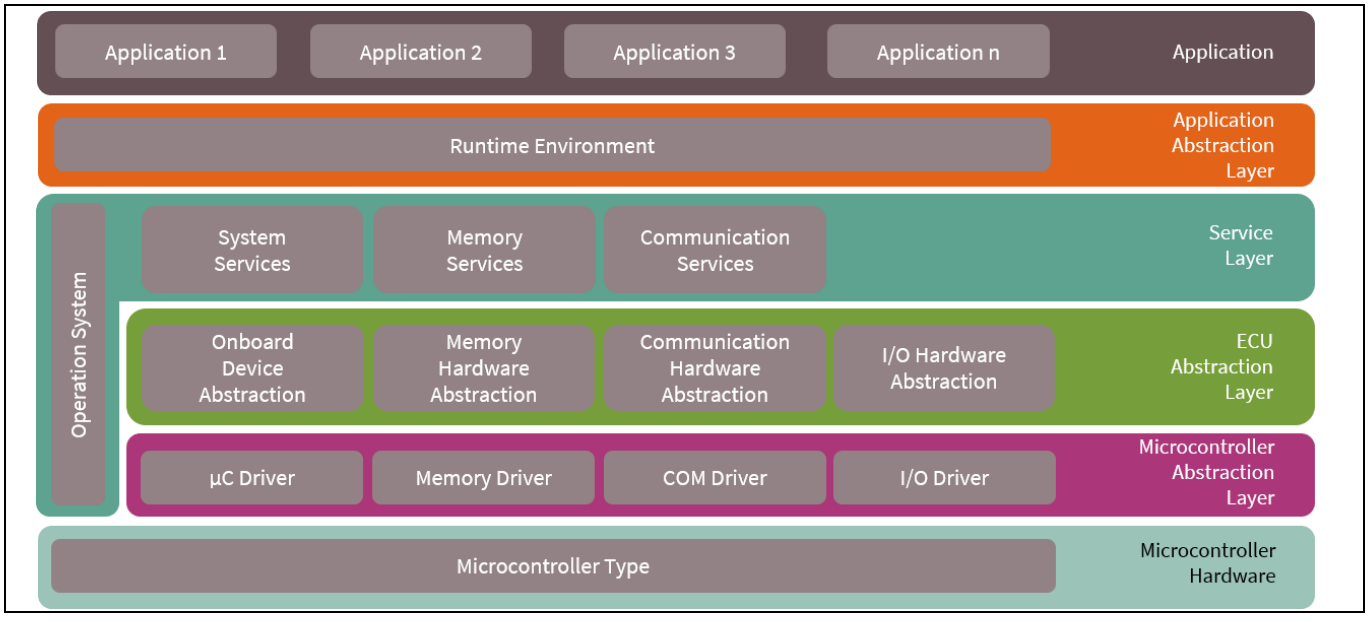

Figure 1 Overview of AUTOSAR software layers

Figure 1 shows the layered AUTOSAR software architecture. Core Test (Figure 2) is part of the microcontroller abstraction layer (MCAL), the lowest layer of basic software in the AUTOSAR environment.

For an exact overview of the AUTOSAR layered software architecture, see the *Layered Software Architecture* [9].

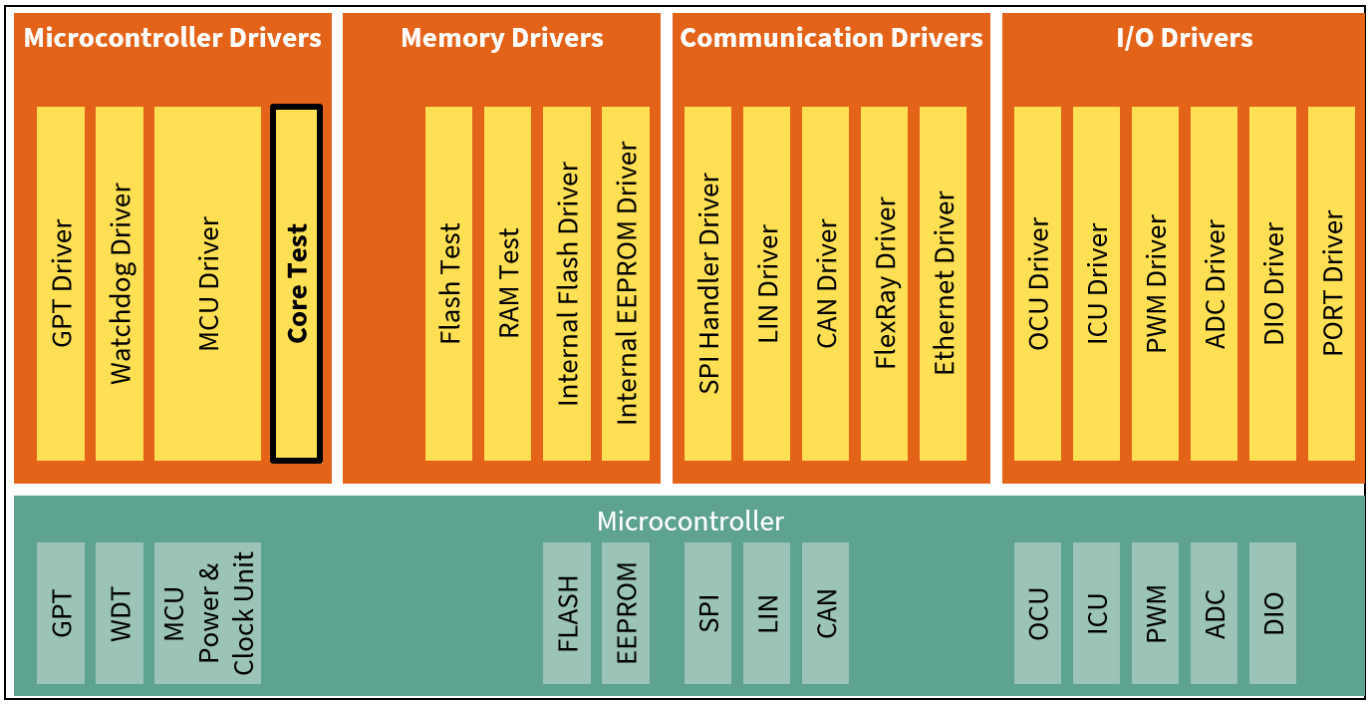

Figure 2 Core Test in MCAL layer

1.4 Supported hardware

Core Test supports the TRAVEO™ T2G microcontroller family. Supported derivatives are listed in the release notes.

Refer to the *Resource module users guide* for supported subderivatives. No further special external hardware devices are required.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The Base, Platforms, Make, and Resource modules are required for proper functionality of Core Test.

1.6 Character set and encoding

All source code files of the Core Test are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 … 0x7F) being used.

2 Using Core Test

2.1 Installation and prerequisites

Note: Before continuing with this chapter, see the EB tresos Studio for ACG8 user's guide [8] first. This provides required basic information about the installation procedure of EB tresos AUTOSAR components and the use of EB tresos Studio and EB tresos AUTOSAR build environment. In particular, you will find an explanation of how to set up and integrate your own application within the EB tresos AUTOSAR build environment.

The installation of Core Test corresponds with the general installation procedure of EB tresos AUTOSAR components given in the documents mentioned above. If the driver has been installed successfully, the driver will appear in the module list of EB tresos Studio (see *EB tresos Studio for ACG8 user's guide* [8]).

In the following sections, it is assumed that the project is properly set up and is using the application template as described in the *EB tresos Studio for ACG8 user's guide* [8]. This template provides the necessary folder structure, project and makefiles needed to configure and compile your application within the build environment. You also must be familiar with the use of the command shell.

2.2 Configuring Core Test

The following basic containers are used to configure the common behavior:

- General
- CorTstBackgroundConfigSet
- CorTstForegroundConfigSet
- CorTstIncludeFile

For detailed information and description, see chapter 4 EB tresos Studio configuration interface.

2.3 Adapting your application

To use Core Test in your application, first include Core Test header files by adding the following lines of code to your source:

```
#include "CorTst.h" /* AUTOSAR Core Test */
```

This publishes all needed function/data prototypes and symbolic names of the configuration into the application.

In addition, you must implement the error callout function for ASIL safety extension by declaring the error callout function in the specified file with the `CorTstIncludeFile` parameter and implementing it in your application (see section 7.1.5 Required callback functions, Error callout API).

The error callout function name can be configured with `CorTstErrorCalloutFunction` parameter.

Core Test initialization can be done as follows:

```
CorTst_Init(void);
```

The `CorTst_Init()` function is called first before calling any other Core Test function except `CorTst_GetState()` and `CorTst_GetVersionInfo()`.

After initializing, the background test can be executed by calling `CorTst_MainFunction()` periodically. If a failure is detected, `CorTst_MainFunction()` calls the Error Callout Handler.

If all tests are completed, or if a failure is detected, `CorTst_MainFunction()` calls the `CorTst_TestCompletedNotification()` function.

The `CorTst_GetCurrentStatus()` and `CorTst_GetSignature()` functions return the result of the latest completed the Core Test run.

`CorTst_GetCurrentStatus()`:

- `CORTST_E_NOT_OK`: 0

  Core Test detected at least one test error.

- `CORTST_E_OKAY`: 1

  Core Test passed without errors.

- `CORTST_E_NOT_TESTED`: 2

  There is no Core Test result available.

`CorTst_GetSignature()`: See `CorTst_GetSignature` in Section 7.1.4 Functions.

After initializing, the foreground test can also be executed by calling `CorTst_Start()`. If a failure is detected, `CorTst_Start()` calls the Error Callout Handler. The `CorTst_GetCurrentStatus()` and `CorTst_GetFgndSignature()` functions return the result of the latest completed Core Test run.

`CorTst_GetCurrentStatus()`: See `CorTst_GetCurrentStatus` in this section.

`CorTst_GetFgndSignature()`: See `CorTst_GetFgndSignature` in Section 7.1.4 Functions.

Core Test uses the sections in Table 1. You need to set these sections in the linker directive file (.ld).

Table 1 Core Test section names

Section name	Allocate area
`CorTst_SecCode`	ROM area
`CorTst_SecConst`	ROM area
`CorTst_SecData`	RAM area
`CorTst_SecDTCM`	DTCM area (Arm® Cortex®-M7 product only)
`CorTst_SecITCM`	ITCM area (Arm® Cortex®-M7 product only)

2.4 Starting the build process

Do the following to build your application:

Note: For a clean build, use the build command with target `clean_all` before (`make clean_all`).

1. On the command shell, type the following command to generate the necessary configuration-dependent files. See 3.3 Generated files.

```
> make generate
```

2. Type the following command to resolve required file dependencies:

```
> make depend
```

3. Type the following command to compile and link the application:

```
> make (optional target: all)
```

If the previous steps have finished successfully, the resulting binary file can be downloaded to the target hardware now.

2.5 Memory mapping

CorTst_MemMap.h in the *$(TRESOS_BASE)/plugins/MemMap_TS_T40D13M0I0R0/include* directory is only a placeholder, which is replaced by the file generated by the MEMMAP module. The input to the MEMMAP module is generated as the *CorTst_Bswmd.arxml* file in the *$(PROJECT_ROOT)/output/generated/swcd* directory of your project folder.

2.5.1 Memory allocation keyword

- `CORTST_START_SEC_CODE_ASIL_B / CORTST_STOP_SEC_CODE_ASIL_B`

  The memory section type is CODE. All executable code is allocated in this section.

- `CORTST_START_SEC_CONST_ASIL_B_UNSPECIFIED / CORTST_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

  The memory section type is CONST. All constants are allocated in this section.

- `CORTST_START_SEC_VAR_INIT_ASIL_B_UNSPECIFIED / CORTST_STOP_SEC_VAR_INIT_ASIL_B_UNSPECIFIED`

  The memory section type is VAR. All initialized variables are allocated in this section.

- `CORTST_START_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED / CORTST_STOP_SEC_VAR_NO_INIT_ASIL_B_UNSPECIFIED`

  The memory section type is VAR. All uninitialized variables are allocated in this section.

3 Structure and dependencies

Core Test consists of static, configuration, and generated files.

3.1 Static files

$(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/CorTst_TS_* is the path to the CorTst driver plugin.

$(PLUGIN_PATH)/lib_src contains all Core Test static source files. These files contain the functionality of the driver, which does not depend on the current configuration. The files are grouped into a static library. (These files exist only in the source distribution.)

$(PLUGIN_PATH)/lib contains the Core Test library files.

$(PLUGIN_PATH)/src comprises configuration-dependent source files or special derivate files. Each file will be built again when the configuration is changed.

All necessary source files will be automatically compiled and linked during the build process; all include paths will be set.

$(PLUGIN_PATH)/include contains the Core Test public header files.

$(PLUGIN_PATH)/lib_include contains the Core Test internal header files.

$(PLUGIN_PATH)/autosar directory contains the AUTOSAR ECU parameter definitions with vendor-, architecture-, and derivate-specific adaptations to create a correct matching parameter configuration for Core Test.

$(PLUGIN_PATH)/config directory contains the configuration structure defined by AUTOSAR to use Core Test.

3.2 Configuration files

Core Test configuration is done using EB tresos Studio. When saving a project, the configuration description is written in the *CorTst.xdm* file, located in your project folder under *$(PROJECT_ROOT)/config/*. This file serves as the input to generate configuration-dependent source and header files during the build process.

3.3 Generated files

During the build process, the following files are generated based on the current configuration. These files are located in the subfolder of your project folder.

- *include/CorTst_Cfg.h* provides all symbolic names of the configuration and is included by *CorTst.h*.
- *include/CorTst_ExternalInclude.h* contains the external include file information.
- *src/CorTst_Cfg.c* contains the constant structure for the Core Test configuration.
- *src/CorTst_Test_Irq.s* contains the Core Test ISR implementation.
- *make/CorTst_cfg.mak* creates additional make switches for Core Test.

Note: Generated source files need not be added to your application make file. They are compiled and linked automatically during the build process.

- *swcd/CorTst_Bswmd.arxml* contains the BSW module description.

*Note: Additional steps are required to generate the BSW module description. Select **Build Project** and click **generate_swcd** from the **Project** menu of EB tresos Studio.*

3.4 Dependencies

3.4.1 DET

If default error detection (DET) is enabled in the Core Test module configuration, DET needs to be installed, configured, and integrated into the application. For detailed information, see 5.10 Default error detection.

3.4.2 DEM

If the diagnostic event manager (DEM) is enabled in the Core Test module configuration, DEM needs to be installed, configured, and integrated into the application. The `CorTst_Start` and `CorTst_MainFunction` functions report the `CORTST_E_CORE_FAILURE` product error to DEM.

3.4.3 BSW scheduler

Core Test uses the following services of the BSW scheduler to enter and leave critical sections:

- `SchM_Enter_CorTst_CORTST_EXCLUSIVE_AREA_0 (void)`
- `SchM_Exit_CorTst_CORTST_EXCLUSIVE_AREA_0 (void)`

You must ensure that the BSW scheduler is properly configured and initialized before using Core Test.

3.4.4 Notification callout handler

The test completed notification callout handler is called whenever all tests in the background test are completed, if `CorTstNotificationSupported` is enabled in the Core Test module configuration. If so, it is necessary to prepare a function for the user application.

3.4.5 Error callout handler

The error callout handler is called on every error that is detected, independent of whether default error detection is enabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via the `CorTstErrorCalloutFunction` configuration parameter.

4 EB tresos Studio configuration interface

For further information, see *EB tresos Studio for ACG8 user's guide* [8].

4.1 CorTstConfigApiServices

This configuration comes preconfigured with the default settings. These settings must be adapted when necessary.

- `CorTstAbortApi` enables/disables the `CorTst_Abort` function.
- `CorTstGetCurrentStatus` enables/disables the `CorTst_GetCurrentStatus` function.
- `CorTstGetFgndSignature` enables/disables the `CorTst_GetFgndSignature` function.
- `CorTstGetSignature` enables/disables the `CorTst_GetSignature` function.
- `CorTstGetStateApi` enables/disables the `CorTst_GetState` function.
- `CorTstStartApi` enables/disables the `CorTst_Start` function.
- `CorTstVersionInfoApi` enables/disables the `CorTst_GetVersionInfo` function.

4.2 CorTstDemEventParameterRefs

This configuration enables/disables the DEM functionality for Core Test, and refers to configured DEM event to report "Core Test failure".

4.3 CorTstGeneral

- `CorTstDevErrorDetect` enables/disables the DET functionality for Core Test.
- `CorTstFgndTestNumber` holds the number of test configurations available for Section 4.6 CorTstForegroundConfigSet.
- `CorTstNotificationSupported` enables/disables the call-back functionality for Core Test.
- `CorTstTestIntervalIdEndValue` is the end value of the Test Interval ID.
- `CorTstTestResultMode` switches enabling test result comparison within the Core Test driver. However, the configuration switch is fixed, and the Core Test driver always compares a test result.
- `CorTstDebuggingSupport` enables/disables the use of the `CorTst_DebuggingValue` variable.
- `CorTstErrorCalloutFunction` is used to specify the error callout function name. The function is called on every error. The ASIL level of this function limits Core Test's ASIL level.

Note: *`CorTstErrorCalloutFunction` must be a valid C function name; otherwise an error would occur in the configuration phase.*

- `CorTstIncludeFile` is a list of file names that must be included within the driver. Any application-specific symbol that is used by the Core Test configuration (such as the error callout function) must be included by configuring this parameter.

Note: *`CorTstIncludeFile` must be a filename with the .h extension and have a unique name; otherwise errors would occur in the configuration phase.*

4.4 InterruptHandlerChannelNumber

Core Test uses an interrupt handler in interrupt test. `InterruptHandlerChannelNumber` determines the interrupt channel number that is used in Core Test. You can ignore this value if interrupt test is disabled.

4.5 CorTstBackgroundConfigSet

This configuration specifies the kind of test to execute by the background test described in Section 5.4 Background test.

- `CorTstSelect`: This function enables the selection of an individual test for background test. Only the test selected here is executed. Only one configuration set can be created.
- `CorTstAddress` enables/disables the address test.
- `CorTstAlu` enables/disables the ALU test.
- `CorTstCache` enables/disables the cache test.
- `CorTstInterrupt` enables/disables the interruption test.
- `CorTstMemoryIf` enables/disables the memory interface test.
- `CorTstMpu` enables/disables the MPU test.
- `CorTstRegister` enables/disables the register rest.
- `CorTstVfp` enables/disables the VFP rest.
- `CorTstVfpRegister` enables/disables the VFP register test.

4.6 CorTstForegroundConfigSet

This configuration specifies the kind of test to execute by foreground test described in Section 5.5 Foreground test.

- `CorTstTestIdFgnd` is the Id which identifies the foreground test.

Note: This value is assigned to a symbolic name. You can use the symbolic names defined in CorTst_Cfg.h for `CorTst_Start` calls.

- `CorTstSelect`: The function enables selections of individual test for the foreground test.
- `CorTstAddress` enables/disables the address test.
- `CorTstAlu` enables/disables the ALU test.
- `CorTstCache` enables/disables the cache test.
- `CorTstInterrupt` enables/disables the interruption test.
- `CorTstMemoryIf` enables/disables the memory interface test.
- `CorTstMpu` enables/disables the MPU test.
- `CorTstRegister` enables/disables the register test.
- `CorTstVfp` enables/disables the VFP test.
- `CorTstVfpRegister` enables/disables the VFP register test.

5 Functional description

5.1 Inclusion

The *CorTst.h* file includes all necessary external identifiers, so your application only needs to include *CorTst.h* to make all API functions and data types available.

5.2 Initialization

The `CorTst_Init()` function initializes all `CorTst`-relevant data structures, global variables, and registers with appropriate values that are used for Core Test. This function changes the execution state to `CORTST_INIT`.

5.3 Deinitialization

The `CorTst_DeInit()` function finishes the Core Test run, and changes the execution state to `CORTST_UNINIT`.

5.4 Background test

Contents of `CorTst_MainFunction`.

After initializing, Core Test is started by BSW, and executes the `CorTst_MainFunction()` function. The function executes each atomic test instruction set. After all atomic test instruction set is completed, the function calls the complete notification as SUCCESS/FAIL. If failure is detected, the atomic test sequence is completed and the error callout handler is called.

5.5 Foreground test

After initializing, you can start Core Test. It then executes the `CorTst_Start()` function. The function executes all test sets, and returns the value as ACCEPT/REFUSE. The function cannot be canceled during execution.

Note: *The function cannot execute a background test at the same time as a foreground test. To execute the foreground test after executing the background test, execute the foreground test at the time of completing of the background test. Avoid restarting the background test.*

5.6 Test cancelling

The `CorTst_Abort()` function can cancel any atomic test instruction set by background test. Background test is not continued if canceled. In addition, the foreground test also cannot be executed.

5.7 Get state

The `CorTst_GetState()` function checks the state of Core Test.

5.8 Get result

The `CorTst_GetCurrentStatus()` function checks the Core Test result. If the result is not found, it returns `CORTST_E_NOT_TESTED`.

5.9 Get signature

The `CorTst_GetSignature()` function and the `CorTst_GetFgndSignature()` function returns the last completed test's signature. If there is no completed result, it returns the value as 0.

5.10 Default error detection

If default error detection (DET) is enabled, this function reports development errors. A development error is an error which shall be detected and fixed during the development phase. A development error shall not occur in production code. All development errors are reported to the DET, a central error hook function within the AUTOSAR environment.

If an error occurs, the error hook routine will be called and the error code, as well as the service and the module ID will be passed on as parameters. For more information about DET and its API, see *specification of default error tracer* [5].

The `CorTst_DeInit`, `CorTst_Abort`, `CorTst_GetCurrentStatus`, `CorTst_GetSignature`, `CorTst_MainFunction`, `CorTst_GetFgndSignature`, and `CorTst_Start` functions check whether Core Test has already been initialized at the beginning of the function.

In the case of an uninitialized driver, the error code `CORTST_E_UNINIT` will be reported.

The `CorTst_Init` function checks if the driver has already been initialized. In the case of an already initialized driver, the error code `CORTST_E_ALREADY_INITIALIZED` will be reported.

The `CorTst_DeInit` and `CorTst_Start` functions check if the driver state is in `CORTST_RUNNING_BGND`. In this case, the error code `CORTST_E_STATUS_FAILURE` will be reported.

The `CorTst_GetVersionInfo` and `CorTst_GetCurrentStatus` functions check if the function is called with NULL pointer. In case of NULL pointer, the error code `CORTST_E_PARAM_POINTER` will be reported.

5.11 Notification

The `CorTst_MainFunction()` function calls the `CorTst_TestCompletedNotification()` function if all atomic test instruction sets are completed or a core failure is detected. You must implement the contents in this function.

5.12 Reentrancy

Only `CorTst_GetVersionInfo` is available for reentrancy.

5.13 Debugging support

The following variables are available for AUTOSAR debugging. All type definitions of variables are accessible by the *CorTst.h* header file.

- The `CorTst_DebuggingValue` variable can change the signature value of a test result if `CorTstDebuggingSupport` in the configuration is selected. The value set in the `CorTst_DebuggingValue` variable is added to a normal signature value, and can cause a pseudo error. The initial value is 0, and is in the normal state.

6 Hardware resources

6.1 CORE

Core Test supports the Arm® Cortex®-M4F and Cortex®-M7 processor core. Arm® Cortex®-M0+ processor core is not supported.

See the *resource module users guide* for supported subderivatives. No further special external hardware devices are required.

6.2 Interrupts

In interrupt test, Core Test sets a test handler for the interrupt number (in the range IRQ8 - IRQ15) specified by the configuration described in section 4.4 InterruptHandlerChannelNumber, and executes tests using it.

6.3 Memory protection unit

In MPU test, Core Test sets the memory management fault handler, and executes tests using it. However, when returning from the test API (`CorTst_Start` or `CorTst_MainFunction`), the handler is restored by Core Test.

6.4 TCM

In the memory interface test, Core Test uses DTCM and ITCM areas (Cortex®-M7 product only).

6.5 Cache

In cache test, Core Test uses instruction cache and data cache, and invalidates both the caches (Arm® Cortex®-M7 product only).

7 Appendix

7.1 API reference

7.1.1 Data types

7.1.1.1 CorTst_ConfigType

Type

```
typedef struct
{
   uint32 CorTstConfigurationData;
} CorTst_ConfigType;
```

Description

Type for configuration for Core Test initialization

7.1.1.2 CorTst_CsumSignatureType

Type

```
uint32
```

Description

Type for Core Test return value if a checksum/signature is returned from the API to the caller of the API

7.1.1.3 CorTst_CsumSignatureBgndType

Type

```
typedef struct
{
   uint32 CorTstBgndSignature;
   uint32 CorTstTestIntervalId;
} CorTst_CsumSignatureBgndType;
```

Description

Type for test signature in background mode

7.1.1.4 CorTst_ErrOkType

Type

```
typedef struct
{
   uint32            CorTstTestIntervalId;
   CorTst_ResultType returnvalue;
} CorTst_ErrOkType;
```

Description

Type of the return from Core Test

7.1.1.5 CorTst_ResultType

Type

```
typedef enum
{
   COTTST_E_NOT_OK     = 0x00, /** Last Test result is NG. */
   CORTST_E_OKAY       = 0x01, /** Last Test result is OK. */
   CORTST_E_NOT_TESTED = 0x02, /** There is no result to return. */
} CorTst_ResultType;
```

Description

Status value returned by the `CorTst_GetCurrentStatus()` API, the result of Core Test that ended last.

7.1.1.6 CorTst_StateType

Type

```
typedef enum
{
   CORTST_ABORT        = 0x00, /** The Core Test has been cancelled
                                  by API CorTst_Abort(). */
   CORTST_INIT         = 0x01, /** The Core Test is initialized
                                  and can be started. */
   CORTST_UNINIT       = 0x02, /** The Core Test can be
                                  initialized. */
   CORTST_RUNNING_BGND = 0x03  /** The Core Test is
                                  currently executed. */
} CorTst_StateType;
```

Description

The state value returned by the `CorTst_GetState()` API

7.1.1.7 CorTst_TestIdFgndType

Type

`uint32`

Description

Type of the parameter (Id) used for a specific foreground test

7.1.1.8 CorTst_Type_U32

Type

`unsigned long`

Description

Type used by the `CorTst_DebuggingValue` variable.

7.1.2 Constants

7.1.2.1 Error codes

The service might return the following error codes:

Table 2 Error codes

Name	Value	Description
CORTST_E_PARAM_INVALID	0x11	API argument incorrect
CORTST_E_UNINIT	0x20	API called without Core Test initialization
CORTST_E_ALREADY_INITIALIZED	0x23	API service CorTst_Init() called again without a CorTst_DeInit() in between
CORTST_E_PARAM_POINTER	0x24	API service called with a NULL pointer for CorTst_GetVersionInfo() and CorTst_GetCurrentStatus()
CORTST_E_CORE_FAILURE_FOR_CALLOUT	0x30	Core failure during tests. This error ID is used to call the error callout handler.
CORTST_E_STATUS_FAILURE	0x01	API called in an unexpected state
CORTST_E_CORE_FAILURE	External allocation by DEM	Core failure during tests

7.1.2.2 Version information

The following version information is published in the driver's header file:

Table 3 Version information

Name	Value	Description
CORTST_SW_MAJOR_VERSION	x (See the delivery notes)	Software major version number
CORTST_SW_MINOR_VERSION	x (See the delivery notes)	Software minor version number
CORTST_SW_PATCH_VERSION	x (See the delivery notes)	Software patch version number

7.1.2.3 Module information

Table 4 Module information

Name	Value	Description
CORTST_MODULE_ID	103	Module ID
CORTST_VENDOR_ID	66	Vendor ID

7.1.2.4 API service IDs

The following API service IDs are published in the driver's header file:

Table 5 API service IDs

Name	Value	Description
CORTST_API_INIT	0x00	Service ID of CorTst_Init
CORTST_API_DEINIT	0x01	Service ID of CorTst_DeInit
CORTST_API_ABORT	0x02	Service ID of CorTst_Abort
CORTST_API_GET_STATE	0x03	Service ID of CorTst_GetState
CORTST_API_GET_CURRENT_STATUS	0x04	Service ID of CorTst_GetCurrentStatus
CORTST_API_GET_SIGNATURE	0x05	Service ID of CorTst_GetSignature
CORTST_API_GET_FGND_SIGNATURE	0x06	Service ID of CorTst_GetFgndSignature
CORTST_API_START	0x07	Service ID of CorTst_Start
CORTST_API_GET_VERSION_INFO	0x08	Service ID of CorTst_GetVersionInfo
CORTST_API_MAINFUNCTION	0x0b	Service ID of CorTst_MainFunction
CORTST_API_TESTCOMPLETED	0x0c	Service ID of CorTst_TestCompletedNotification

7.1.3 Variables

7.1.3.1 CorTst_DebuggingValue

Type

CorTst_Type_U32

Description

This value is added to the normal signature value, and can cause a pseudo error. The initial value is 0, and is in the normal state. If a value other than 0 is set for this variable, Core Test will cause an error. This variable can be used only when CorTstDebuggingSupport in the configuration is selected.

7.1.4 Functions

7.1.4.1 CorTst_Init

Syntax

```
void CorTst_Init(
   const CorTst_ConfigType* ConfigPtr
)
```

Service ID

0x00

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

`ConfigPtr` (Pointer to the selected configuration set; however, it always has a NULL_PTR value.)

Parameters (inout)

None

Parameters (out)

None

Return value

None

DET errors

`CORTST_E_ALREADY_INITIALIZED` – The Core Test has already been initialized.

DEM errors

None

Description

The `CorTst_Init` function initializes Core Test, and changes the state to `CORTST_INIT`. Functions other than `CorTst_GetState` and `CorTst_GetVersionInfo` are available after this execution.

7.1.4.2 CorTst_DeInit

Syntax

```
void CorTst_DeInit(
   void
)
```

Service ID

0x01

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (inout)

None

Parameters (out)

None

Return value

None

DET errors

`CORTST_E_UNINIT` – Core Test has not been initialized.

`CORTST_E_STATUS_FAILURE` – The function is called in an unexpected state.

DEM errors

None

Description

The `CorTst_DeInit` function changes the state from `CORTST_ABORT` or `CORTST_INIT` to `CORTST_UNINIT`. Functions other than `CorTst_Init`, `CorTst_GetState`, and `CorTst_GetVersionInfo` are unavailable after the execution of this function.

7.1.4.3 CorTst_Abort

Syntax

```
void CorTst_Abort(
   void
)
```

Service ID

0x02

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (inout)

None

Parameters (out)

None

Return value

None

DET errors

`CORTST_E_UNINIT` – Core Test has not been initialized.

DEM errors

None

Description

The `CorTst_Abort` function changes the state from `CORTST_INIT` or `CORTST_RUNNING_BGND` to `CORTST_ABORT`.

When the Core Test state is `CORTST_ABORT`, the state is not changed.

After a call to `CorTst_Abort`, `CorTst_MainFunction` does not begin testing again when called by the scheduler before a complete reinitialization of the Core Test module takes place by calling `CorTst_DeInit` and `CorTst_Init` again.

`CorTst_Start` cannot execute before a complete reinitialization of the Core Test module.

When `CorTst_Abort` is called, it sets the result of the `CorTst_GetCurrentStatus` function to return `CORTST_E_NOT_TESTED`.

7.1.4.4 CorTst_GetState

Syntax

```
CorTst_StateType CorTst_GetState(
   void
)
```

Service ID

0x03

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (inout)

None

Parameters (out)

None

Return value

`CorTst_StateType` (See 7.1.1 Data types)

DET errors

None

DEM errors

None

Description

The `CorTst_GetState` function returns the Core Test state. This service is available before and after the initialization.

7.1.4.5 CorTst_GetCurrentStatus

Syntax

```
void CorTst_GetCurrentStatus(
   CorTst_ErrOkType* ErrOk
)
```

Service ID

0x04

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (inout)

None

Parameters (out)

`ErrOk` (See 7.1.1 Data types)

Return value

None

DET errors

`CORTST_E_UNINIT` – Core Test has not been initialized.

`CORTST_E_PARAM_POINTER` – The function is called with a NULL pointer.

DEM errors

None

Description

The `CorTst_GetCurrentStatus` function gets the result of the last execution of Core Test. If the result is not found, it returns `CORTST_E_NOT_TESTED`.

7.1.4.6 CorTst_GetSignature

Syntax

```
CorTst_CsumSignatureBgndType CorTst_GetSignature(
   void
)
```

Service ID

0x05

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (inout)

None

Parameters (out)

None

Return value

`CorTst_CsumSignatureBgndType`

DET errors

`CORTST_E_UNINIT` – Core Test has not been initialized.

DEM errors

None

Description

The `CorTst_GetSignature` function gets the signature of the last execution of Core Test in background mode. The signature value which this function returns changes with tests selected by the configuration. See Table 6. When two or more tests are chosen, the signature value is the sum of the signature value of the selected test.

Table 6 Signature values

Selected test	Signature value
`CorTstAddress`	0x00000101
`CorTstAlu`	0x00000202
`CorTstCache`	0x00000404
`CorTstInterrupt`	0x00000808
`CorTstMemoryIf`	0x00001010

Selected test	Signature value
CorTstMpu	0x00002020
CorTstRegister	0x00004040
CorTstVfp	0x01010000
CorTstVfpRegister	0x02020000

7.1.4.7 CorTst_GetFgndSignature

Syntax

```
CorTst_CsumSignatureType CorTst_GetFgndSignature(
   void
)
```

Service ID

0x06

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

None

Parameters (inout)

None

Parameters (out)

None

Return value

`CorTst_CsumSignatureType`

DET errors

`CORTST_E_UNINIT` – Core Test has not been initialized.

DEM errors

None

Description

The `CorTst_GetFgndSignature` function gets the signature of the last execution of Core Test in foreground mode. The signature value which this function returns changes with tests selected by the configuration. See Table 6. When two or more tests are chosen, the signature value is the sum of the signature value of the selected test.

7.1.4.8 CorTst_Start

Syntax

```
Std_ReturnType CorTst_Start(
   CorTst_TestIdFgndType TestId
)
```

Service ID

0x07

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

`TestId` (Id of the foreground test configuration to be executed.)

Parameters (inout)

None

Parameters (out)

None

Return value

`Std_ReturnType`

(`E_OK`: Foreground test processed)

(`E_NOT_OK`: Foreground test not accepted)

DET errors

`CORTST_E_PARAM_INVALID` – The test id is not configured.

`CORTST_E_UNINIT` – Core Test has not been initialized.

DEM errors

`CORTST_E_CORE_FAILURE` – Core failure during tests

Description

The `CorTst_Start` function executes the foreground test of Core Test.

If the execution state is `CORTST_ABORT`, this function returns without any action and the return value is `E_NOT_OK`.

7.1.4.9 CorTst_GetVersionInfo

Syntax

```
void CorTst_GetVersionInfo(
   Std_VersionInfoType* versioninfo
)
```

Service ID

0x08

Sync/Async

Synchronous

Reentrancy

Reentrant

Parameters (in)

None

Parameters (inout)

None

Parameters (out)

`versioninfo` (Pointer to where to store the version information of Core Test.)

Return value

None

DET errors

`CORTST_E_PARAM_POINTER` – The function is called with a NULL pointer.

DEM errors

None

Description

The `CorTst_GetVersionInfo` function returns the version information of Core Test.

7.1.4.10 CorTst_MainFunction

Syntax

```
void CorTst_MainFunction(
   void
)
```

Service ID

0x0b

DET errors

`CORTST_E_UNINIT` – Core Test has not been initialized.

DEM errors

`CORTST_E_CORE_FAILURE` – Core failure during tests.

Description

The `CorTst_MainFunction` function executes the background test of Core Test. It is cyclically called by the scheduler, and executes all the tests when called repeatedly.

If the Core Test module is in the state of `CORTST_INIT`, this function set state to `CORTST_RUNNING_BGND`. However, if the all tests of Core Test are completed, the function sets the state to `CORTST_INIT`.

7.1.5 Required callback functions

7.1.5.1 CorTst_TestCompletedNotification

Syntax

```
void CorTst_TestCompletedNotification(
   CorTst_ErrOkType ResultOfLastCorTstRun
)
```

Service ID

0x0c

Sync/Async

Synchronous

Reentrancy

Non-reentrant

Parameters (in)

`ResultOfLastCorTstRun`

(`CORTST_E_OKAY`: Last Core Test execution successfully finished with no errors)

(`CORTST_E_NOT_OK`: Last Core Test execution finished with errors.)

Parameters (inout)

None

Parameters (out)

None

Return value

None

Description

The `CorTst_TestCompletedNotification` function is called when all tests of the background test are completed. The user application should configure this function, because it is not distributed with Core Test. Only the declaration of this function exists in *CorTst.h*.

7.1.5.2 DET

Syntax

```
Std_ReturnType Det_ReportError(
   uint16 ModuleId,
   uint8 InstanceId,
   uint8 ApiId,
   uint8 ErrorId
)
```

Service ID

0x01

Sync/Async

Depending on implemented functionality

Reentrancy

Reentrant

Parameters (in)

`ModuleId` (Module ID of calling module.)

`InstanceId` (Instance ID of calling module.)

`ApiId` (ID of the API service that calls this function.)

`ErrorId` (Error code of the detected error.)

Parameters (inout)

None

Parameters (out)

None

Return value

Returns always `E_OK` (is required for services).

Description

Service for reporting development errors.

7.1.5.3 DEM

Syntax

```
void Dem_ReportErrorStatus(
   Dem_EventIdType EventId,
   Dem_EventStatusType EventStatus
)
```

Service ID

0x0f

Sync/Async

Asynchronous

Reentrancy

Reentrant

Parameters (in)

`EventId` (Identification of an event by assigned Event ID.)

`EventStatus` (Monitor test result of given event.)

Parameters (inout)

None

Parameters (out)

None

Return value

None

Description

Service for reporting diagnostic events.

7.1.5.4 Error callout API

Syntax

```
void Error_Handler_Name (
   uint16 ModuleId,
   uint8 InstanceId,
   uint8 ApiId,
   uint8 ErrorId
)
```

Service ID

None

Sync/Async

Synchronous

Reentrancy

Reentrant

Parameters (in)

`ModuleId` (Module ID of calling module.)

`InstanceId` (Instance ID of calling module.)

`ApiId` (ID of the API service that calls this function.)

`ErrorId` (Error code of the detected error.)

Parameters (inout)

None

Parameters (out)

None

Return value

None

Description

Service for reporting errors.

7.1.6 State machine diagram

Each state in Core Test and the function that can use the status are as follows:

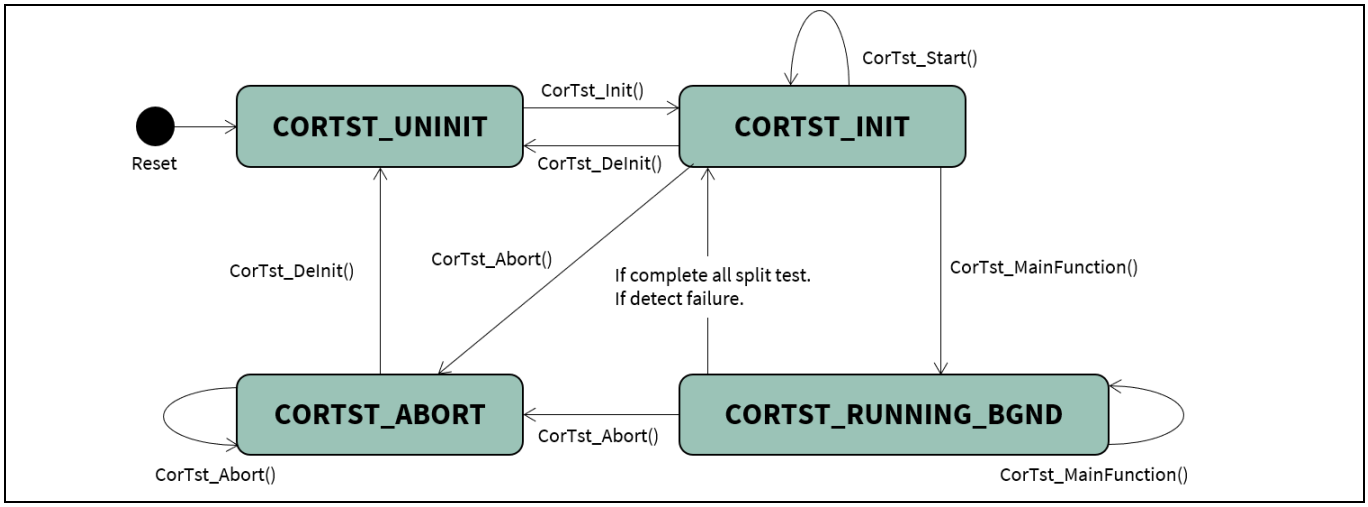

Figure 3 State machine diagram

8 Appendix access register table

8.1 Access register table

8.1.1 Core

Table 7 ARM core register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
APSR (Application program status register)	31:0	Word (32 bits)	0x00000000	Provide flag information. [16] – [19] and [27] – [31] bit of this register is changed.	`Alu Test`	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
IPSR (Interrupt program status register)	31:0	Word (32 bits)	- (Read only.)	Provide the exception number of the exception being processed.	`Interrupt Test`	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

CONTROL (The special-purpose CONTROL register)	2:0	3 bits	bit[0] = 0	Provide execution privilege in Thread mode, stack to be used and FP extension is active.	`Interrupt Test`
					`Mpu Test`
					`Cache Test`
					(Only when calling Interrupt Test, Mpu Test or Cache Test in non-privileged mode.)
						0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

FPSCR (Floating-point status and control register)	31:0	Word (32 bits)	0x00000000	Provide floating point system information.	`Vfp Test` `Vfp Register Test`	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

8.1.2 CM4_SCS

Table 8 Cortex®-M4F system control space (SCS) access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
VTOR (Vector table offset register)	31:0	Word (32 bits)	– (The address of CorTst ram area)	The vector table address	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ISER0 (Interrupt set-enable registers)	31:0	Word (32 bits)	– (Depends on configuration value.)	Enables or shows the current enabled state of interrupt. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ISPR0 (Interrupt set-pending registers)	31:0	Word (32 bits)	– (Depends on configuration value.)	Changes the state of interrupt to pending, or shows whether the state of the interrupt is pending. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ICER0 (Interrupt clear-enable registers	31:0	Word (32 bits)	– (Depends on configuration value.)	Disables or shows the current enabled state of interrupt. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ICPR0 (Interrupt clear-pending registers)	31:0	Word (32 bits)	– (Depends on configuration value.)	Clears the pending state of interrupt, or shows whether the state of the interrupt is pending. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_IPR2, NVIC_IPR3 (Interrupt priority registers)	31:0	Byte (8 bits)	– (Depends on configuration value.)	Sets or reads interrupt priorities. [5] – [7], [13] – [15], [21] – [23] and [29] – [31] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
STIR (Software triggered interrupt register)	31:0	Word (32 bits)	– (Depends on configuration value.)	Changes the state of interrupt to pending.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_IABR0 (Interrupt active bit registers)	31:0	Word (32 bits)	– (Read only.)	Provides whether each interrupt is active.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
ICSR (Interrupt control and state register)	31:0	Word (32 bits)	– (Read only.)	Provides the exception number of the current executing exception.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
SHCSR (System handler control and state register)	31:0	Word (32 bits)	– (Read only.)	Provides UsageFault is active.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
CCR (Configuration and control register)	31:0	Word (32 bits)	– (Read only.)	Provides instruction cache enable.	Interrupt Test Mpu Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
CFSR (Configurable fault status register)	31:0	Word (32 bits)	0x00000001 0x00000082 0x00010000	Clear the fault status	Interrupt Test Mpu Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

8.1.3 CM7_SCS

Table 9 Cortex®-M7 system control space (SCS) access register table

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
VTOR (Vector table offset register)	31:0	Word (32 bits)	– (The address of CorTst ram area)	The vector table address	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ISER0 (Interrupt set-enable registers)	31:0	Word (32 bits)	– (Depends on configuration value.)	Enables or shows the current enabled state of interrupt. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ISPR0 (Interrupt set-pending registers)	31:0	Word (32 bits)	– (Depends on configuration value.)	Changes the state of interrupt to pending, or shows whether the state of the interrupt is pending. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ICER0 (Interrupt clear-enable registers	31:0	Word (32 bits)	– (Depends on configuration value.)	Disables or shows the current enabled state of interrupt. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_ICPR0 (Interrupt clear-pending registers)	31:0	Word (32 bits)	– (Depends on configuration value.)	Clears the pending state of interrupt, or shows whether the state of the interrupt is pending. [8] – [15] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_IPR2, NVIC_IPR3 (Interrupt priority registers)	31:0	Byte (8 bits)	– (Depends on configuration value.)	Sets or reads interrupt priorities. [5] – [7], [13] – [15], [21] – [23] and [29] – [31] bit of this register is changed by the configuration.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
STIR (Software triggered interrupt register)	31:0	Word (32 bits)	– (Depends on configuration value.)	Changes the state of interrupt to pending.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
NVIC_IABR0 (Interrupt active bit registers)	31:0	Word (32 bits)	– (Read only.)	Provides whether each interrupt is active.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
ICSR (Interrupt control and state register)	31:0	Word (32 bits)	– (Read only.)	Provides the exception number of the current executing exception.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
SHCSR (System handler control and state register)	31:0	Word (32 bits)	– (Read only.)	Provides UsageFault is active.	Interrupt Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
CCR (Configuration and control register)	31:0	Word (32 bits)	– (Read only.)	Provides instruction cache enable.	Interrupt Test Mpu Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
DCCIMVAC (D-cache clean and invalidate by MVA to PoC)	31:0	Word (32 bits)	– (The address of CorTst ram area)	Cleans and invalidates D-cache.	Cache Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
ICIMVAU (I-cache invalidate by MVA to PoU)	31:0	Word (32 bits)	– (The address of CorTst flash area)	Invalidates I-cache.	Interrupt Test Mpu Test Cache Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)
CFSR (Configurable fault status register)	31:0	Word (32 bits)	0x00000001 0x00000082 0x00010000	Clear the fault status	Interrupt Test Mpu Test	0x00000000 (monitoring is not needed.)	0x00000000 (monitoring is not needed.)

Revision history

Revision	Issue date	Description of change
**	2018-01-17	New spec.
*A	2018-06-05	- Related documentation Updated data sheet name and number - 2.5 Memory mapping Changed file name from BswImplemation.bmd to CorTst_Bswmd.arxml. - 3.3 Generated files Changed file name from BswImplemation.bmd to CorTst_Bswmd.arxml. Changed descriptions of CorTst_Irq.c and .h - A.1.4 Functions Modified table number of description in CorTst_GetFgndSignature and CorTst_GetSignature. - B.1 Access register table Added CONTROL and CM4_SCS_STIR register
*B	2018-12-05	- Glossary Added ASIL, TCM, DTCM and ITCM - Related documentation Updated hardware documentation - 1.5 Development environment Added platforms module - 2.3 Adapting your application Added CorTst_SecDTCM and CorTst_SecITCM in Table 2-1 - 4.5 CorTstBackgroundConfigSet Added CorTstCache - 4.6 CorTstForegroundConfigSet Added CorTstCache - 6.1 CORE Added Cortex®-M7 - 6.4 TCM Added new section - 6.5 Cache Added new section - A.1.4 Functions Added CorTstCache in Table A-5 - B.1.1 Core Added IPSR - B.1.2 CM4_SCS Removed CM4_SCS_ prefix Added NVIC_IABR0, ICSR, SHCSR and CCR

Revision history

Revision	Issue date	Description of change
		- B.1.3 CM7_SCS Added new section
*C	2019-06-17	- Related documentation Updated hardware documentation information.
*D	2020-09-06	- 2.5 Memory mapping Changed a memmap file include folder in section "Memory mapping". -4.6 CorTstForegroundConifgSet Added note.
*E	2020-11-20	MOVED TO INFINEON TEMPLATE.
*F	2021-12-23	Updated to Infineon style.
*G	2023-10-06	Updated 8.1.2 CM4_SCS - Added CFSR. Updated 8.1.3 CM7_SCS - Added CFSR.
*H	2023-12-08	Web release. No content updates.

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2023-12-08

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-22286 Rev. *H

Warnings
Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.