MICROSAR OS SafeContext

Safety Manual

RH850 with Green Hills Compiler

Authors	Senol Cendere, Yohan Humbert
Version	1.05
Status	Released
Document ID	OS03.00124.10

Document Information

History

Author	Date	Version	Remarks
Senol Cendere	2014-02-17	1.00	Creation for RH850
Senol Cendere	2014-02-26	1.01	Updated the Requirement IDs
Senol Cendere	2014-05-09	1.02	Adaption for RH850 P1M
Senol Cendere	2014-08-18	1.03	Reworked after Safety Manual Review
Senol Cendere	2014-09-22	1.04	Added reference for Renesas Electronics RH850/P1M Safety Application Note Removed CPU derivative specification Removed compiler options (both are specified in safety case)
Yohan Humbert	2014-12-03	1.05	Added level support

Reference Documents

No.	Source	Title	Version
[1]		AUTOSAR Operating System Specification[1]	3.x - 4.x
[2]		OSEK/VDX Operating System Specification[2]	2.2.3
[3]	Vector Informatik GmbH	User manual of Vector MICROSAR OS TechnicalReference_Microsar_Os.pdf	6.03
[4]	Vector Informatik GmbH	User manual of Vector MICROSAR OS RH850, hardware specific part TechnicalReference_MICROSAROS_RH850_SafeContext.pdf	1.04
[5]		International Organization for Standardization, Draft International Standard ISO/DIS 26262 Road Vehicles - Functional Safety (all parts), 2009	
[6]	Renesas Electronics	V850E3v5 Architecture Specifications	(4th edition)
[7]	Renesas Electronics	RH850 G3M User's Manual: Software r01us0042ej0020_rh850g3m.pdf	Rev. 0.10 Oct. 2012
[8]	Renesas Electronics	RH850/P1x Group User's Manual: Hardware r01uh0436ej0041_rh850p1x.pdf	Rev. 0.60 Jul. 2014
[9]	Green Hills Software	MULTI: Building Applications for Embedded V850 and RH850 build_v800.pdf	PubID: build_v800-472243 Date: September 12, 2012
[10]	Vector Informatik GmbH	Vector MICROSAR OS SafeContext Concept	1.03
[11]	Renesas Electronics	RH850/P1M Safety Application Note	Rev.0.20 September 1, 2014

[1] This document is available in PDF-format on the Internet at the Autosar homepage: http://www.Autosar.org

[2] This document is available in PDF-format on the Internet at the OSEK/VDX homepage: http://www.osek-vdx.org

Contents

Illustrations

Tables

1 Purpose

1.1 Safety Element out of Context (SEooC)

MICROSAR OS SafeContext is a Safety Element out of Context (SEooC). It is developed based on assumptions on the intended functionality, use and context, including external interfaces. To have a complete safety case, the validity of these assumptions has to be checked in the context of the actual item after integration of the SEooC.

The application conditions for SEooC provide the assumptions made on the requirements (including safety requirements) that are placed on the SEooC by higher levels of design and also on the design external to the SEooC and the assumed safety requirements and assumptions related to the design of the SEooC.

Information given by this document helps to check whether the SEooC fulfills the item requirements, or whether a change to the SEooC is necessary in accordance with the requirements of ISO 26262.

1.2 Standards and Legal requirements

Standards followed by the development of MICROSAR OS SafeContext:

- ISO 26262[3]
- OSEK OS[4]
- AUTOSAR OS[5]

[3] International Standard ISO 26262 Road Vehicles - Functional Safety (all parts), 2011

[4] OSEK/VDX Operating System, v2.2.3

[5] AUTOSAR Specification of Operating System

2 Concept

This chapter provides a description of the assumed safety requirements and the main concept.

2.1 SafeContext Is One Part of a Whole

SafeContext is part of Vector SafeExecution. SafeExecution consists of SafeContext for prevention from corrupted data and SafeWatchdog for supervision of timing behavior. This document covers SafeContext only. [SPMF92:0050]

2.2 Safety Goal

The safety goal is to ensure context integrity for all safety critical parts. Whenever a safety critical code is executed, it is guaranteed that the code is executed with the correct context. After pre-emption or interruption, execution is resumed with the correct context. The integrity of the memory is ensured by usage of hardware (e.g. MPU) and software measures.

2.3 Safety Requirements

To achieve this safety goal, the following assumed safety requirements are provided by SafeContext:

ASA_OS_1: Non-trusted software must be prevented from overwriting data of safety relevant software.

ASA_OS_2: A *runtime context* (Task, Hook, (Non-)Trusted-Function and ISR) must not be destroyed by a switch (to another runtime context).

ASA_OS_3: A runtime context shall be set up according to compiler and processor specifications.

ASA_OS_4: Services to prevent data inconsistencies by racing conditions shall be provided.

ASA_OS_5: The OS never writes to unintended memory locations.

2.4 SafeContext Functionality

MICROSAR OS SafeContext implements the AUTOSAR OS and OSEK OS standards of a real-time operating system with some restrictions.

The functionality provided by MICROSAR OS is task switching, interrupt handling, memory protection handling, timer services and others. SafeContext provides an efficient solution also in respect of safety. Therefore the OSEK/AUTOSAR OS functionality is divided into 3 groups:

1. Functionality implemented according ISO 26262 to achieve ASIL.
2. Functionality implemented according to “Silent” principle to prevent from safety data overwritten by OS code.
3. Functionality which is not supported in SafeContext. It is assumed that this functionality is not needed by a safety related ECU or it can be reached by other existing means.

This chapter describes which of the groups certain functionality falls into.

Basic Knowledge
The idea of “Silent” code is not to disturb other software by means of unintended memory writes. To provide high performance, this is not achieved by a hardware protection mechanism (which would require MPU reconfiguration for each API call) but by analysis of the OS code.

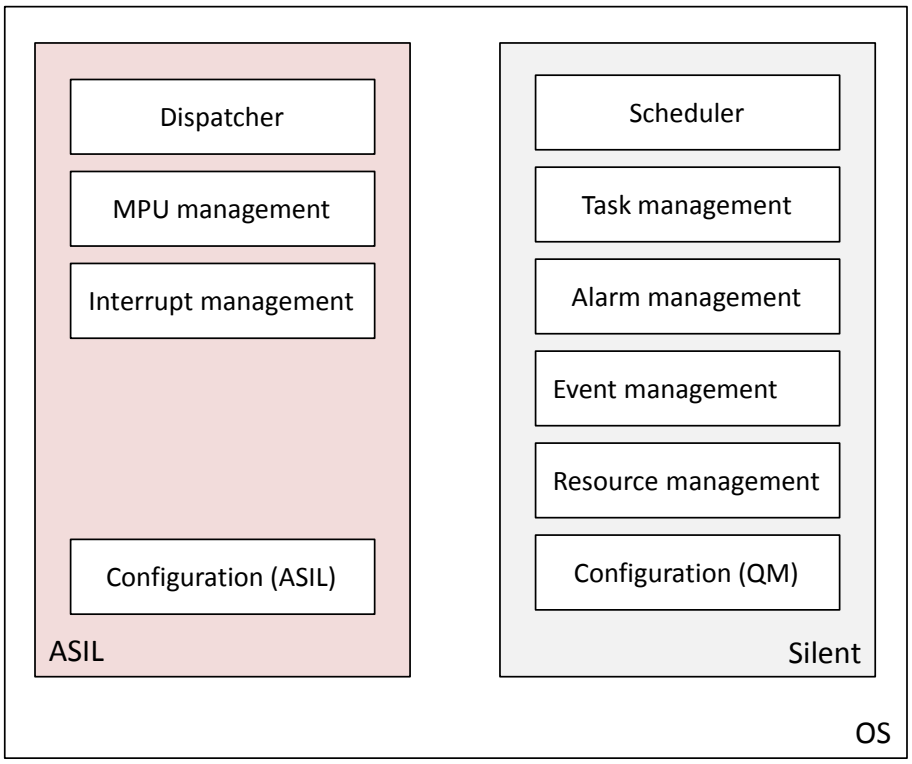

Figure 2-1 Quality Levels of SafeContext Functionalities

AUTOSAR OS is a statically configured operating system. The configuration uses configuration editors and code generators to configure the OS. To ease the safety development the configuration is divided into two parts.

1. Configuration of Safety relevant functionality. The configuration is stored in the ECU in a separate configuration block.
2. Configuration of silent functionality: This configuration is generated and compiled into the OS code.

Additionally some safety relevant configuration parameters are generated and compiled into the OS code. Those have to be reviewed by the user according the rules in this document.

2.4.1 Safety Part

SafeContext provides the following functionality safely:

1. context switching
2. MPU management
3. Interrupt API
4. no unintended overwriting of memory

The *context* is defined as:

- The set of registers, which is used by the compiler
- The stack pointer
- CPU mode (including interrupt state and privilege mode)
- Memory Access Rights

Explanations:

- A context switch occurs in several situations. These are: Task start/switch, ISR entry/exit, call of OS services, call of (Non-)Trusted Functions and Hook routines.

Conclusions:

- If a user program is executed, it will always be executed with the correct context
- After interruptions it is guaranteed that execution is resumed with the correct context
- Freedom from interference with respect to memory will be achieved by using memory protection hardware (e.g. MPU) for non-trusted code.
- Data inconsistency due to race conditions can be prevented by using the interrupt API.

Note

Other OS functionality follows the silent principle. For example the sequence of task executions (scheduling) including the Task pre-emption is provided on QM level.

The operating system provides safe switching of memory access rights during context switches to ensure that non-trusted code does not modify data of other OS-Applications (if not explicitly allowed). In addition the OS interrupts Tasks or ISRs to execute higher priority ISRs. By switching to another context the correct context is set up. By switching back to an interrupted Task or ISR, the correct and unchanged context is restored. To avoid change of a saved context of an interrupted or waiting task, memory protecting hardware is used.

All points in the OS where context switches are performed or are necessary to perform are identified and developed according the safety standard.

At each point in time only one context is active. All other contexts are saved and protected by hardware against accidental alterations.

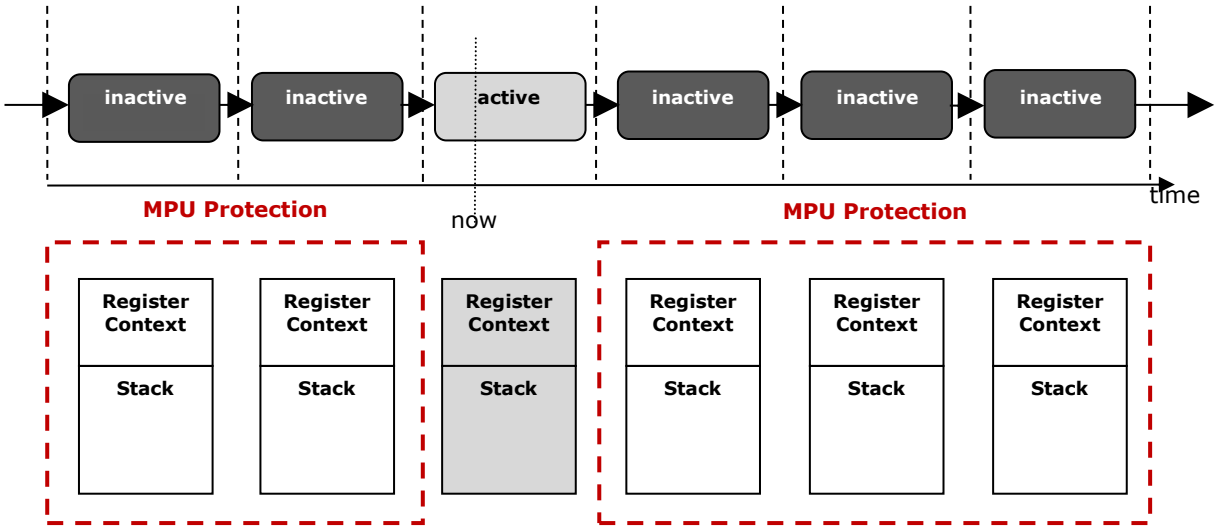

Figure 2-2 Stored and active contexts

2.4.2 Detailed List of Functionality

2.4.2.1 Safety

The following OS services and there functionality is developed according to ASIL.

Class	Description
OS Service APIs	StartOS osInitialize osInitINTC ShutdownOS
	DisableAllInterrupts EnableAllInterrupts SuspendAllInterrupts ResumeAllInterrupts SuspendOSInterrupts ResumeOSInterrupts
	CallTrustedFunction CallNonTrustedFunction
	osReadPeripheral8 osReadPeripheral16 osReadPeripheral32 osWritePeripheral8 osWritePeripheral16 osWritePeripheral32 osModifyPeripheral8 osModifyPeripheral16 osModifyPeripheral32
	osCheckMPUAccess osGetConfigBlockVersion GetApplicationID GetISRID GetTaskID osGetStackUsage osGetSystemStackUsage osGetISRStackUsage
OS System Hooks	StartupHook ErrorHook ProtectionHook ShutdownHook

Table 2-1 Safety Functionality

2.4.2.2 Silent

The following API functions are developed according to Silent principle. SafeContext ensures that they do not violate the assumed safety goal mentioned. Exceptions may be possible, if one of these features has been explicitly ordered as safety relevant. Read the hardware specific part of this document if so.

Class	Description

OS service API	`ActivateTask`
	`TerminateTask`
	`ChainTask`
	`Schedule`
	`GetTaskState`
	`GetResource`
	`ReleaseResource`
	`SetEvent`
	`ClearEvent`
	`GetEvent`
	`WaitEvent`
	`GetAlarm`
	`SetRelAlarm`
	`SetAbsAlarm`
	`CancelAlarm`
	`GetActiveApplicationMode`
	`CheckObjectAccess`
	`CheckObjectOwnership`
	`StartScheduleTableRel`
	`StartScheduleTableAbs`
	`StopScheduleTable`
	`NextScheduleTable`
	`GetScheduleTableStatus`
	`IncrementCounter`
	`GetElapsedValue/GetElapsedCounterValue`
	`GetCounterValue`

Timer handling	The handling of timer hardware is realized as QM software.
Scheduling	The correct sequence of processing application programs is realized with QM-Software (priority handling, including Resources).
ORTI	ORTI is only supported in Silent part of the OS.

Table 2-2 Silent Functionality

2.4.2.3 Not provided

The features listed in the following table are not supported in SafeContext per default. Exceptions may be possible, if one of these features has been explicitly ordered. Read the hardware specific part of this document if so.

Class	Description

OS service API	`TerminateApplication`
	`CheckISRMemoryAccess`
	`CheckTaskMemoryAccess`
	> For ShutdownOS the AUTOSAR OS requirement OS054 is not supported, i.e. non-trusted OS-Applications may successfully call ShutdownOS.
	`StartScheduleTableSynchron`
	`SyncScheduleTable`
	`SetScheduleTableAsync`

COM	OSEK COM inter task communication with messages is not supported
Interrupt resources	Resources are only available at task level.
Protection Reaction	The only allowed protection reaction in the `ProtectionHook` is `PRO_SHUTDOWN`. Other reactions will be interpreted as `PRO_SHUTDOWN`. [SPMF92:0020]
Killing	Forcible terminating Tasks or Applications is not supported.

OS Hooks	`PreTaskHook` (only for debugging!)
	`PostTaskHook` (only for debugging!)
	`ISRHook`
	`PreAlarmHook`

OS Application specific Hooks	`StartupHook<ApplName>`
	`ErrorHook<ApplName>`
	`ShutdownHook<ApplName>`

Address Parameter Check	In case API functions with out-parameters are called with illegal address, they do not return with the error code E_OS_ILLEGAL_ADDRESS as required by the AUTOSAR specification. Instead the out parameter is written with the access rights of the caller, which may lead to a memory protection violation in case the given pointer is invalid.
Stack optimization	Single stack model is not supported.
Internal Resources	Internal Resources are not supported.

Configuration Aspects	The following hooks must be enabled:
	`StartupHook`
	`ErrorHook`
	`ShutdownHook`
	`ProtectionHook`

	Only `SCALABILITYCLASS SC3` or `SC4` is supported. Memory protection must be active.
	`STACKMONITORING` must be enabled.
	`OSInternalChecks` must be configured to `Additional`.
	`ORTIVersion = 2.0` is not supported.
	`ErrorInfoLevel = Modulenames` is not supported.

Table 2-3 Functionality – Not provided

2.5 Safe State

The safe state in SafeContext is shutdown (endless loop with interrupts disabled). The safe state is entered whenever the OS detects a violation of its safety goal or even an attempt. Before the safe state is entered, the `ShutdownHook` is called. The `ShutdownHook` may contain user code which is necessary to reach the defined safe state of the system. This might lead to a reset in combination with a watchdog.

3 Overview of Requirements to the OS User

For integration of the SafeContext into a particular context, the user has the following requirements to be fulfilled. They can be seen as steps to integrate the SEooC in the ECU without harming the assumed safety goal.

The top level requirements are listed in the following table. They are considered in more detail later. If all sub-requirements are checked, you can check the according top level requirement too.

Description of requirements to the OS user	Fulfilled
Check that all assumptions made by SafeContext are valid (see chapter “General SafeContext Assumptions”)	
Check code integrity of the used OS sources (see chapter “OS Source Checksum”)	
Add CRC into the configuration block after linkage (see chapter “Patching the Configuration Block”)	
Check general configuration guidelines (see chapter “General Configuration Guidelines”)	
Review the safety relevant configuration data (see chapter “Review General Part of Configuration Block”)	
Review the generated code (see chapter “Review Generated Code”)	
Review your software (see chapter “Review User Software”)	
Execute MICROSAR Safe Silence Verifier (MSSV) on silent OS part (see chapter “Qualifying Silent OS Part”)	
Check specific requirements to the user (see chapter “Hardware Specific Part”)	

Caution
All requirements listed in this document must be checked and fulfilled by the user!

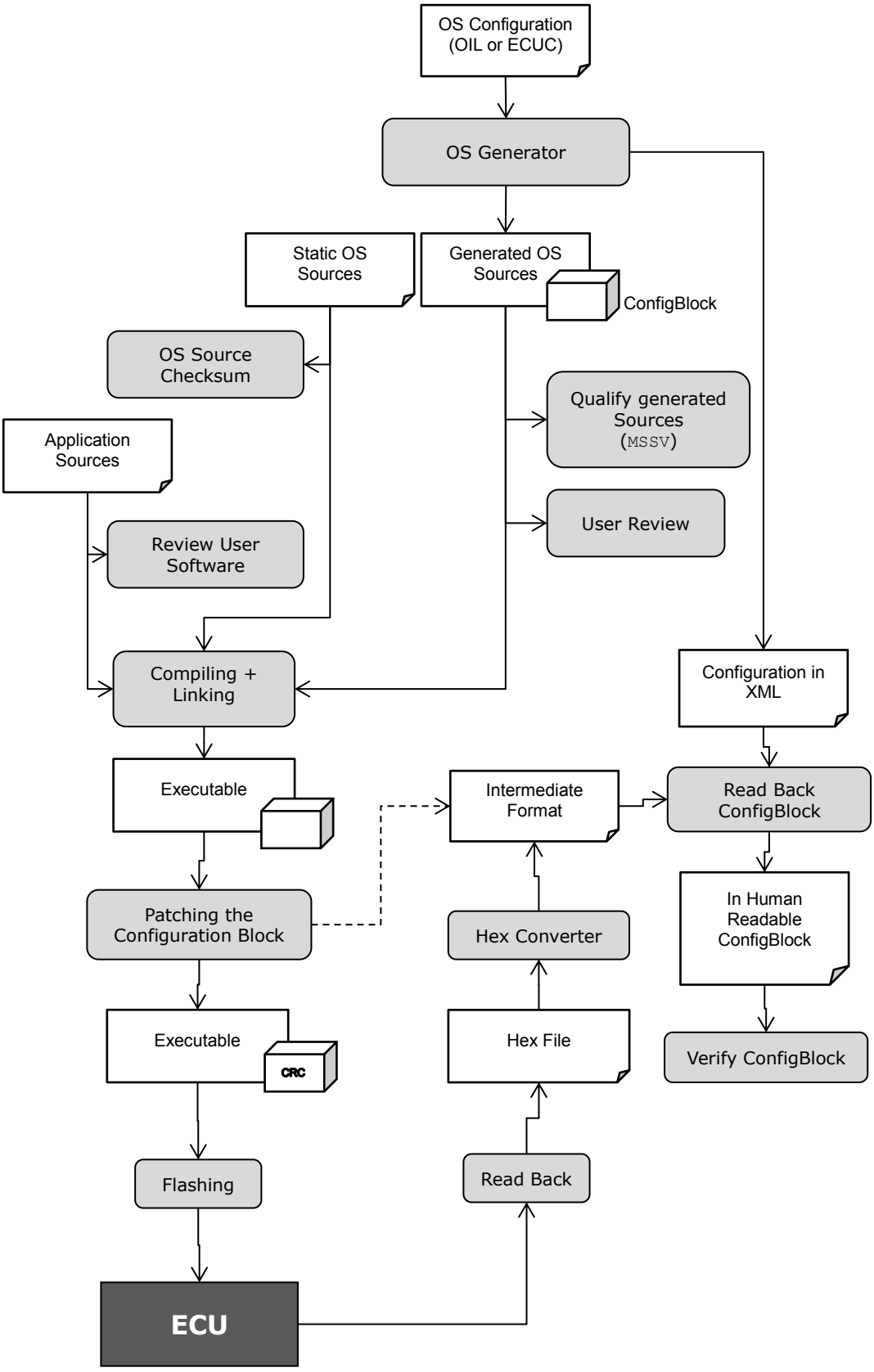

Figure 3-1 Strategy for safety configuration

4 General SafeContext Assumptions

All assumptions must be checked to be true. Assumptions concerning the focus of SafeContext are given by the safety goals and related safety requirements described in the safety case. Assumptions about the environment are described in this chapter.

Description of requirements to the OS user	Fulfilled
Know the SafeContext concept The system safety concept must not rely on OS functionality developed according to the Silent principle. A complete list of API functions and the guarantees they give is provided in chapter “Detailed List of Functionality”. [SPMF92:0075]	
Know your memory configuration Setup of memory sections must be planned by the system designer. Whether or not the planned setup is configured correctly must be verified by reading the configuration back from the ECU and reviewing it against system design and hardware manuals.	
Know the OS specifications The user shall read the OS specifications for OSEK OS and AUTOSAR OS.	
Know how to use the OS The user shall read the OS manuals: > General Technical Reference Manual > Specific Technical Reference Manual Versions are listed in the delivered safety case.	
Correctness of processor The processor provides its functionality with sufficient safety, so that the OS needs not take care about potential hardware failure. This might be assured by usage of a lockstep processor.	
Correctness of memory The memory works with sufficient safety, so that the OS needs not to take care about potential hardware failure.	
Correctness of MPU The MPU provides its functionality with sufficient safety, so that the OS needs not take care about potential hardware failure. The OS provides an API (`osCheckMPUAccess`) which can be used by the user to check the MPU.	
Correctness of hardware manuals The Hardware manuals and the compiler manuals are sufficiently reliable, so that the OS needs not take care about potential deviations between hardware functionality and its description in the manuals. Versions of the used hardware manuals are listed in the delivered safety case. [SPMF92:0017]	
Correctness of compiler tool chain SafeContext assumes that the compiler, assembler and linker generate code with the required safety level.	
Correctness of compiler version and options	

Description of requirements to the OS user	Fulfilled
The used compiler version and options are identical to them which are used during development. Used compiler version and options are listed in the delivered safety case.	
Code integrity The source code and generated configuration of MICROSAR OS SafeContext is compiled, linked and downloaded to the ECU correctly and not modified afterwards. [SPMF92:0043]	
Context definition The user shall not rely on registers, which are not part of the context of the OS. The context definition is listed in chapter 4.1	
Hardware handled by the OS shall not be manipulated by user code User code shall not handle hardware which is handled by the OS. This may include: > Interrupt Controller [SPMF92:0083] > MPU [SPMF92:0085] > Timer	
Don't manipulate short addressing base registers Do not manipulate registers which are used by the compiler for relative addressing of code or data. [SPMF92:0084]	

Table 4-1 General SafeContext Assumptions

4.1 Context Definition

The context which is used by the OS consists of the following registers:

Register	Size in Byte
R1	4
R2	4
R4 R5 … R30 R31	28 * 4 = 112
EIPC	4
EIPSW	4
CTPC	4
CTPSW	4
MPLA0	4
MPUA0	4

5 OS Source Checksum

The OS is delivered as source code. To assure that source code files are not altered after the testing and release a checksum is calculated. The user shall calculate the checksum to verify the correctness of the source code he is using. [SPMF92:0042]

A checksum calculation program (CCodeSafe.exe) is provided to the user. It is called with the following argument:

```
CCodeSafe <config.ini>
```

This tool calculates a CRC32 checksum over a given list of files given in `<config.ini>`.

Description of requirements to the OS user	Fulfilled
Use the delivered source files from Vector! Do not use changed copies in a productive system! Also consider header include order of the compiler.	
The `<config.ini>` shall contain the OS sources listed in the safety case.	
The calculated checksum returned by `CCodeSafe` is identical to the checksum given in the safety case.	

Example
An example for a <config.ini> file:

```
# src directory:
<INSTALLATION_DIRECTORY>\BSW\Os\atosappl.c
<INSTALLATION_DIRECTORY>\BSW\Os\atostime.c
<INSTALLATION_DIRECTORY>\BSW\Os\osek.c
<INSTALLATION_DIRECTORY>\BSW\Os\osekasm.c
<INSTALLATION_DIRECTORY>\BSW\Os\osekalrm.c
<INSTALLATION_DIRECTORY>\BSW\Os\osekerr.c
<INSTALLATION_DIRECTORY>\BSW\Os\osekevnt.c
<INSTALLATION_DIRECTORY>\BSW\Os\osekrsrc.c
<INSTALLATION_DIRECTORY>\BSW\Os\osekstart.c
<INSTALLATION_DIRECTORY>\BSW\Os\osektask.c
<INSTALLATION_DIRECTORY>\BSW\Os\osektime.c
<INSTALLATION_DIRECTORY>\BSW\Os\osSysCallTable.c

# include directory:
<INSTALLATION_DIRECTORY>\BSW\Os\Os.h
<INSTALLATION_DIRECTORY>\BSW\Os\Os_cfg.h
<INSTALLATION_DIRECTORY>\BSW\Os\osek.h
<INSTALLATION_DIRECTORY>\BSW\Os\osekasrt.h
<INSTALLATION_DIRECTORY>\BSW\Os\osekcov.h
<INSTALLATION_DIRECTORY>\BSW\Os\osekerr.h
<INSTALLATION_DIRECTORY>\BSW\Os\osekext.h
<INSTALLATION_DIRECTORY>\BSW\Os\osekasm.h
<INSTALLATION_DIRECTORY>\BSW\Os\osekstart.h
<INSTALLATION_DIRECTORY>\BSW\Os\emptymac.h
<INSTALLATION_DIRECTORY>\BSW\Os\testmac1.h
<INSTALLATION_DIRECTORY>\BSW\Os\vrm.h
<INSTALLATION_DIRECTORY>\BSW\Os\osSysCallTable.dld
```

6 Patching the Configuration Block

Configuration information which is relevant to reach the safety goal is stored in a data structure called the configuration block. The integrity of this information is checked in `StartOS` using a CRC checksum.

The CRC must be calculated after compiling and linking the application. There are two programs provided to calculate and apply the CRC to the binary file:

> `ElfConverter`	For patching the CRC into an ELF file and optionally create an intermediate file for reading back the configuration block.
> `ConfigBlockCRCPatch`	For patching the CRC into an Intel Hex or Motorola SREC file.

The following steps are necessary to patch the configuration block:

1. Compile and link your application
2. Run CRC patch software
3. Write modified executable into flash memory

Description of requirements to the OS user	Fulfilled
Check that CRC is non-zero. If so, change user configuration version to avoid zero CRC. [SPMF92:0071]	

6.1 Using ElfConverter

The program `ElfConverter.exe` is called with the following parameters:

```
ElfConverter <ELF File> <ConfigBlock Symbol> --out <Intermediate File>
```

Parameter	Description
`--out <file>`	Dump the configuration block in intermediate format
`--patch_crc`	Calculate the configuration block's CRC and write it into the ELF file
`--print_header`	Print ELF header
`--print_sections`	Print ELF sections
`--print_symbols`	Print ELF symbols
`--print_hexdump`	Print a hex dump of the configuration block
`--help`	Show help

Table 6-1 ElfConverter parameters

Example
Patching the configuration block in ELF file `testappl.out`:

```
ElfConverter.exe testappl.out _osConfigBlock --out testcfg.hex
--patch_crc
```

6.2 Using ConfigBlockCRCPatch

The program `ConfigBlockCRCPatch.exe` is called with the following parameters:

```
ConfigBlockCRCPatch <HEX File> <Output file> <ConfigBlock Address>
```

Example
Example (convert ELF file prj.out into prj.hex with modified CRC):

```
ConfigBlockCRCPatch prj.out prj.hex 0x00800000
```

The address of the configuration block may be taken from symbol `osConfigBlock` in the linker generated map file. [SPMF92:0016]

7 General Configuration Guidelines

Description of requirements to the OS user	Fulfilled
Non-ASIL user code shall be part of Non-Trusted Applications All non-ASIL user code must be executed by Non-Trusted Applications with no write access to safety relevant data (including stacks) and no read or write access to safety relevant peripherals. [SPMF92:02.0034]	

ASIL user code shall not violate the SafeContext safety goal
All user code, which has access to safety relevant data (including stacks, and OS data) or peripherals, must be implemented on ASIL level. This code shall never violate the safety goals of SafeContext. [SPMF92:0011]
Code which typically has access to safety relevant data (depending on user configuration):
> Trusted Functions [SPMF92:0080] [SPMF92:03.0008]
> Trusted Tasks
> Trusted ISRs
> System Hooks
  > StartupHook [SPMF92:0040] [SPMF92:03.0007]
  > ErrorHook [SPMF92:0012] [SPMF92:03.0006]
  > ProtectionHook [SPMF92:0009] [SPMF92:03.0005]
  > ShutdownHook [SPMF92:0013] [SPMF92:03.0009]
> Reset handler / Startup Code [SPMF92:0005] [SPMF92:03.0001]
> Exception Handlers [SPMF92:0087]
> Category 1 ISRs [SPMF92:0054] [SPMF92:03.0004]	

Alignment of data sections All data sections shall be linked with MPU alignment granularity (e.g. 32 bytes). See the controller's reference manual to know what your MPU granularity is. [SPMF92:0065]	
Consider category 1 ISRs Category 1 ISRs are completely transparent to the OS. The OS does not perform stack switching for category 1 ISRs! Consider this during configuration of stack sizes. [SPMF92:0086]	
NMIs shall be category 1 ISRs Non-maskable Interrupts (NMI) shall be configured to be category 1 ISRs. [SPMF92:0053] [SPMF92:03.0002]	
Link global safety data considering stack growing direction Link global safety data (all OS data and at least ASIL relevant application data) so that it cannot be corrupted by stack overflows (see Figure "Linking example" below for an example). [SPMF92:0091]	

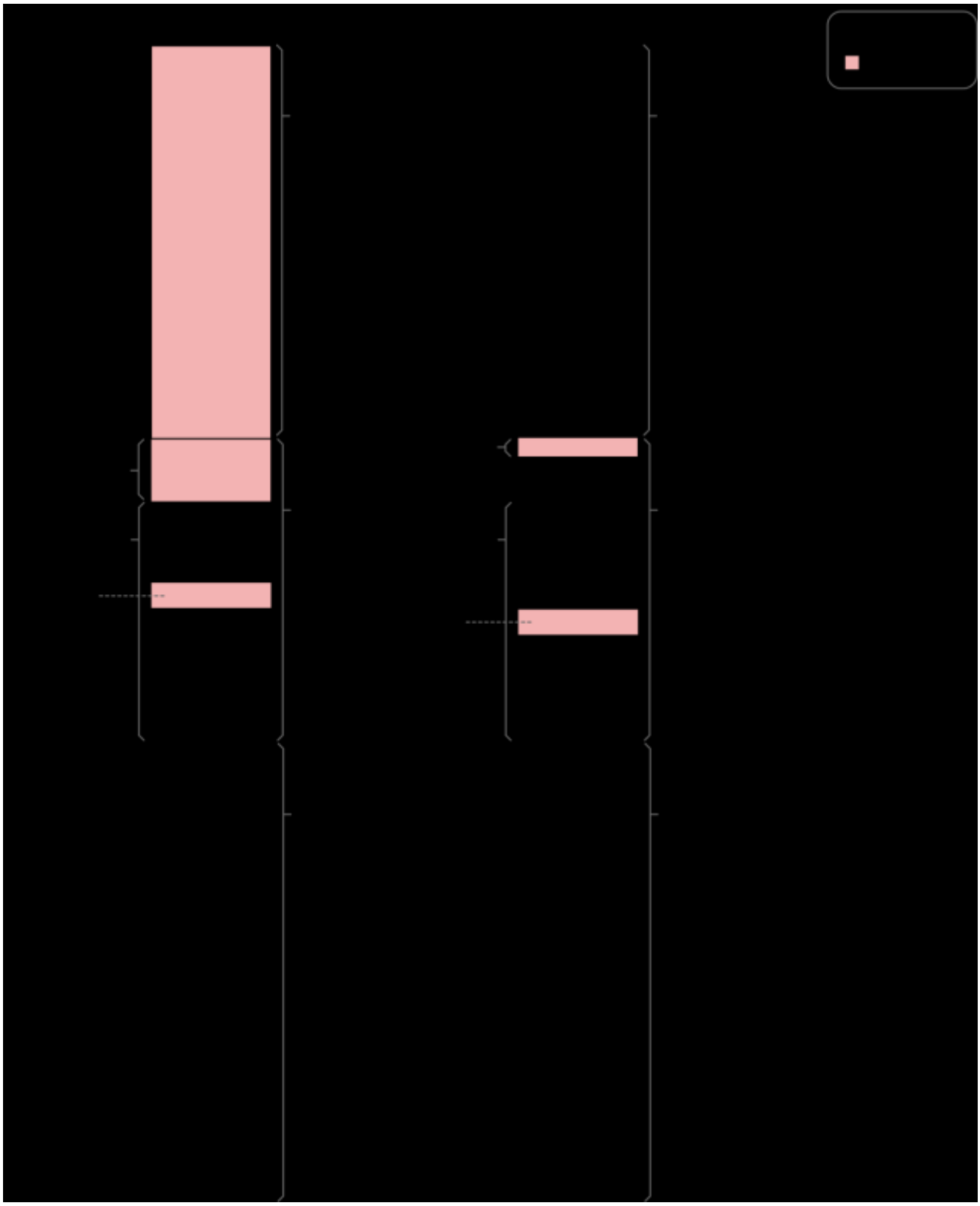

Figure 7-1 Linking example

8 Review General Part of Configuration Block

The configuration of MICROSAR OS SafeContext is generated into C-code. The generator itself has not been developed in accordance to ASIL. Therefore, the generated configuration information needs to be reviewed. The safety relevant configuration is generated into a structure called configuration block (or ConfigBlock). This chapter describes how to review this ConfigBlock. As the ConfigBlock is simply a constant data structure in the flash memory of an ECU, humans will have difficulties to read it. Therefore, the configuration viewer is able to transform the ECU internal representation into a human readable format. The process of reading the ConfigBlock and transforming it into the human readable format is described in the following subchapter. [SPMF92:0038]

The setup of the memory protecting hardware depends on the correct configuration of the OS. All configuration parameters, which are necessary to ensure the safety goal, are stored in a contiguous memory block (configuration block). The configuration block can be located to a fix address and can be read back from the ECU, e.g. by XCP or a debug interface. [SPMF92:0034]

The configuration block is secured by a 16 bit CRC. The way how the configuration block is read back does not need to be safe. The configuration block is translated into a human readable format to allow a review against the intended configuration.

8.1 How to Read Back the Configuration

The configuration is read back in two steps:

1. Create intermediate ConfigBlock file format, by either using
   - `ElfConverter` with parameter `-out`
   - Or read back the ECU binary and convert it into the intermediate format (using the `HexConverter`)
2. Conversion of the intermediate format into human readable output (using the `ConfigViewer`)

The configuration block format is platform dependent. Also this information may be retrieved in different ways, e.g. as the HEX output of the linker or as an upload from the ECU via a protocol like XCP. As this may result in various file formats a conversion into an intermediate format is required.

8.1.1 Using HexConverter

The program `HexConverter.exe` is called with the following parameters:

```
HexConverter –i <Input File> -o <Output File> -b <Base Address>
-s <ConfigBlock Size>
```

All parameters are mandatory.

Parameter	Value	Description
`-i`	Input File Path & Name	File containing the HEX data produced by the linker
`-o`	Output File Path & Name	File with intermediate format generated by HexConverter
`-b`	Base Address	Base address of the configuration block as defined in the linker map file. It is the address of the symbol ‘osConfigBlock’. Value has to be given as HEX (e.g. 0x8001f000).
`-s`	0xFFFF	This value must be at least the size of the configuration block in byte. Bigger values are also allowed. E.g. this value can be set to 0xFFFF

Table 8-1 HexConverter parameters

8.1.2 Using ConfigViewer

The program `ConfigViewer.exe` is called with the following parameters:

```
ConfigViewer –i <Input File> -o <Output File> -c <XML File>
```

Parameter	Value	Description
`-i`	Input File Path & Name	File containing the intermediate data produced by the HexConverter
`-o`	Output File Path & Name	File with human readable configuration description generated by ConfigViewer
`-c`	XML-ConfigFile	XML file, generated by the OS generator describing the OS configuration

Table 8-2 ConfigViewer parameters

The configuration viewer expects the intermediate format to contain nothing else than the configuration block.

8.2 General Configuration Information

The following subchapters concentrate on the output of the configuration viewer and on the rules to review it. The configuration viewer and the format converters are described separately.

To minimize variants in code, the OS generator introduces dummies for each OS object type. These dummies can be identified by the prefix “osSystem” and are handled the same way like other OS objects. None of these objects is active at runtime.

Description of requirements to the OS user	Fulfilled
Check that the listed configuration block address and length is correct and matches the map file.	
Check that the listed configuration block format is 2.00	
Check that the listed version of MICROSAR OS SafeContext matches your delivered version.	
If you are using the user configuration version, check that the listed one matches your configured value.	
Check that the listed number of OS objects matches your configuration. (Consider that dummy OS objects are generated, to minimize variants in the OS code.)	
Check number of listed OS objects matches the elements in the sub containers. [SPMF92:0074]	
Check that the listed specific stack start and end addresses match your configuration. (The stack end addresses in the configuration block point to the first address outside the stack.)	

9 Review Generated Code

MICROSAR OS is a massively configurable software component. As a result, the analysis of the OS modules cannot be completely performed until the user's configuration data is available. The user shall use MICROSAR Safe Silence Verifier (MSSV) to qualify the generated part of the OS, which depends on user's configuration. MSSV is a Vector tool, which performs checks of potential dangerous code constructs. [SPMF92:0049] For more information about MSSV see the Technical Reference Manual of MSSV[6].

Description of requirements to the OS user	Fulfilled
The user must not modify a generated module configuration code file manually unless explicitly required by the technical reference manual or explicitly direction formulated by Vector.	
All generated files of a software project shall be generated based on the same configuration. Generated files of several configurations must not be mixed up unless explicitly allowed by Vector.	
The user shall apply steps for qualifying the generated sources on the final configuration which is used for the production. If the configuration changes, source qualification steps shall be reapplied.	

9.1 Manual Reviews

Some generated code parts currently cannot be checked automatically. Therefore the user has to check them manually.

9.1.1 Review generated file tcb.h

Description of requirements to the OS user	Fulfilled

If interrupt level support is supported in your delivery, you shall review the system level (`osdSystemLevel`) to be the maximum of all category 2 ISR priorities and `osdSystemLevelMask` to be the corresponding value, which has to be stored in PMR register to mask (disable) all category 2 interrupts. [SPMF92:02.0019] [SPMF92:04.0017] [SPMF92:02.0033]
`#define osdSystemLevel <MAXIMUM_OF_ALL_CAT2_ISR_PRIORITIES>`
`#define osdSystemLevelMask <PMR_VALUE_MASK_ALL_CAT2_ISR>`	

Check that osdExceptionDetails is defined = 1 `#define osdExceptionDetails 1`	

[6] TechnicalReference_MSSV.pdf, v1.3

10 Qualifying Silent OS Part

MICROSAR OS is a massively configurable software component. As a result, the analysis of the OS modules cannot be completely performed until the user's configuration data is available. The user shall use MICROSAR Safe Silence Verifier (MSSV) to qualify the generated part of the OS, which dependent on user's configuration. MSSV is a Vector tool, which performs checks of potential dangerous code constructs inside BSW modules which depend on user configuration data. [SPMF92:0049] For more information about MSSV see the Technical Reference Manual of MSSV[7]. [SPMF92:0088]

10.1 Using MICROSAR Safe Silence Verifier (MSSV)

The following chapter tells how you shall apply MSSV on the OS sources.

MSSV is called with the following parameters:

```
MSSV.exe --inputDir <PATH_TO_TCB> --inputDir <PATH_TO_OS_INCLUDE>
--reportFile <REPORT.html>  --define osdNOASM
```

Parameter	Description
<PATH_TO_TCB>	Path to generated OS files (typically contained in the tcb folder).
<PATH_TO_OS_INCLUDE>	Path to OS header files (typically contained in the implementation folder).
<REPORT.html>	File name which shall be used to save the MSSV report.
--define osdNOASM	Disable assembler parts.

Description of requirements to the OS user	Fulfilled
The MICROSAR Safe Silence Verifier shall only be executed on Windows XP SP3+ (32Bit) or Windows 7 (32Bit or 64Bit).	
The user must not modify the MICROSAR Safe Silence Verifier report.	
The user shall verify that the used OS sources are checked by verifying the names and paths of the modules within the report.	
The user shall verify that the evaluated report matches to the execution of the MICROSAR Safe Silence Verifier by verifying the name, creation date and time, path and folder of the report.	
Check that MSSV returns with no errors, no warnings and the final verdict of the report is "pass". If MSSV did not pass contact OS support. [SPMF92:0088]	

[7] TechnicalReference_MSSV.pdf, v1.3

Example

An example for qualifying generated OS sources:

```
MSSV.exe -i "tcb" -i"..\..\include" --define osdNOASM
```

The output of MSSV should look like:

```
note: MICROSAR Safe Silence Verifier Version 1.01.02
note: Copyright (C) 2012-2013 Vector Informatik GmbH
note: License <CBD0900253> VDO AUTOMOTIVE AG

note: MSSV.exe started at 09:39:55 2014-02-19
…
note: MSSV.exe finished at 09:39:57 2014-02-19
note: 0 Errors
note: 0 Warnings
note: the final verdict of the report is 'pass'
```

11 Review User Software

Some code parts run in supervisor mode without any memory protection active or with high memory access granted. Therefore, freedom from interference is not guaranteed by the OS and the hardware. Trusted software has to guarantee freedom from interference on its own. The application programmer typically knows best, what is to do in order to guarantee freedom from interference. Anyhow, there are few additional points to be covered when an OS is used.

The following requirements shall generally be fulfilled by trusted software (also valid for software which runs in supervisor mode or with access to safety relevant memory areas):

Description of requirements to the OS user	Fulfilled
OS code coverage shall be disabled Check that `osdEnableCoverage` is not defined when you compile the OS. [SPMF92:0077]	
FPU usage shall be disabled Check that osdRH850_FPU is **not** defined via compiler option –DosdRH850_FPU Check that osdRH850_FPU is only defined in file osekext.h	
Unhandled exception details shall be enabled Check that osdExceptionDetails is defined = 1 (see tcb.h) Check that OS attribute EnumeratedUnhandledISRs is set TRUE	
No usage of system call instructions in the user software Any system call causes the CPU to change into supervisor mode. Therefore, the application (trusted and non-trusted parts) shall not use system calls directly. Instead, system calls shall only be used by using OS APIs.	
User header usrostyp.h If trusted functions are configured, ensure that `usrostyp.h` does not endanger SafeContext safety requirements.	
Enabling interrupts where it is not allowed Interrupts shall not be enabled by the application where Category 2 ISRs are disabled by default (e.g. in Hooks). This applies not to ISRs. In ISRs it is allowed to enable interrupts. [SPMF92:0066]	
Use only documented APIs Trusted software shall only call documented API functions, which are listed in chapter “Detailed List of Functionality”. [SPMF92:0068]	
Stack usage measurement is not exact Stack usage measurement is implemented by counting magic patterns (`osdStackCheckPattern`) on the stack which have been written there during startup. The returned value may not be correct, if the magic pattern did not change (e.g. the user application uses the same value). [SPMF92:0072]	
Usage of osCheckMPUAccess API If you are using the `osCheckMPUAccess` API, the destination address parameter shall point to an address, where reading and writing does not produce other exceptions than MPU exceptions.	

Description of requirements to the OS user	Fulfilled
Usage of osCheckMPUAccess API If you are using the `osCheckMPUAccess` API, Check that the API function is only called with addresses which, reading and writing does not have any side effects (e.g. potentially not true for peripheral registers). [SPMF92:02.0017].	
If you have write access to stacks, stack overflows cannot be detected by hardware The OS cannot safely detect stack overflows in software which has write access to all stacks. If write access to all stacks is really needed (e.g. for RAM checking), the user has to ensure that the software does not produce a stack overflow! [SPMF92:0078]	
APIs in exception handlers Exception handlers must not call any OS API function beside: > `DisableAllInterrupts` > `EnableAllInterrupts` > `SuspendAllInterrupts` > `ResumeAllInterrupts` > `SuspendOSInterrupts` > `ResumeOSInterrupts` [SPMF92:0067]	
Category 1 ISRs shall be transparent All ISRs of category 1 must be implemented such that they are transparent with respect to the processor state for the code they interrupt. This includes core registers, MPU settings and the current interrupt priority.	
APIs in category 1 ISRs Category 1 ISRs shall not call any OS API function beside: > `DisableAllInterrupts` > `EnableAllInterrupts` > `SuspendAllInterrupts` > `ResumeAllInterrupts` [SPMF92:0067] [SPMF92:02.0018]	
Check out-parameters in Trusted Functions Trusted functions which get a pointer shall check the pointer address to be in an expected range before they write to the pointer address. This shall prevent overwriting of safety relevant data when writing to the pointer address. [SPMF92:0001]	
Check caller in Trusted Functions Trusted functions shall validate whether they are called by an authorized caller only. This may be done by using the API function `GetApplicationID`. [SPMF92:0047]	
No (Non-)Trusted Functions in Hooks Hook routines shall not call any trusted function or non-trusted function.	
No APIs in NMIs Non-maskable interrupts shall not use any OS APIs.	

Description of requirements to the OS user	Fulfilled
[SPMF92:0019], [SPMF92:0067]	
Using Interrupt API before calling StartOS If the user needs to use the interrupt API before he calls `StartOS`, he shall call `osInitialize` and `osInitINTC`. After calling these functions interrupt API works only for the straight forward case. OS error handling and MPU won't be initialized, so the OS won't be able to handle any user errors or detect stack overflows.	

12 Hardware Specific Part

For RH850 SafeContext the following safety relevant requirements must be checked by the user:

- All assembly code (outside the SafeContext) shall be reviewed, not to change the content of registers of R4 and R5 after StartOS is called [SPMF92:04.0001]. Check in list files that only the startup module and the OS modules do modify registers R4 and R5.
- The user has to review that each ISR is called at least once (coverage of application). The tests shall cover the activation of all ISRs and verify that the correct ISR was started. This measure shall prevent the activation of wrong ISRs because of a mix up in the interrupt vector table [SPMF92:0008].
- The user has to review the configuration by means of the ConfigBlock in accordance to the review rules which are defined in chapter 12.2 [SPMF92:0014],[SPMF92:05.0008].
- The user has to review that all libraries fit to the used compiler options. All used libraries need to be checked for using the correct compiler options (e.g. SDA usage need to be identical to the specified options for the OS) [SPMF92:0010].
- The PreTaskHook and the PostTaskHook must not be used in safety code which is released for serial production. Pre/PostTaskHook shall only be used for debugging or test purposes. Absence of Pre/PostTaskHook must be reviewed in generated file tcb.h: [SPMF92:02.0022],[SPMF92:02.0023],[SPMF92:05.0011]

  The user must check that the following defines are set in the generated file tcb.h:

  ```
  #define osdPreTaskHook 0
  #define osdPostTaskHook 0
  ```
- The complete config block content must be reviewed by the means of the BackReader [SPMF92:04.0002], [SPMF92:04.0006].
- The address value of the application specific linker symbols for MPU region start and end address must be checked that between them only the corresponding application data sections are mapped [SPMF92:04.0004], [SPMF92:04.0010], [SPMF92:04.0011].
- The address value of the linker symbols _osGlobalShared_StartAddr and _osGlobalShared_EndAddr must be checked that between them only the global shared data sections are mapped [SPMF92:04.0013].
- The CPU must run in supervisor mode when StartOS is called [SPMF92:04.0007].
- The application shall check the config block version by using OS API function osGetConfigBlockVersion [SPMF92:0045],[SPMF92:05.0010].
- The user has to review that all task stacks, all ISR stacks and the system stack have 4 Byte alignment [SPMF92:04.0008].
- The user has to review the generated linker include files osdata.dld, osrom.dld, ossdata.dld osstacks.dld and ostdata.dld if they are used for serial production [SPMF92:04.0014]. See chapter 12.4.

- The user has to review that coverage is disabled [SPMF92:04.0015]. osdEnableCoverage shall not be defined in header and source files and it shall not be defined via compiler option –DosdEnableCoverage.
- The user has to consider DMA controller usage. The RH850 P1M series incorporates a DMA controller (DMAC). The DMA controller has direct access to the data bus. Therefore DMA access to memory is not controlled by MPU protection. This must be considered especially for safety OS systems if any DMA access is wanted. [SPMF92:01.0002]
- The user has to review that osInitialize and osInitINTC are called before StartOS is called if OS interrupt API functions or OS peripheral interrupt API functions are used before StartOS [SPMF92:0058], [SPMF92:0070].
- Interrupt priority level ceiling is not supported by MICROSAR OS RH850 SafeContext. The user has to check that before StartOS is called the interrupt priority mask register PMR must have the value 0x00000000 and it shall not be changed after StartOS. [SPMF92:0059],[SPMF92:0060]. This is not checked by RH850 SafeContext.
- The user has to review that all generated files belong together [SPMF92:0064]. Each generated header and source file must start with the following comment block:

```
/* file: <file_path><file_name> */
/* automatically generated by genRH850SCTX.exe, Version: 6.11 */
/* from: <configuration_file_name> */
/* Generation time: <date> <time> <year> */
/* <license_and_licensee_information> */
/* Implementation: RenesasRH850_P1M */
/* Version of general code: 6.16 */
/* Dcf-file semantic version: 2.00 */
/* Dcf-file content version: 2.01 */
```

- The user has to review that all generated files are compiled and linked [SPMF92:0064]:
  - intvect.c
  - osConfigBlock.c
  - osStacks.c
  - tcb.c
  - trustfct.c
- The user has to check that the application does not modify interrupt controller registers EBASE, INTBP, INTCFG, SCBP and SCCFG after StartOS is called [SPMF92:0069].
- The application shall not modify any register in interrupt controller unit INTC by own functions or routines after StartOS is called. The application shall only use the OS API functions for changing registers in unit INTC. [SPMF92:0083]
- The user has to check validity and type of the reference parameter when calling the following OS API functions [SPMF92:05.0001]:
  - GetTaskID
  - GetTaskState
  - GetEvent
  - GetAlarm
  - GetScheduleTableStatus
  - GetCounterValue
  - GetElapsedValue/GetElapsedCounterValue

- The user has to review that the size of array oskAlarmHeaps is same as osdNumberOfCounters and that each entry of the array looks like [SPMF92:05.0002],[SPMF92:05.0004]:

```
{
  os<CounterName>Heap,
  &osAlarmHeapCount[<CounterName>]
},
```

- The user has to review that the values of the counter defines are an adjoining set from zero to osdNumberOfCounters-1 in the generated file tcbpost.h [SPMF92:05.0003]:

  example

  tcb.h:

```
#define osdNumberOfCounters 8
```

  tcbpost.h:

```
#define <CounterName0> ((CounterType) 0)
#define <CounterName1> ((CounterType) 1)
#define <CounterName2> ((CounterType) 2)
...
#define <CounterName7> ((CounterType) 7)
```

- The user has to review that the size of the heap arrays os<CounterName>Heap[ ] must be equal to 1 plus the number of alarms that are related to the counter <CounterName> plus the number of schedule tables that are related to the counter <CounterName>. [SPMF92:05.0005]
- The user has to review that the first parameter of all calls of osSysSetEvent in tcb.c is a task identifier which must be defined in tcbpost.h and that the value of the define is smaller than osdNumberOfExtendedTasks. [SPMF92:05.0006]
- The user has to review that all calls of osWorkHeap in Timer ISR look like followed: [SPMF92:05.0007]

```
osWorkHeap(&oskAlarmHeaps[<CounterDefineName>], <CounterDefineName>);
```

- The user has to review that the parameter of each call of osSysActivateTask in tcb.c is a task identifier which must be defined in tcbpost.h [SPMF92:05.0009].
- The user has to review that the values of the task defines are adjoining set from zero to osdNumberOfAllTasks-1 in the generated file tcbpost.h [SPMF92:05.0010].
- The user has to review that the size of the task activation arrays osQTaskActivation_<index> which are listed in oskQActivationQueues is the same value as oskQMaxActivations+1 [SPMF92:05.0012].
- The user has to review that the values of the Alarm defines are adjoining set from zero to osdNumberOfAlarms-1 in the generated file tcbpost.h [SPMF92:05.0013].
- The user has to review that the array oskAlarmCounterRef[] contains exactly osdNumberOfAlarms elements and that each element contains the index of the counter related to that alarm [SPMF92:05.0014].
- The user has to review that the PMR register is not manipulated by his code [SPMF92:04.0018].

12.1 Interrupt Vector Table

Basically the interrupt vector tables must be provided by the application. An example vector table is generated into file intvect.c. This file is generated by QM software and must not be used directly as ASIL code. It must be reviewed carefully for compliance to the description below because the code which is called by the interrupt vector table runs automatically in supervisor mode and therefore this code must be developed according to ASIL level [SPMF92:0004], [SPMF92:02.0020], [SPMF92:02.0021], [SPMF92:04.0012].

The file intvect.c consists of the following parts:

- Header Include Section
- Core Exception Vector Table
- EIINT Vector Table
- CAT2 ISR Wrappers

The following subchapters describe these parts and do intentionally not describe any comments.

12.1.1 Header Include Section

This part of the code must be exactly like:

```
#if defined USE_QUOTE_INCLUDES
 #include "vrm.h"
#else
 #include <vrm.h>
#endif
#define osdVrmGenMajRelNum 6
#define osdVrmGenMinRelNum 11
#if defined USE_QUOTE_INCLUDES
 #include "vrm.h"
#else
 #include <vrm.h>
#endif

#if defined USE_QUOTE_INCLUDES
#include "Os.h"
#else
#include <Os.h>
#endif

#if defined USE_QUOTE_INCLUDES
#include "osekext.h"
#else
#include <osekext.h>
#endif
```

12.1.2 Core Exception Vector Table

The core exception vector table section starts exactly with the following lines:

```
#pragma asm
   .align 512
   .section ".osExceptionVectortable", ax
   .globl _osExceptionVectorTable
_osExceptionVectorTable:
```

That part is followed by 32 vector table entries:

```
   .offset   <offset_addr>
   .globl    _osCoreException_<offset_addr>
_osCoreException_<offset_addr>:
   jr         <handler_function>
```

<offset_addr> is the hexadecimal address offset for each exception interrupt vector.

The valid range is 0x0000, 0x0010, 0x0020 ... 0x01F0

<handler_function> is the name of the function which is called when an exception occurs.

If no handler function is configured then osUnhandledCoreException is called.

The sequence of vector table entries starts at vector address 0x0000, increases in steps of 0x0010 and ends with vector address 0x01F0.

The core exception vector table section ends exactly with the following lines:

```
   .globl _osExceptionVectorTableEnd
_osExceptionVectorTableEnd:
 #pragma endasm
```

12.1.3 EIINT Vector Table

The EIINT vector table section starts exactly with the following lines:

```
#pragma asm
   .align 512
   .section ".osEIINTVectortable", ax
   .globl _osEIINTVectorTable
_osEIINTVectorTable:
```

For each unused interrupt source the following table entry must be generated:

```
   .word     _osUnhandledEIINT_<index>
```

<index> is the channel index of the corresponding interrupt source. The valid range is 0 ... 383.

For each interrupt source used as category 1 ISR the following table entry must be generated:

```
   .word     _<cat1_EIINT_handler>
```

<cat1_EIINT_handler> is the name of the application specific EIINT handler which is called when an interrupt on the corresponding source occurs.

For each interrupt source used as category 2 ISR the following table entry must be generated:

```
   .word     _<cat2_EIINT_handler>_CAT2
```

<cat2_EIINT_handler> is the name of the OS ISR wrapper which is called when an interrupt on the corresponding source occurs. The CAT2 ISR wrapper section is described in the next chapter.

The EIINT vector table section ends exactly with the following lines:

```
   .globl _osExceptionVectorTableEnd
_osExceptionVectorTableEnd:
 #pragma endasm
```

12.1.4 CAT2 ISR Wrappers

The category 2 ISR wrapper section starts exactly with the following line:

```
#pragma ghs section text=".os_text"
```

Each category 2 ISR handler must be generated exactly like the following line [SPMF92:0044]:

```
osCAT2ISR(<ISR_Function_Name>, <ISR_Priority_Level>)
```

This macro is defined in osek.h. It defines the CAT2 ISR prologue for each interrupt priority level.

<ISR_Function_Name> is the unique name of the ISR function. This must be the same name as used in the EIINT vector table at the corresponding channel index position with postfix _CAT2.

<ISR_Priority_Level> is the value of the interrupt priority level which is configured for the corresponding ISR. The valid range of <ISR_Priority_Level> is 0 ... 15

The category 2 ISR wrapper section ends exactly with the following line:

```
#pragma ghs section text=default
```

12.2 Configuration Block

This chapter defines the rules to review and interpret the entries of the configuration block.

The configuration of MICROSAR OS RH850 SafeContext is generated into C-Code. The generator itself has not been developed in accordance to safety requirements. Therefore, the generated configuration information needs to be reviewed. The safety relevant configuration is generated into a structure called ConfigBlock. This chapter describes how to review the ConfigBlock. As the ConfigBlock is simply a constant structure in the memory of an ECU, users will have difficulties to read it. Therefore, the configuration viewer is able to transform the ECU internal representation into a user readable format. The process of reading the ConfigBlock and transforming it into the user readable format is described in the following subchapter.

12.2.1 How to read back the ConfigBlock

The ConfigBlock is internal information of a program to control an ECU. It might be available in different formats. Therefore it is difficult to provide one single configuration viewer program to read the information and transform it into a human readable format.

So the configuration viewer is a console application which reads configuration information from a proprietary intermediate file and writes it into a text file. Different reader programs are available to transform the ConfigBlock from standard formats into the intermediate format. In addition, the intermediate format is quite simple, so it may also be produced by manual adaptation of a debugger output (hex dump) with any text editor.

The configuration viewer expects the intermediate format to contain nothing else than the ConfigBlock. Therefore, the transformation programs need the information, where the ConfigBlock is located. That information is passed to the transformation program by means of the parameter ‘–b’.

The address of the ConfigBlock is easily taken out of the linker map file. It is the address of the constant *osConfigBlock* [SPMF92:0016].

The following subchapters concentrate on the output of the configuration viewer and on the rules to review it. The configuration viewer and the format converters are described separately.

12.2.2 Additional Information

The operating system and therefore also the ConfigBlock typically use indexes instead of the names of tasks, ISRs, applications and so on. Each index value is related to exactly one object in the configuration of the OS. However, the relation to the configured object is typically not so obvious.

Therefore, the configuration viewer provides the possibility to add the names of configuration objects to its output. That information is taken from the so called XML config-file. The OS-generator produces this file together with the source files it generates.

The configuration viewer outputs the object names in case a parameter is set to use the information from the XML config-file. All information taken from the XML config-file is represented between ‘(‘ and ‘)’ to mark it as unsafe additional information.

The reviewer of the ConfigBlock has the possibility to make use of the information taken from the XML config-file or to skip that information completely. The review rules in the chapters below describe for both cases how the review shall be performed. The examples of configuration viewer output below always contain the information from XML config-file.

Although the information taken from the XML config-file (information in parentheses) is unsafe, the relation between index and name of an object is guaranteed to be fix. This means, once the configuration viewer has read in the object names from the XML config-file, it always translates each certain index to the exact same object name. So the relation between object index and object name needs to be reviewed only once and can be seen as reliable everywhere else in the configuration viewer output.

The review rules below already contain rules to validate the information taken from XML config-file, so that information can safely be used although it is taken from an unsafe source.

12.2.3 How to start the review

The configuration viewer starts the output with something like [SPMF92:0063]:

```
OS Configuration Viewer V3.00 (General) V1.02 (HW Specific) (Apr 24 2014 15:42:59)
(c) 2013 Vector Informatik GmbH

Started at:Tue May 13 16:23:24 2014

Remark: Text between '(' and ')' shall be treated as unsafe additional information

=====================================================
Start of Config Block                       0x00002000
Length                                      588
CRC                                         0x9622
Config Block Format Version                 02.00
MICROSAR OS RH850  SafeContext Version      06.05
User Config Version                         2
=====================================================
```

The user should start with some consistency checks like:

- Is the date (Started at) equal to the date/time when the configuration viewer was started
- Do the start address (Start of configuration block), length (Length) and CRC fit to the configuration block which shall be reviewed
- Check that the listed version of MICROSAR OS RH850 SafeContext matches your delivered version.
- Review that the user config version represents the number which was configured by means of the configuration attribute UserConfigurationVersion [SPMF92:0045].
- Is the output complete (see below)?

The output is complete when it ends on:

```
=====================================================
          End of configuration block reached
=====================================================
```

12.2.3.1 Indexes of applications, task , ISRs, trusted and non-trusted functions

Check indexes of listed applications:
The indexes must begin with 0
The indexes must be consecutive. Gaps are not allowed
The indexes must end at <Number of applications>-1

Check indexes of listed tasks:
The indexes must begin with 0
The indexes must be consecutive. Gaps are not allowed
The indexes must end at <Number of tasks>-1

Check indexes of listed category 2 ISRs:
The indexes must begin with 0
The indexes must be consecutive. Gaps are not allowed
The indexes must end at <Number of category 2 ISRs> - 1

Check indexes of listed trusted functions:
The indexes must begin with 0
The indexes must be consecutive. Gaps are not allowed
The indexes must end at <Number of trusted functions> - 1

Check indexes of listed non-trusted functions:
The indexes must begin with 0
The indexes must be consecutive. Gaps are not allowed
The indexes must end at <Number of non-trusted functions> - 1

Note: The table items in the configuration viewer output may not be sorted by index.

12.2.3.2 Review against User’s Design

The chapters below describe, how the configuration viewer output can be reviewed against the OS-configuration. However, the review shall be performed against the user’s design of the system.

The reason is that OS developers can only describe, where the respective information can be found in the configuration but have only limited knowledge about the system design.

12.2.4 How to review the general information (block 0)

The configuration viewer generates the general information as followed [SPMF92:0063]:

```
0. General Information
-------------------------------------------------------
Property                          Value       Checked?
-------------------------------------------------------
Number of Tasks                   4             [  ]
Number of Cat2 ISRs               3             [  ]
Number of All ISRs                4             [  ]
Number of Trusted Functions       1             [  ]
Number of Non-Trusted Functions   0             [  ]
Number of Applications            3             [  ]
Number of Peripheral Regions      0             [  ]
Stack Usage Measurement           Yes           [  ]
Number of MPU Regions             12            [  ]
Number of Dynamic MPU Regions     2             [  ]
Number of Static MPU Regions      4             [  ]
Number of Available Cores         1             [  ]
System Stack Start-Address        0xfedf04b0    [  ]
System Stack End-Address          0xfedf0640    [  ]
-------------------------------------------------------
```

The review shall be performed like this:

- Count the number of configured tasks and compare it against the number of tasks presented by the configuration viewer in the part about general information
- Check the number of Cat2 ISRs:
  - Count the number of configured ISRs with CATEGORY = 2 and compare it against the number of Cat2 ISRs presented by the configuration viewer in the part about general information.
  - Consider that the OS internally uses a further Cat2 ISR for alarm and schedule table handling. That ISR is only available in case at least one alarm or a schedule table has been configured that uses the system timer.
- Check the number of all ISRs:
  - Count the number of configured EIINT ISRs with CATEGORY = 1 and 2 and compare it against the number of all ISRs presented by the configuration viewer in the part about general information.
- Count the number of configured trusted functions and compare it against the number of trusted functions presented by the configuration viewer in the part about general information. Trusted function configurations may be found in the configuration of any OS-applications with TRUSTED = TRUE.
- Count the number of configured non-trusted functions and compare it against the number of non-trusted functions presented by the configuration viewer in the part about general information. Non-trusted function configurations may be found in the configuration of any OS-applications with TRUSTED = FALSE.
- Count the number of configured OS-applications and compare it against the number of OS-applications presented by the configuration viewer in the chapter about general information.
- Count the number of peripheral regions configured over all non-trusted applications and compare it against the number of peripheral regions presented by the configuration viewer.
- Compare the configuration of stack usage measurement against the respective output of the configuration viewer in the chapter about general information. The configuration of stack usage measurement is found in the general configuration of the OS.

- Check the number of MPU regions which are provided by the used CPU derivative.
- Check the number of dynamic MPU regions which must be same as the number of MPU regions configured in the application with most dynamic MPU regions.
- Check the number of static MPU regions configured for the OS.
- Check the number of CPU cores which are provided by the used CPU derivative.
- Review of system stack addresses [SPMF92:0056],[SPMF92:04.0003]:
  - Compare the system stack start address in the configuration viewer output against the label *_osSystemStack_StartAddr* in the linker map-file.
  - Compare the system stack end address in the configuration viewer output against the label *_osSystemStack_EndAddr* in the linker map-file.
  - Check that both labels are 4 Byte aligned [SPMF92:04.0002]. This prevents that any data of other sections is accessible by the same MPU region.
  - Check that only the array variable osSystemStack is located between the labels described above.
  - Check the difference between the system stack start- and end-address against the designed (and therefore also configured) system stack size.

12.2.5 How to review the task start addresses (block 1)

The configuration viewer generates the task start addresses as followed [SPMF92:0063]:

```
1. Task start addresses:
-------------------------------------------------------
Task-ID  Task-Name                Value       Checked?
-------------------------------------------------------
0        (eTask1)                 0x000030cc    [  ]
1        (osSystemExtendedTask)   0x000059a2    [  ]
2        (bTask1)                 0x00003052    [  ]
3        (osSystemBasicTask)      0x000059a0    [  ]
-------------------------------------------------------
```

The review shall be performed like this:

- Check the linker-map-file of the project for a function called `<TaskName>func`. This function must be linked to the address value as specified in the configuration viewer output [SPMF92:02.0025].

12.2.6 How to review the task trusted information (block 2)

The configuration viewer generates the task trusted information as followed:
[SPMF92:0061],[SPMF92:0063], [SPMF92:02.0027]

```
2. Task trustedness information:
-------------------------------------------------------
Task-ID  Task-Name                Value         Checked?
-------------------------------------------------------
0        (eTask1)                 non-trusted     [  ]
1        (osSystemExtendedTask)   trusted         [  ]
2        (bTask1)                 non-trusted     [  ]
3        (osSystemBasicTask)      trusted         [  ]
-------------------------------------------------------
```

The review shall be performed like this:

- Check for each task, that the setting of the attribute TRUSTED of the owner application fits to the value of the task trustedness information. In case the owner application has the configuration TRUSTED=TRUE, the task must be trusted. In case the owner application has the configuration TRUSTED=FALSE, the task must be non-trusted.
- osSystemExtendedTask and osSystemBasicTask must always be trusted.

12.2.7 How to review the task preemptive information (block 3)

The configuration viewer generates the task preemptive information as followed [SPMF92:0063]:

```
3. Task preemptiveness information:
-------------------------------------------------------------
Task-ID  Task-Name                Value               Checked?
-------------------------------------------------------------
0        (eTask1)                 fully-preemptible     [  ]
1        (osSystemExtendedTask)   fully-preemptible     [  ]
2        (bTask1)                 fully-preemptible     [  ]
3        (osSystemBasicTask)      non-preemptible       [  ]
-------------------------------------------------------------
```

The review shall be performed like this:

- Take the task name (as printed) and check whether this respective task preemptive setting was really configured as printed. If preemptive is printed here, the task configuration must contain SCHEDULE = FULL, if non-preemptive is printed here, the task configuration must contain SCHEDULE=NON. The OS generator assures that each task has the attribute SCHEDULE exactly once with possible values FULL or NON.

12.2.8 How to review the task stack start and end addresses (block 4 and 5)

The configuration viewer generates the task stack start addresses as followed [SPMF92:0063]:

```
4. Task stack lower boundary addresses:
-------------------------------------------------------
Task-ID  Task-Name                Value       Checked?
-------------------------------------------------------
0        (eTask1)                 0xfedf0640    [  ]
1        (osSystemExtendedTask)   0x00000010    [  ]
2        (bTask1)                 0xfedf07d0    [  ]
3        (osSystemBasicTask)      0x00000010    [  ]
-------------------------------------------------------
```

The configuration viewer generates the task stack end addresses as followed:

```
5. Task stack upper boundary addresses:
-------------------------------------------------------
Task-ID  Task-Name                Value       Checked?
-------------------------------------------------------
0        (eTask1)                 0xfedf07d0    [  ]
1        (osSystemExtendedTask)   0x00000000    [  ]
2        (bTask1)                 0xfedf0960    [  ]
3        (osSystemBasicTask)      0x00000000    [  ]
-------------------------------------------------------
```

The review shall be performed like this [SPMF92:0056]:

- Check that the difference between stack start and stack end address fits to the configured size of the task stack
- Check that the stacks do not overlap
- Compare printed address value of each stack with the address given in the linker map file.
- Check that all task stack sizes are a multiple of 4 Bytes [SPMF92:04.0008]. This prevents that any data of another section is accessible in the same MPU region.
- Check that each defined address region (task stack start address, task stack end address) only covers exactly one array variable named osTaskStack<StackIndex>.

12.2.9 How to review the task ownership information (block 6)

The configuration viewer generates the task ownership information as followed [SPMF92:0063]:

```
6. Task to application mapping:
-------------------------------------------------------------------------------
Task-ID  Task-Name               Appl-ID  Appl-Name                   Checked?
-------------------------------------------------------------------------------
0        (eTask1)                2        (xyzAppl)                     [  ]
1        (osSystemExtendedTask)  1        (osSystemApplicationCore0)    [  ]
2        (bTask1)                2        (xyzAppl)                     [  ]
3        (osSystemBasicTask)     1        (osSystemApplicationCore0)    [  ]
-------------------------------------------------------------------------------
```

The review shall be performed like this [SPMF92:0062]:

- The configuration of each application contains a list of the tasks which it owns. Check for each of the named applications that it owns exactly those tasks listed in the configuration viewer output. The OS generator assures that each task has exactly one owner application.

12.2.10 How to review the category 2 ISR start addresses (block 7)

The configuration viewer generates category 2 ISR start addresses as followed [SPMF92:0063]:

```
7. Category 2 ISR function start address:
-------------------------------------------------
ISR-ID  ISR-Name             Value       Checked?
-------------------------------------------------
0       (ISR1)               0x000030f4    [  ]
1       (osSystemCat2ISR)    0x000059a4    [  ]
2       (osTimerInterrupt)   0x00009b9c    [  ]
-------------------------------------------------
```

The review shall be performed like this:

- Check the linker map file of the project for symbol _<ISRName>func. This ISR function must be linked to the address value as specified in the configuration viewer output [SPMF92:02.0026].
- In case the ISR has a configured SpecialFunctionName, search for the symbol _<SpecialFunctionName>func instead, which is located at the address, shown in the configuration block.

12.2.11 How to review the CAT2 ISR trusted information (block 8)

The configuration viewer generates CAT2 ISR trusted information as followed: [SPMF92:0061],[SPMF92:0063], [SPMF92:02.0027]

```
8. Category 2 ISR trustedness:
-------------------------------------------------
ISR-ID  ISR-Name            Value         Checked?
-------------------------------------------------
0       (ISR1)              non-trusted     [  ]
1       (osSystemCat2ISR)   trusted         [  ]
2       (osTimerInterrupt)  trusted         [  ]
-------------------------------------------------
```

The review shall be performed like this:

- Check for each cat2 ISR, that the setting of the attribute TRUSTED of the owner application fits to the value of the ISR trusted information. In case the owner application has the configuration TRUSTED=TRUE, the ISR must be trusted. In case the owner application has the configuration TRUSTED=FALSE, the ISR must be non-trusted. The trustedness of the system timer ISR osTimerInterrupt cannot be configured and is always a trusted ISR.

12.2.12 How to review the CAT2 ISR nested information (block 9)

The configuration viewer generates the CAT2 ISR nested information as followed [SPMF92:0063]:

```
9. Category 2 ISR nesting:
-------------------------------------------------
ISR-ID  ISR-Name             Value       Checked?
-------------------------------------------------
0       (ISR1)               nested        [  ]
1       (osSystemCat2ISR)    non-nested    [  ]
2       (osTimerInterrupt)   nested        [  ]
-------------------------------------------------
```

The review shall be performed like this:

- Check for each cat2 ISR, that the setting of the attribute EnableNesting fits to the value of the ISR nested information [SPMF92:02.0031]. In case the cat2 ISR has the configuration EnableNesting =TRUE, the ISR must be nested. In case the cat2 ISR has the configuration EnableNesting =FALSE, the ISR must be not nested. This setting for the system timer ISR osTimerInterrupt is configured via OS attribute SystemTimerNestable. Check that the setting of attribute SystemTimerNestable fits to the ISR nested information of ISR osTimerInterrupt.

12.2.13 How to review CAT2 ISR stack start and end addresses (block 10 and 11)

The configuration viewer generates the CAT2 ISR stack start addresses as followed [SPMF92:0063]:

```
10. Category 2 ISR stack start address:
----------------------------
Level  Value       Checked?
----------------------------
0      0x00000000    [  ]
1      0x00000000    [  ]
2      0x00000000    [  ]
3      0x00000000    [  ]
4      0x00000000    [  ]
5      0x00000000    [  ]
6      0x00000000    [  ]
7      0x00000000    [  ]
8      0x00000000    [  ]
9      0x00000000    [  ]
10     0xfedf0000    [  ]
11     0x00000000    [  ]
12     0xfedf0320    [  ]
13     0x00000000    [  ]
14     0x00000000    [  ]
15     0x00000000    [  ]
----------------------------
```

The configuration viewer generates the CAT2 ISR stack end addresses as followed [SPMF92:0063]:

```
11. Category 2 ISR stack end address:
----------------------------
Level  Value       Checked?
----------------------------
0      0x00000000    [  ]
1      0x00000000    [  ]
2      0x00000000    [  ]
3      0x00000000    [  ]
4      0x00000000    [  ]
5      0x00000000    [  ]
6      0x00000000    [  ]
7      0x00000000    [  ]
8      0x00000000    [  ]
9      0x00000000    [  ]
10     0xfedf0320    [  ]
11     0x00000000    [  ]
12     0xfedf04b0    [  ]
13     0x00000000    [  ]
14     0x00000000    [  ]
15     0x00000000    [  ]
----------------------------
```

The review shall be performed like this [SPMF92:0056]:

- Check that the difference between stack start and stack end address fits to the configured size of the ISR stack
- Check that the stacks do not overlap
- Compare printed address value of each stack with the address given in the linker map file.

- Check that all ISR stack sizes are a multiple of 4 Bytes [SPMF92:04.0008]. This prevents that any data of another section is accessible in the same MPU region.
- Check that each defined address region (ISR stack start address, ISR stack end address) only covers exactly one array variable named osIntStackLevel<PriorityLevel>.

12.2.14 How to review the CAT2 ISR ownership information (block 12)

The configuration viewer generates the CAT2 ISR ownership information as followed [SPMF92:0063]:

```
12. Category 2 ISR to application assignment:
-------------------------------------------------------------------------
ISR-ID  ISR-Name            Appl-ID  Appl-Name                   Checked?
-------------------------------------------------------------------------
0       (ISR1)              2        (xyzAppl)                     [  ]
1       (osSystemCat2ISR)   1        (osSystemApplicationCore0)    [  ]
2       (osTimerInterrupt)  1        (osSystemApplicationCore0)    [  ]
-------------------------------------------------------------------------
```

The review shall be performed like this:

- The configuration of each application contains a list of the cat2 ISRs which it owns. Check for each of the named applications that it owns exactly those ISRs listed in the configuration viewer output [SPMF92:02.0028]. The OS generator assures that each ISR has exactly one owner application. The owner of the ISR osTimerInterrupt is the OS itself.
- Check that category 1 ISRs are not listed in the configuration block. [SPMF92:0055]

12.2.15 How to review the trusted functions start addresses (block 13)

The configuration viewer generates trusted functions start addresses as followed [SPMF92:0063]:

```
13. Trusted function start address:
-----------------------------------------------------------
Tfunc-ID  Tfunc-Name                  Value       Checked?
-----------------------------------------------------------
0         (StartTestStep)             0x000039d8    [  ]
1         (GetTestStep)               0x000039aa    [  ]
2         (SetCheckPoint)             0x000039c2    [  ]
3         (TF1)                       0x00003006    [  ]
4         (TriggerISR1)               0x000039ee    [  ]
5         (TriggerISR2)               0x00003a06    [  ]
6         (osSystemTrustedFunction)   0x00005982    [  ]
-----------------------------------------------------------
```

The review shall be performed like this:

- Check the linker map file of the project for symbol TRUSTED_<FuncName>. This function must be linked to the address value as specified in the configuration viewer output [SPMF92:0048],[SPMF92:0057].

12.2.16 How to review the non-trusted functions start addresses (block 14)

The configuration viewer generates non-trusted functions start addresses as followed [SPMF92:0063]:

```
14. Non-trusted function start address:
-----------------------------------------
NTfunc-ID NTfunc-Name Value      Checked?
-----------------------------------------
0         (NTF1)      0x00002e6a   [ ]

1         (NTF2)      0x00003de0   [ ]
-----------------------------------------
```

The review shall be performed like this:

- Check the linker map file of the project for symbol NONTRUSTED_<FuncName>. This function must be linked to the address value as specified in the configuration viewer output [SPMF92:02.0024].

12.2.17 How to review the non-trusted functions ownership information (block 15)

The configuration viewer generates the non-trusted functions ownership as followed [SPMF92:0063]:

```
15. Non-trusted function to application assignment:
-----------------------------------------------------------
NTfunc-ID NTfunc-Name Appl-ID Appl-Name          Checked?
-----------------------------------------------------------
0          (NTF1)       1       (NonTrustedApplA)    [ ]

1          (NTF2)       2       (NonTrustedApplB)    [ ]
-----------------------------------------------------------
```

The review shall be performed like this:

- The configuration of each application contains a list of the functions which it owns. Check for each of the named applications that it owns exactly those functions listed in the configuration viewer output [SPMF92:02.0029]. The OS generator assures that each function has exactly one owner application.

12.2.18 How to review the application dynamic MPU settings (block 16)

It is necessary to refer the Renesas Electronics user's manual architecture [6] for exactly understanding of the Memory Protection Unit (MPU).

The configuration viewer generates the dynamic MPU settings as followed [SPMF92:0063]:

```
16. Application MPU configuration (dynamic):
-------------------------------------------------------------------------------
Appl-ID  Appl-Name                    Region  Start-Addr  End-Addr    Checked?
-------------------------------------------------------------------------------
0        (AppTrusted)                1       0x00000010  0x00000000    [  ]
                                     2       0x00000010  0x00000000    [  ]
-------------------------------------------------------------------------------
1        (NonTrustedApplA)           1       0xffe50000  0xffe5fffc    [  ]
                                     2       0xfedf0a80  0xfedf0a7f    [  ]
-------------------------------------------------------------------------------
2        (NonTrustedApplB)           1       0xffe50000  0xffe5fffc    [  ]
                                     2       0xfedf0a80  0xfedf0b7f    [  ]
-------------------------------------------------------------------------------
3        (TraceAppl)                 1       0x00000010  0x00000000    [  ]
                                     2       0x00000010  0x00000000    [  ]
-------------------------------------------------------------------------------
4        (osSystemApplicationCore0)  1       0x00000010  0x00000000    [  ]
                                     2       0x00000010  0x00000000    [  ]
-------------------------------------------------------------------------------
```

The review shall be performed like this [SPMF92:0052],[SPMF92:04.0003],[SPMF92:04.0005]:

- Each application must have the same number of rows.
- The number of rows for each application must be same as shown in the output of configuration block 0: Number of Dynamic MPU Regions
- Trusted applications must always have Start-Addr = 0x00000010
- Trusted applications must always have End-Addr = 0x00000000
- Unused MPU regions in non-trusted applications must have Start-Addr = 0x00000010
- Unused MPU regions in non-trusted applications must have End-Addr = 0x00000000
- Check that values of Start-Addr and End-Addr in row of non-trusted applications are same as configured in the applications settings
- Check that used MPU regions in non-trusted applications have Start-Addr value smaller than the End-Addr value
- Check that all trusted and non-trusted applications are listed.
- Check that all Start-Addr values are aligned to 4 Byte boundary [SPMF92:04.0009]
- Check that all End-Addr values point to the last valid byte in the specified area.
- Overlapping of memory regions is not allowed [SPMF92:04.0010].
- The next region after the end address must be aligned at to a 4 Byte boundary.
- Compare Start-Addr of non-trusted applications that the address value fits to address of the application specific linker symbol used in configuration settings [SPMF92:0052]
- Compare End-Addr of non-trusted applications that the address value fits to address of the application specific linker symbol used in configuration settings [SPMF92:0052]
- If a linker section for application data memory region is empty then the end address is below the start address. The start or end address may then overlap with other linker sections. This can be ignored because it does not harm the MPU functionality.

12.2.19 How to review the interrupt channel index (block 17)

The configuration viewer generates the interrupt channel index for CAT1 and CAT2 ISRs as followed [SPMF92:0063]:

```
17. Category 2 ISR interrupt channel index:
-------------------------------------------
ISR-ID  ISR-Name            Channel  Checked?
-------------------------------------------
0       (ISR1)              137        [  ]
1       (osSystemCat2ISR)   0          [  ]
2       (osTimerInterrupt)  74         [  ]
3       (TestTimer1)        134        [  ]
-------------------------------------------
```

The review shall be performed like this:

- Check for each cat1 and cat2 ISR that the number is corresponding to the default priority number listed in the hardware user manual of the used processor derivative (look for symbol EIINT<Number>).
- Check that the biggest listed number is not greater than 383.
- Channel index of ISR osSystemCat2ISR can be ignored because it is not used at all.

12.2.20 How to review the ISR interrupt priority level (block 18)

The configuration viewer generates the interrupt priority level for CAT1 and CAT2 ISRs as followed [SPMF92:0063]:

```
18. Category 2 ISR priority level:
-----------------------------------------------
ISR-ID  ISR-Name             Priority  Checked?
-----------------------------------------------
0       (ISR1)               10          [  ]
1       (osSystemCat2ISR)    128         [  ]
2       (osTimerInterrupt)   12          [  ]
3       (TestTimer1)         7           [  ]
-----------------------------------------------
```

The review shall be performed like this [SPMF92:02.0032]:

- Check for each category 1 and category 2 ISR that the setting of the attribute InterruptPriority fits to the value of the printed ISR information “Priority” from configuration viewer.
- Interrupt priority level of osSystemCat2ISR can be ignored because it is not used at all. The priority value must always be 128.

12.2.21 How to review the peripheral regions configuration (block 19)

The configuration viewer generates the application peripheral regions as followed [SPMF92:0063]:

```
19. Peripheral Regions Configuration:
-----------------------------------------------------
Region  Appl-Mask   Start-Addr  End-Addr    Checked?
-----------------------------------------------------
0       0x00000005  0xfede0000  0xfede1fff    [  ]
1       0x0000000c  0xfede4000  0xfede4fff    [  ]
-----------------------------------------------------
```

The review shall be performed like this:

- Check that the listed peripheral region access rights match your configuration [SPMF92:02.0030]
- Check for each row that Appl-Mask fits to the accessing applications. Each bit which is set means that the corresponding application has access rights to this region. E.g. if bit 0 is set then application with ID=0 has permission to access the region, if bit 1 is set then application with ID=1 has permission to access the region etc.
- Check that the listed peripheral region start and end addresses matches your configuration [SPMF92:02.0030]
- Check for each row that Start-Addr fits to the start address value of the configured peripheral region. Start-Addr must point to the first valid byte in the region.
- Check for each row that End-Addr fits to the end address value of the configured peripheral region. End-Addr must point to the last valid byte in the region.
- Check that the peripheral region index starts with 0
- Check that the peripheral region index is consecutive with no gaps
- Check that the peripheral region belongs to the printed application.

12.2.22 How to review the static MPU regions configuration (block 20)

The configuration viewer generates the static MPU regions configuration as followed:

```
20. MPU configuration (static):
-----------------------------------------------------
Region  Start-Addr  End-Addr    Attributes  Checked?
-----------------------------------------------------
1       0x00000010  0x00000000  0x000000db    [  ]
2       0x00000010  0x00000000  0x000000db    [  ]
3       0x00000000  0x000ffffc  0x000000fd    [  ]
4       0xff000000  0xfffffffc  0x000000d8    [  ]
5       0xfedf0960  0xfee00000  0x000000d9    [  ]
6       0xfedf0000  0xfedf0960  0x000000c9    [  ]
7       0x00000000  0x00000000  0x00000000    [  ]
8       0x00000000  0x00000000  0x00000000    [  ]
9       0x00000000  0x00000000  0x00000000    [  ]
10      0x00000000  0x00000000  0x00000000    [  ]
11      0x00000000  0x00000000  0x00000000    [  ]
-----------------------------------------------------
```

Start-Addr are the values which are written to MPU registers MPLAn
End-Addr are the values which are written to MPU registers MPUAn
Attributes are the values which are written to MPU registers MPATn

The review shall be performed like this:

- Check that all regions 1 … 11 are listed.
- If Start-Addr=0x00000010, End-Addr=0x00000000 and Attributes=0x000000db then the MPU region is used dynamic, i.e. it is reprogrammed when the context is switched.
- If Start-Addr=0x00000000, End-Addr=0x00000000 and Attributes=0x00000000 then the MPU region is not used at all.
- All other regions are user specific MPU region settings.
- Check the user specific MPU region settings that Start-Addr, End-Addr and Attributes fit to the corresponding configurations.
- Check each user specific MPU region setting that Start-Addr is lower than End-Addr.
- Check each user specific MPU region setting that Attributes contains correct application ID.

12.2.23 How to review the application trusted information (block 21)

The configuration viewer generates the application trusted information as followed:

```
21. Application trustedness information:
-------------------------------------------------------------
Appl-ID  Appl-Name                    Value        Checked?
-------------------------------------------------------------
0        (NTAppl)                    non-trusted    [  ]
1        (abcAppl)                   trusted        [  ]
2        (osSystemApplicationCore0)  trusted        [  ]
3        (xyzAppl)                   non-trusted    [  ]
-------------------------------------------------------------
```

The review shall be performed like this:

- Check for each application, that the setting of the attribute TRUSTED fits to the value of the application trustedness information.
- If the application has attribute TRUSTED=TRUE, then Value must be trusted.
- If application has attribute TRUSTED=FALSE, then Value must be non-trusted.
- OS system application osSystemApplicationCore0 must always be trusted.

12.2.24 How to review the core control block address information (block 22)

The configuration viewer generates the core control block information as followed:

```
22. Core control block address:
------------------------------
Core-ID  Address     Checked?
------------------------------
0        0xfedf0c40    [  ]
------------------------------
```

The review shall be performed like this:

- Check that Address value is the same value for linker symbol _osCtrlVarsCore0 in the project map file which is generated by the linker.

12.3 Linker Memory Sections

The OS uses specific memory section names for the linker. Check that these section names are used in the linker file and check that they are used in the assigned area.
The OS uses the linker memory section names described in the following table:

Section Name and Type	Description and Link Requirements
.osExceptionVectorTable section type .text	Contains the core exception vector table. It is generated into file intvect.c
.osEIINTVectorTable section type .text	Contains the EIINT exception vector table. It is generated into file intvect.c
.os_text section type .text	Contains the interrupt handler for category 2 ISRs. It is generated into file intvect.c Must be linked to program code section.
.os_text section type .text	Contains all OS program code, except those which must be placed in special sections (e.g. vector table). Must be linked to program code section.
.os_rodata section type .rodata	Contains the OS constant data, except those which must be placed in special sections (e.g. configuration block osConfigBlock). Must be linked to constant data section.
.os_rosdata section type .rosdata	Contains all OS constants which are placed in ROSDA area, except those which must be placed in special sections (e.g. configuration block osConfigBlock). Must be linked to the constant data in ROSDA section
.osConfigBlock_rodata section type .rodata	Contains the configuration block if SDA optimization is disabled. Must be linked to constant data section.
.osConfigBlock_rosdata section type .rosdata	Contains the configuration block if SDA optimization is enabled Must be linked to constant data in ROSDA section.
.os_bss section type .bss	Contains the uninitialized OS variables Optional initialized to zero by system startup code. Must be linked to the data section.
.os_data section type .data	Contains the initialized OS variables which must be copied from ROM to RAM by system startup code. Must be linked to the data section. This section must be empty!
.os_sbss section type .sbss	Contains uninitialized OS variables which are placed in SDA area if SDA optimization is enabled. Optional initialized to zero by system startup code. Must be linked to the SDA section.
.os_sdata section type .sdata	Contains initialized OS variables which are placed in SDA area if SDA optimization is enabled. Must be linked to the SDA section. This section must be empty!

Section Name and Type	Description and Link Requirements
.osTaskStack<TaskIndex> section type .bss	Contains the uninitialized task specific stack. Must be linked to the data section.
.osSystemStack section type .bss	Contains the uninitialized OS system stack. Must be linked to the data section.
.osIntStackLevel<Priority> section type .bss	Contains the uninitialized ISR specific stack. Must be linked to the data section.
.osAppl_<ApplName>_bss section type .bss	Contains uninitialized application data. Optional initialized to zero by system startup code. Must be linked to the data section.
.osAppl_<ApplName>_sbss section type .sbss	Contains uninitialized application data in SDA area if SDA optimization is enabled. Optional initialized to zero by system startup code. Must be linked to SDA section.
.osAppl_<ApplName>_data section type .data	Contains initialized application data. Must be linked to data section.
.osAppl_<ApplName>_sdata section type .sdata	Contains initialized application data in SDA area if SDA optimization is enabled. Must be linked to SDA section.
.osGlobalShared_bss section type .bss	Contains uninitialized global shared data. Optional initialized to zero by system startup code. Must be linked to data section.
.osGlobalShared_sbss section type .sbss	Contains uninitialized shared data in SDA area if SDA optimization is enabled. Optional initialized to zero by system startup code. Must be linked to SDA section.
.osGlobalShared_data section type .data	Contains initialized shared global data. Must be linked to data section.
.osGlobalShared_sdata section type .sdata	Contains initialized shared data in SDA area if SDA optimization is enabled. Must be linked to SDA section.

12.4 Linker Include Files

The generated linker include files shall be reviewed if they are used for serial production.

12.4.1 Review File osdata.dld

File osdata.dld contains mapping of section types .bss and .data for trusted applications.

For each trusted application the generated lines must look like:

```
/* trusted application <ApplName> */
.osAppl_<ApplName>_bss   align(4) :>.
.osAppl_<ApplName>_data  align(4) :>.
```

The linker include file ends with the section mapping for OS data which must look like:

```
.os_bss    align(4) :>.
.os_data   align(4) :>.
```

12.4.2Review File ossdata.dld

The generated example file ossdata.dld contains the mapping of section types .sbss and .sdata for trusted and non-trusted applications.
For non-trusted applications it also contains the mapping of section types .bss and .data. This is necessary due to optimization for MPU region settings.

For each trusted application the generated lines must look like:

```
/* trusted application <ApplName> */
.osAppl_<ApplName>_sbss         align(4) :>.
.osAppl_<ApplName>_sdata        align(4) :>.
```

For each non-trusted application the generated lines must look like:

```
/* non-trusted application <ApplName> */
.osAppl_<ApplName>_bss          align(4) :>.
.osAppl_<ApplName>_data         align(4) :>.
.osAppl_<ApplName>_sbss         align(4) :>.
.osAppl_<ApplName>_sdata        align(4) :>.
_<Appl_Section_StartAddr> = addr(.osAppl_<ApplName>_bss);
_<Appl_Section_EndAddr>   = endaddr(.osAppl_<ApplName>_sdata)-1;
```

After the mapping of the application sections the mapping for OS SDA sections must be generated like the following lines:

```
.os_sbss    align(4) :>.
.os_sdata   align(4) :>.
```

After the mapping of the OS SDA sections the mapping for global shared sections must be generated like the following lines:

```
.osGlobalShared_sbss    align(4) :>.
.osGlobalShared_sdata   align(4) :>.
.osGlobalShared_bss     align(4) :>.
.osGlobalShared_data    align(4) :>.
_osGlobalShared_StartAddr = addr(.osGlobalShared_sbss);
_osGlobalShared_EndAddr   = endaddr(.osGlobalShared_data)-1;
```

12.4.3 Review File osstacks.dld

File osstacks.dld contains mapping of all stack sections.

Each stack section mapping for used interrupt priority levels must look like:

```
.osIntStackLevel<PrioLevel> align(4) :>.
_osIntStackLevel<PrioLevel>_StartAddr = addr(.osIntStackLevel<PrioLevel>);
_osIntStackLevel<PrioLevel>_EndAddr = endaddr(.osIntStackLevel<PrioLevel>);
```

<PrioLevel> is the interrupt priority level 0 … 15

The mapping for the system stack section must look like:

```
.osSystemStack align(4)  :>.
_osSystemStack_StartAddr = addr(.osSystemStack);
_osSystemStack_EndAddr = endaddr(.osSystemStack);
```

The system stack section is followed by the OS task stack sections which must look like:

```
.osTaskStackosSystemApplicationCore00 align(4) :>.
_osTaskStackosSystemApplicationCore00_StartAddr =
                                addr(.osTaskStackosSystemApplicationCore00);
_osTaskStackosSystemApplicationCore00_EndAddr =
                                endaddr(.osTaskStackosSystemApplicationCore00);

.osTaskStackosSystemApplicationCore01 align(4) :>.
_osTaskStackosSystemApplicationCore01_StartAddr =
                                addr(.osTaskStackosSystemApplicationCore01);
_osTaskStackosSystemApplicationCore01_EndAddr =
                                endaddr(.osTaskStackosSystemApplicationCore01);
```

The OS task stack sections are followed by the application task stack sections.
Each application task stack section mapping must look like:

```
.osTaskStack<applname><index>  align(4)  :>.
_osTaskStack<applname><index>_StartAddr = addr(.osTaskStack<applname><index>);
_osTaskStack<applname><index>_EndAddr = endaddr(.osTaskStack<applname><index>);
```

<applname> is the name of the owner application

<index> is the index number of the task stack: 0 … number of tasks per application

12.4.4Review File osrom.dld

File osrom.dld contains the sections used for initialized variables.

For each application which has data to be initialized during startup code the following lines must be generated:

```
.ROM_osAppl_<ApplName>_data   ROM(.osAppl_<ApplName>_data)  :>.
.ROM_osAppl_<ApplName>_sdata  ROM(.osAppl_<ApplName>_sdata) :>.
.ROM_osAppl_<ApplName>_tdata  ROM(.osAppl_<ApplName>_tdata) :>.
```

File osrom.dld ends with the global shared initialized data sections which must look like:

```
.ROM_GlobalShared_data   ROM(.osGlobalShared_data)  :>.
.ROM_GlobalShared_sdata  ROM(.osGlobalShared_sdata) :>.
.ROM_GlobalShared_tdata  ROM(.osGlobalShared_tdata) :>.
```

12.4.5Review File ostdata.dld

File ostada.dld contains the mapping for application data in TDA section.

For each application which has data in TDA section the following line must be generated:

```
.osAppl_<ApplName>_tdata  align(4) MAX_SIZE(0x0100) :>.
```

File ostdata.dld ends with the mapping for global shared data in TDA section which must look like:

```
.osGlobalShared_tdata  align(4) MAX_SIZE(0x0100) :>.
```

12.5 Stack Size Configuration

The size of task stacks, ISR stacks and the system stack is configured by the user. The application code must not use more stack then configured. Before trusted or non-trusted application code (tasks, ISRs, trusted and non-trusted functions) is executed the OS always reprogramms MPU region 0 in order to protect the stack memory areas.

The following table provides an overview of the stacks and which code parts need to be considered for the analysis of the required stack sizes.

Stack	Usage
System Stack	StartupHook ErrorHook ProtectionHook [SPMF92:0082] ShutdownHook [SPMF92:0081]
Task Stacks	the corresponding task function and its call tree ISRs of category 1 (when interrupting a task) ErrorHook Storing a context (144 Byte)
ISR Stacks	the corresponding category 2 ISR function and its call tree ISRs of category 1 (when interrupting an ISR) ErrorHook Storing a context (144 Byte)

If no static analysis for the stack requirement is made, the stack usage may be measured by means of the API functions osGetStackUsage, osGetISRStackUsage and osGetSystemStackUsage, when StackUsageMeasurement is configured. Measurement has to consider the maximum stack usage of the code under measure. It has to be ensured, that all directly and indirectly called functions are executed and use the maximum possible stack.

Stack Usage measurement is implemented by filling the stack with a pattern on startup and counting the number of continuous patterns which have not been overwritten with another value. This may lead to a too small measured value in case the function under measure uses this pattern as value on its stack.

As the hardware allows to enable interrupts even in non-trusted code, any non-trusted ISR may enable nesting. Therefore, the user shall expect that interrupt nesting can always occur when defining the system stack size [SPMF92:0089].

The stack usage must be measured after the maximum call depth has been reached [SPMF92:0090].

12.6 Stack Monitoring

The stack memory area is write protected via MPU region 0. Trusted and non-trusted applications and the OS cannot write to stack areas which belong to other applications.

This hardware based stack monitoring does not detect all stack errors [SPMF92:0076]. Stack overflow cannot be detected if the task or ISR stack is mapped immediately after the corresponding application data or global shared section. Stack underflow cannot be detected if the task or ISR stack is mapped immediately before the corresponding application data or global shared section.

12.7 Usage of MPU Regions

MPU region 0 is always used for stack area protection. It is always reprogrammed when the context is switched. Therefore MPU region 0 cannot be configured by the user.

Each MPU region 1 … 11 can be configured for static or dynamic usage:

- If a MPU region is configured for static usage, then it is initialized in StartOS and not changed any more. For static MPU regions the user must specify the access attributes.
- If a MPU region is configured for dynamic usage, then it is initialized in StartOS and not always reprogrammed when the context is switched. The access attributes for dynamic MU regions are configured by the OS.

All stack sections must be mapped to a consecutive memory area. A static MPU region must be configured for this memory area so that trusted and non-trusted application have only read access to it. That means in supervisor and in user mode only reading is possible. Write access to dedicated stack is achieved at runtime via reprogrammed MPU region 0 when a task or ISR is started.

A static MPU region must be configured for the applications data area so that in supervisor mode read and write is possible and in user mode only reading is possible. Write access to the dedicated application data area is achieved via dynamic MPU region which must be configured for each non-trusted application.

A static MPU region must be configured for the code and const area (i.e. ROM/FLASH) so that in supervisor mode (trusted applications) read, write and execute is possible and in user mode (non-trusted applications) only read and execute is possible in that area.

12.8 Usage of Peripheral Interrupt API

The OS provides functions which allow write access to EI level interrupt control registers EICn and to EI level interrupt mask registers IMRn in user mode. Non-trusted applications can enable or disable peripheral interrupt sources by means of this functions. Call of the OS peripheral interrupt API functions must be checked in every application that only valid interrupt sources are modified [SPMF92:04.0016].