MICROSAR Classic OS HAL

Technical Reference

TriCore

Version 1.23.0

Author	Vector Informatik GmbH
Status	Released

Document Information

History

Author	Date	Version	Remarks
virleh	2020-03-24	1.0.0	First released version.
visbpz, virrlu	2020-04-29	1.1.0	Hardware overview extended. Added chapter Exception Context Manipulation.
visto	2020-06-23	1.2.0	Added information about correct initialization of BTV and BIV registers. Extend the PSW register handling chapter. Added chapters about SYSCON and DBGSR register usage.
visto	2020-07-13	1.3.0	Mark optional used registers within “Hardware Software Interface” chapter. Refine SYSCON register handling chapter.
visto	2020-09-15	1.4.0	TC49x derivative added (development status)
visto	2020-11-30	1.5.0	Chapter “Access Rights for Protection Set 0” added Chapter “Vector Table Calls” added
virsmn	2020-12-23	1.6.0	Removed author identity.
visto	2021-02-17	1.7.0	Added hint for PSW.CDC (call depth counter)
visbpz	2021-03-30	1.8.0	Updated the integration chapter
visto	2021-04-29	1.9.0	Updated chapter platform restrictions “Exception handlers”
visto	2021-08-11	1.10.0	Added chapter “Note on SMI-481”
visto	2021-08-31	1.11.0	Added chapter “Preconditions upon Os_Init”
visto	2021-12-21	1.12.0	Added chapter “Restriction for privileged OS Applications” Correct PSW.IO handling
visto	2022-02-08	1.13.0	Added chapter “User Defined Class 3 Exception” Added chapter “Floating Point Context Extension” INT_SRC handling in HSI refined
visto	2022-07-20	1.14.0	Renamed MICROSAR to MICROSAR Classic Added additional information about syscall “Note on SMI-540” Added additional information about Infineon safety manual item "SMC[SW]:IR:FFI_CONTROL”

viswip	2022-10-04	1.15.0	Added explanation about the Cyber Security Bit handling.
visto	2023-05-05	1.16.0	Added chapter “System Call Instruction”
visdri	2023-08-07	1.17.0	Added support for TC4Dx
visdri	2023-11-17	1.18.0	Update TC4x HW manuals, HSI, notes
vistmo	2023-12-27	1.19.0	Added HighTec (LLVM) compiler support for TC4x
vistmo	2024-01-17	1.20.0	Added Diab support for TC4x
visdri	2024-03-19	1.21.0	Updated Interrupt / Exception Vector Table caution.
visdri	2024-06-04	1.22.0	Added TC49xN HW manual
vistmo	2024-10-30	1.23.0	Added MMU chapter Added restriction on STM module configuration

Reference Documents

No.	Source	Title	Version
[1]	Vector	MICROSAR Classic OS Technical Reference	See delivery information
[2]	Vector	MICROSAR Classic SafetyManual	See delivery information
[3]	Infineon	AURIX TC3xx Safety Manual	v2.0

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

Contents

Illustrations

Tables

1 Introduction

This document describes platform specific details about the usage and functions of “MICROSAR Classic OS” on the TriCore derivative family. General information about “MICROSAR Classic OS” can be found in [1].

1.1 Hardware Overview

The following table summarizes information about MICROSAR Classic OS. It gives detailed information about the derivatives and supported compilers of the TriCore derivative family. As very important information the documentations of the hardware manufacturers are listed. MICROSAR Classic OS is based upon these documents in the given version.

Derivative	Hardware Manufacturer Document Name	Document Version
TC21x TC22x TC23x	User Manual: tc23x_tc22x_tc21x_um_v1.1.pdf	V1.1
	Errata Sheet: TC22x_TC21x_AB_Errata_Sheet_v1_2_03804A.pdf	V1.2
TC24x	Target Specification: tc24x_ts_v2.0_OPEN_MARKET.pdf	V2.0
TC26x	User Manual: tc26xB_um_v1.3._usermanual_rev1v3.pdf	V1.3
	Errata Sheet: TC26x_BB_Errata_Sheet_rev1v2_03989A_2016-04-18.pdf	V1.2
TC27x	User Manual: tc27xD_um_v2.2_UserManual_rev2v2_2014-12.pdf	V2.2
	Errata Sheet: TC27x_BC_Errata_Sheet_rev1v5_2015_09_16.pdf	V1.5
TC29x	User Manual: tc29xB_um_v1.3._TC29x_B-Step_User_Manual_rev_1v3_2014_12.pdf	V1.3
	Errata Sheet: TC29x_BA_Errata_Sheet_v1_0.pdf	V1.0
TC32x	User Manual: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0
	Appendix: TC33x_TC32x_um_appx_v1.4.0.pdf	V1.4.0
TC33x	User Manual: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0
	Appendix: TC33X_um_appx_V1.1.0.pdf	V1.1.0
TC35x	User Manual:	V1.4.0

	TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	
	Appendix: TC35X_um_appx_V1.1.0.pdf	V1.1.0
TC36x	User Manual: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0
	Appendix: TC36X_um_appx_V1.1.0.pdf	V1.1.0
TC37x	User Manual: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0
	Appendix: TC37x_um_appx_v1.4.0.pdf	V1.4.0
TC38x	User Manual: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0
	Appendix: TC38X_ts_appx_V2.3.0.pdf TC38x_Data_Sheet_Addendum_v1.5.pdf	V2.3, V1.5
TC39x	User Manual: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0
	Errata Sheet: TC39x AA_Errata_Sheet_rev1v0_2016-06-08.pdf Appendix: TC39xB_um_appx_v1.4.0.pdf	V1.0, V1.4.0
TC3Ex	User Manual: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0
	Appendix: TC3Ex_um_appx_v1.4.0.pdf	V1.4.0
TC49x	User Manual: Aurix TC49x User Manual - Draft	V0.73, Draft, 2021-12-02
TC4Dx	User Manual: Aurix TC4Dx User Manual - Draft	V0.73, Draft, 2023-04-14
	Errata Sheet. TC4Dx Errata Sheet - Draft	V0.62, Draft, 2023-08-24
TC49xN	User Manual: Aurix TC49xN User Manual - Draft	V0.71, Draft, 2024-02-29

Table 1-1 Supported TriCore Aurix Hardware

Tasking	v4.2r2 (TC2xx only) v6.0r1p2 (TC3xx only) Smart Code (TC4xx only)
HighTec (GNU)	V4.6.3.0 (Non-TC4xx)
WindRiver Diab	V4.9.6.4 (Non-TC4xx) V5.9.8.3 (TC4xx only)
GreenHills (GHS)	2019.1.5 (Non-TC4xx) 2023.1.4 (TC4xx only)
HighTec (LLVM)	8.1.0 (TC4xx only)

Table 1-2 Supported TriCore Aurix Compilers

2 Functional Description

2.1 Unhandled Interrupts

In case of an unhandled interrupt request the application can obtain the source number of the triggering interrupt request by the OS API Os_GetUnhandledIrq() (see [1] for details). On the TriCore derivative family, the 32-bit value returned by the API represents the level of the interrupt. The level is equal to the interrupt priority.

2.2 Unhandled Exceptions

In case of an unhandled exception the application can obtain the source number of the triggering exception by the OS API Os_GetUnhandledExc() (see [1] for details). On the TriCore derivative family, the 16 upper bits of the 32-bit value returned by the API represent the exception class. The 16 lower bits represent the TIN (Trap identification number) that can be looked up in the hardware manual.

2.3 Exception Context Manipulation

The TriCore derivative family supports the feature to read and modify the interrupted context in case of a hardware exception by using OS APIs Os_GetExceptionContext and Os_SetExceptionContext (see [1] for details).

2.4 PSW Register Handling

PSW.S bit handling	MICROSAR Classic OS sets the safety task identifier bit to 1 for trusted software parts and to 0 for non-trusted software parts. Note: This bit is called “Alternate task identifier Selector” for TC4xx derivatives.
PSW.IS bit handling	MICROSAR Classic OS sets the interrupt stack bit to 1. Thus automatic hardware stack switch is not supported.
PSW.GW bit handling	MICROSAR Classic OS sets the global address register write permission to 0. Write permission to A0, A1, A8 and A9 are not allowed.
PSW.CDE bit handling	MICROSAR Classic OS sets the call depth enable bit to 1 upon start of a thread. Call depth counting is enabled.
PSW.CDC bits handling	MICROSAR Classic OS sets the call depth counter to 1 upon start of a thread.
PSW.USB bits handling	MICROSAR Classic OS does not manipulate the USB flags.
PSW.PRS bits handling	MICROSAR Classic OS sets these bits during context switch (of Tasks, ISRs, non-trusted functions etc) to the configured value of the owning application. Configuration parameter: App Memory Protection Identifier (OsAppMemoryProtectionIdentifier)
PSW.IO bits handling	MICROSAR Classic OS sets these bits during context switch (of Tasks, ISRs, non-trusted functions etc) to the configured value of the owning application. Configuration parameter: OsApplicationIsPrivileged If OsApplicationIsPrivileged is true PSW.IO will be set to “Supervisor Mode” otherwise it will be set to “User-1 Mode”. Note: The OS does not implement the usage of “User-0 Mode”.

Note
The OS enables the call depth counting feature (PSW.CDE = 1). As a result nestable function calls are limited.

Within a task or ISR it is possible to have 63 nested function calls. If the limit is exceeded a class 3 TIN 2 exception (Call Depth Overflow) is raised.

2.5 SYSCON (/CORECON) register handling

SYSCON register is renamed to CORECON for TC4xx derivatives. Some of the fields of SYSCON are marked as reserved in CORECON.

SYSCON.BHALT	Only for TC3xx derivatives. The OS uses this bit within the StartCore and StartNonAutosarCore API to get a core out of reset.
SYSCON.U1_IOS	This bit is never manipulated by OS and it may be altered by application software.
SYSCON.U1_IED	The OS sets this bit to 1 during initialization, when the memory protection feature is configured. If the memory protection is not configured the OS does not alter this bit and it may be modified by application software.
SYSCON.ESDIS	Only for TC3xx derivatives. This bit is never manipulated by OS. It may be used by application software.
SYSCON.TS	During Os initialization this bit is set to 1 in order to set the PSW.S bit to 1 upon trap entry.
SYSCON.IS	During Os initialization this bit is set to 1 in order to set the PSW.S bit to 1 upon trap entry.
SYSCON.TPROTEN	This bit is never manipulated by OS. It may be used by application software.
SYSCON.PROTEN	The OS sets this bit to 1 during initialization, when the memory protection feature is configured. If the memory protection is not configured the OS does not alter this bit and it may be modified by application software.
SYSCON.FCDSF	This bit is never manipulated by OS. It may be used by application software.

Note
The OS assumes that the COMPAT.SP register is set to one (reset value). Otherwise, the SYSCON register will be safety endinit protected.

If the SYSCON register is endinit protected an exception is raised if the OS tries to initialize it during Os_Init.

Therefore COMPAT.SP should stay on its reset value.

2.6 BOOTCON register handling

BOOTCON.BHALT	Only for TC4xx derivatives. The OS uses this bit within the StartCore and StartNonAutosarCore API to get a core out of reset.

2.7 DBGSR register handling

DBGSR.EVTSRC	Read only. This bit is not used by the OS
DBGSR.PEVT	This bit is set to 0 upon call of StartCore and StartNonAutosarCore API.
DBGSR.PREVSUSP	Read only. This bit is not used by the OS
DBGSR.SUSP	This bit is set to 0 upon call of StartCore and StartNonAutosarCore API.
DBGSR.SIH	Read only. This bit is not used by the OS
DBGSR.HALT	These bits are written to 0x2 upon call of StartCore and StartNonAutosarCore API.
DBGSR.DE	This bit is set to 0 upon call of StartCore and StartNonAutosarCore API.

Note
The DBGSR register of core 0 is never altered by the OS.

Note
The DBGSR register may be accessed by application software.

2.8 MMU

This platform does not support MMUs.

3 Integration

3.1 Hardware Software Interface

The following chapter describes the Hardware-Software Interface of the TriCore derivative family.

There are registers which are always under control of the OS and may never be altered by application software.

On the other hand, there are registers which are only under supervision of the OS if certain aspects have been configured. The following chapters will list under which configuration aspects such registers are handled exclusively by the OS.

3.1.1 Context

Register	Handled by OS	Altering by application SW
A0-A15	Always	Allowed (except A8; see the note below)
D0-D15	Always	Allowed
PSW	Always	Not allowed
PCXI	Always	Not allowed
DPR0L, DPR0H	If memory protection has been configured (SC3 or SC4 systems)	Allowed in SC1 or SC2 systems

Note
The register A8 is exclusively used by the OS to hold the pointer to the current thread. Thus any addressing modes which would use A8 register are not possible.

3.1.2 Core Registers

Register	Handled by OS	Altering by application SW
ICR	Always	Not allowed
SYSCON (/CORECON)	Always	SYSCON (/CORECON) handling is described in chapter “SYSCON register handling”
PCXI	Always	Not allowed
FCX	Always	Allowed in startup code (before OS is started)
PSW	Always	Not allowed
PC	Always	Not allowed
PPRS (TC4xx only)	Always	Not allowed
DBGSR	In multicore systems	DBGSR handling is described in chapter “DBGSR register handling”
DPRxL, DPRxH	If memory protection has been configured (SC3 or SC4 systems)	Allowed in SC1 or SC2 systems
CPRxL, CPRxH		
DPREx		
DPWEx		
CPXEx		

3.1.3 Interrupt Registers

3.1.3.1 Handling for category 2 ISRs

Register	Handled by OS	Altering by application SW
INT_SRC0 – INTSRC255 (Aurix TC2xx)	Always	Not allowed
INT_SRC0 – INTSRC1023 (Aurix TC3xx)	Always	Not allowed
INT_SRC0 – INTSRC2047 (Aurix TC4xx)	Always	Not allowed

3.1.3.2 Handling for category 0 and 1 ISRs

Register	Handled by OS	Altering by application SW
INT_SRC0 – INTSRC255 (Aurix TC2xx)	The OS pre-initializes the priority (SRPN) and type of service (TOS) during OS initialization. It does not enable the interrupt source.	The application software may access the registers to enable the interrupt source or to trigger the ISR. The application software may also access the registers if category 0 or category 1 ISRs shall be allowed before StartOS is called.
INT_SRC0 – INTSRC1023 (Aurix TC3xx)		
INT_SRC0 – INTSRC2047 (Aurix TC4xx)		

3.1.3.3 Cyber Security Bit handling

Note
On the Aurix TC4xx, the cyber security bit (CS) is set if the interrupt is assigned to the CS core, otherwise it remains zero.

3.1.4 GPT Registers

Register	Handled by OS	Altering by application SW
T2, T3, T6	If the GPT timer is used as driver timer for an OS counter	Is allowed if the GPT timer is not configured as driver timer for an OS counter
T2CON, T3CON, T6CON		
CAPREL		

3.1.5 STM Registers

Register	Handled by OS	Altering by application SW
TIM0, TIM5, TIM6	If the STM timer is used as driver timer for an OS counter	Is allowed if the STM timer is not configured as driver timer for an OS counter
CMCON		
CAP		
CMP0		
CMP1		

3.1.6 Core STM Registers (TC4xx only)

Register	Handled by OS	Altering by application SW
ABS	If the STM timer is used as driver timer for an OS counter	Is allowed if the STM timer is not configured as driver timer for an OS counter
CMCON		
ICR		
CMP0		
CMP1		

3.1.7 System Call Instruction

The OS internally uses the system call instruction in order to switch to privileged mode in case of an OS API call. Any use of this instruction causes that some OS service gets performed. The system call is implemented by use of the following assembly instruction:

```
SYSCALL
```

Direct (accidental) use of this instruction outside the OS would circumvent some safety measures of the OS on the call of and return from an OS API service.

3.2 Configuration of X-Signal

Logical Priority	A low number for OsIsrInterruptPriority attribute means a low logical priority
X-Signal ISR Interrupt Priority	Aside from the priority rules for the X-Signal receiver ISR described in [1] the OsIsrInterruptPriority can be chosen freely.
X-Signal ISR Interrupt Source	Any interrupt source, which is not used by other modules, may be used for the X-Signal ISR. The offset of the SRC register of the used interrupt source has to be specified for OsIsrInterruptSource.

3.3 Configuration of Interrupt Mapping

Supported Interrupt Type	DMA (TC2xx and TC3xx)
	DMA0, DMA1, GTM, PPU (TC4xx only)
ISR Category	Category 0 must be used
Supported on derivative families	All

After mapping an interrupt source to the DMA, the interrupt will be routed to the DMA interrupt controller. The configuration of the DMA controller must be done by the user.

The mapped interrupt is not in the generated core interrupt vector table and an unhandled interrupt handler is generated.

3.4 Configuration of Interrupt Sources

Special care must be taken when configuring the attribute “OsIsrInterruptSource”.
Within the TriCore platform this attribute specifies the offset of the Interrupt Router SRC register of a specific interrupt source.
The offset is relative to the interrupt router register base address and must be specified as 16-bit value.

Caution
The offset must always be a multiple of four. During OS initialization, an exception is raised if the offset is not a multiple of four.

3.5 Initialization of BTV and BIV registers

For TC2xx/TC3xx derivatives, since the BTV and BIV registers are endinit-protected registers, the OS does not initialize them. It assumes that the BTV and BIV registers are set up correctly during startup code before Os_Init is called.
The OS and the linker provide symbols for setting up the BTV and BIV registers to the OS-generated exception and interrupt tables.

3.5.1 Symbols for HighTec (GNU), Diab and GHS compiler

Set up the registers using the following symbols.

Register	Symbols
BTV	_OS_EXCVEC_CORE<core ID>_CODE_START
BIV	_OS_INTVEC_CORE<core ID>_CODE_START

Table 3-1 Exception and Interrupt Table Symbols

3.5.2 Symbols for Tasking Compiler

Set up the registers using the following symbols.

Register	Symbols for OS version <= 2.47.xx	Symbols for OS version >= 2.48.xx
BTV	_lc_u_trap_tab_tc<core ID>	_OS_EXCVEC_CORE<core ID>_CODE_START
BIV	_lc_u_int_tab_tc<core ID>	_OS_INTVEC_CORE<core ID>_CODE_START

Table 3-2 Exception and Interrupt Table Symbols for Tasking Compiler

3.6 Category 1 ISRs before StartOS

Refer to [1] to find out the steps necessary to activate and serve Category 1 ISRs before the OS has been started. Table 3-3 Global interrupt flag / Interrupt priority registers shows the registers that are **only allowed** to be written for servicing Category 1 ISRs before StartOS. After the OS has been started the rules of the Hardware Software Interface (HSI) apply, for details refer to 3.1.

Platform	Register	Altering by application SW
All derivative families	ICR	Allowed

Table 3-3 Global interrupt flag / Interrupt priority registers

3.7 Note on SMI-481

The OS safety manual [2] defines SMI-481. This safety manual item states that in case of a severe error the PanicHook might not be called in all cases.

But this behavior depends on the platform the OS is running on.

With respect to SMI-481 the TriCore Aurix platform behaves as follows:

Before calling the PanicHook, the OS tries to disable all interrupts. If the OS is not in supervisor mode, a class 1 TIN 1 trap (privileged instruction) occurs.

The OS will initiate protection violation handling and the ProtectionHook will be called instead of the PanicHook.

3.8 Preconditions for calling Os_Init

For a correct startup, the Microsar Classic OS for Aurix requires the following conditions before Os_Init may be called.

ICR.IE	Interrupts must be globally disabled (ICR.IE = 0)
COMPAT.SP	SYSCON (/CORECON) Safety Protection Mode Compatibility must be set (COMPAT.SP = 1). I.e., SYSCON (/CORECON) is not safety endinit protected (TC1.3 behavior).
Context Saving Areas	The startup code must have initialized the CSAs

3.9 User Defined Class 3 Exception

The OS generates a special handling if a user defined class 3 exception (context management exception) is configured.

The generated code does not consume any CSAs before jumping to the user defined class 3 exception handler.

Usual Exception Handling sequence	Class 3 Exception Handling Sequence
Exception is triggered.	
The hardware saves an upper context and jumps to the corresponding handler in the exception vector table.	
The OS saves a lower context.	The OS loads the whole 32-bit address of the user exception handler into register A15
The OS calls the user exception handler (implemented as void/void function).	The OS performs a 32-bit jump (using A15) to the user exception handler.
After returning from the user exception handler the OS restores a lower context	
The OS perform a return from exception instruction (“rfe”)	

Table 3-4 Usual vs. class3 user exception handling

Note
The user class 3 exception handler must not be left with a usual return from function (“ret”) instruction.

If the program execution shall be resumed (if possible) then the user handler must be left with a return from exception (“rfe”) instruction.

3.10 Floating Point Context Extension

The OS technical reference [1] describes the floating-point context extension feature.

Since the TriCore Aurix platform has no dedicated FPU registers which need preservation among context switches, there is not the need to extend the context due to floating-point usage.

Therefore, switching on this feature within the OS configuration (as explained in [1]) does have no effect with TriCore Aurix.

The recommendation is to have it always switched off (OsFpuUsage = NONE).

3.11 Note on SMI-540

The OS safety manual [2] defines SMI-540. This safety manual item states that the user of MICROSAR Classic Safe shall verify that the user software does not contain any system call instructions.

Within the TriCore Aurix architecture this is the “syscall” instruction. Only OS code may include this instruction.

A possible verification method might be to review the user code for (inline) assembler statements, pragmas or intrinsic functions containing the “syscall” instruction.

3.12 Note on SMC[SW]:IR:FFI_CONTROL

SMC[SW]:IR:FFI_CONTROL is an assumption of use from the Infineon Safety Manual [3] related to the access protection mechanism of interrupt router registers (IR).

Basically, it states that accesses to interrupt router registers shall be restricted. Since IR registers are part of the OS HSI access to some registers are needed by the OS.

The following table shows which OS features requires access to which IR registers.

OS Feature	Access in API	IR register	Note
Initialization	Os_Init	INT_SRCx[0..15] INT_SRCx[16..31]	Only the OS master core needs access to all configured interrupt sources
Interrupt source API	Os_Dis/EnableInterruptSource Os_IsInterruptSourceEnabled Os_InitialEnableInterruptSources	INT_SRCx[0..15]	The calling core needs access to the interrupt source
	Os_IsInterruptPending Os_ClearPendingInterrupt	INT_SRCx[16..31]	
X-Signals	all OS API functions which are used cross core (e.g. ActivateTask)	INT_SRCx[16..31]	The caller core needs access to the interrupt source which is assigned to the X-Signal
OS Timer	During timer initialization	INT_SRCx[16..31]	The core which uses the timer needs access to the timer interrupt source

Table 3-5 OS Access to IR registers

4 Platform Restrictions

4.1 Exception Handlers

- In an SC3/SC4 configuration the exception handler for trap class 1 is implemented by the OS
- In an SC3/SC4 configuration the exception handler for trap class 6 is implemented by the OS

4.2 Memory

Caution

The TriCore Hardware enforces that a configured MPU region must be followed by at least 15 padding bytes before the next region may be started.

MICROSAR Classic OS obey to this rule within the generated linker scripts. For other additional configured MPU regions the user has to take care to fulfill this requirement

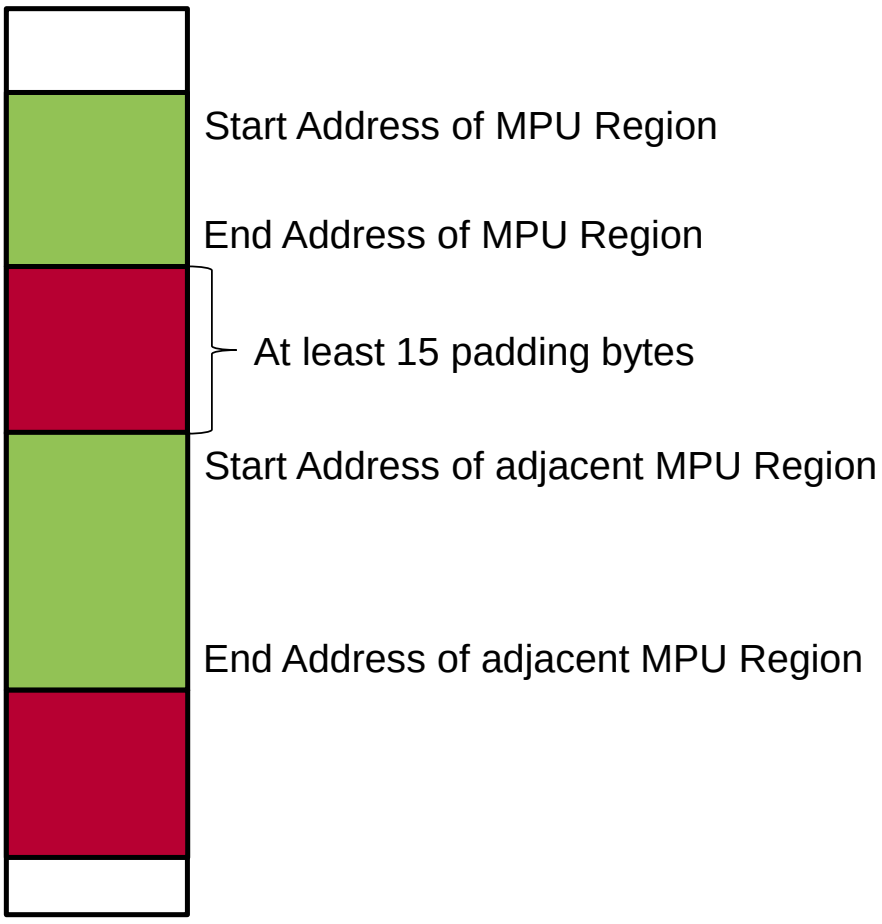

Figure 4-1 Padding bytes between MPU regions

Caution
Due to MPU granularity all start addresses and end addresses of configured MPU regions must be a multiple of 8.

MICROSAR Classic OS programs the MPU to grant access to the memory region between start address and end address.

> Access to configured start address itself is granted
> Access to configured end address is prohibited

Caution
MICROSAR Classic OS does not use the System MPU to achieve freedom of interference between cores.

This has to be done by the application.

The system MPU has to be initialized by a lockstep core. It must not be accessed by non-lockstep cores.

Caution
The App Memory Protection Identifier of each System Application must be set to 0.

Note
All stack sizes shall be configured to be a multiple of 8

Expert Knowledge
For proper context management exception handling the LCX should be initialized during startup code that it does not point to the last available CSA.

In this way some CSAs are reserved which can be used within the context exception handling for further function calls.

4.3 Interrupt / Exception Vector Table

Caution regarding alignment of interrupt/exception vector tables
The interrupt vector table (whose address is set using BIV register) and exception vector table (whose address is set using BTV register) must be aligned in the linker script.

The interrupt vector table must be aligned to 0x2000 byte boundary.

The exception vector table must be aligned to a 0x100 byte boundary.

The alignment is done in the generated linker file which can be included in the user linker file using e.g.:

```
#define OS_LINK_INTVEC_CODE
#include "Os_Link_Core0.ld"
```

4.4 Section Symbols

Note
In order to have access to all memory within a section,

the linker symbol <section name>_LIMIT must be used instead of

<section name>_END to configure the end address of an MPU region.

4.5 Access Rights for Protection Set 0

The TriCore Aurix hardware switches automatically to protection set 0 whenever an interrupt or exception is taken. These interrupt or exception entry points always contain OS code in the first place.

Therefore, access rights for protection set 0 must cover OS access rights.

It is recommended to configure access rights for trusted software as described in the OS technical reference.

Reference
“TechnicalReference_Os.pdf” (see [1]) chapter “Recommended Configuration”

4.6 Restriction for privileged OS Applications

Privileged OS applications must be configured to run in protection set 0.

Within the OS configuration this means when OsApplicationIsPrivileged is true

OsAppMemoryProtectionIdentifier has to be set to 0.

4.7 Vector Table Calls

The interrupt vector table as well as the exception vector table is generated to use a “call” instruction.

This instruction is capable to call a function which address must lie within an address range ±16 Mbyte relative to the current PC (24-bit address range).

Therefore, the vector tables and the ISR handlers must be linked into the same 24-bit memory space.

4.7.1 Exception Handlers

The following handlers must lie within a ±16 Mbyte memory relative to the exception table (section OS_EXCVEC_CORE<Core ID>_CODE):

- Os_Hal_SysCall
- All user defined trap handlers

4.7.2 Interrupt Handlers

The following handlers must lie within a ±16 Mbyte memory relative to the interrupt table (section OS_INTVEC_CORE<Core ID>_CODE):

- Os_Hal_UnhandledIrq
- Os_Hal_IsrRun
- All user defined category 1 ISRs
- All user defined category 0 ISRs

4.8 Restriction on STM channels configuration

Caution

It is not allowed to assign HW counters using the same STM module to applications running on different cores, even when these counters use two different STM channels of the same STM module.

The reason is that an STM module shall be accessed by one core only. Multicore access to the same STM module can result in an unpredictable timer behavior (since the same STM module registers would be accessed concurrently by different cores causing race conditions).

The erroneous configuration is automatically detected by Davinci Configurator and signaled through an error message.

5 Derivatives Specifics

5.1 TC4xx family

Note on CPU_TC.134

CPU_TC.134 is a HW erratum related to FPU conversion instructions raising the invalid flag incorrectly. Please refer to the errata sheet for details.

Microsar OS does not implmented the proposed CPU_TC.134 workaround.
The workaround can be implemented by the OS user though.

Explanation:

The workaround is only possible if trapping on invalid FPU operation is configured (i.e.: FPU_SYNC_TRAP_CON.FIE==1).

Microsar OS does not write the FPU_SYNC_TRAP_CON.FIE bit which has the reset value of 0.

1/ If the user does not set FPU_SYNC_TRAP_CON.FIE bit, e.g. before the OS starts, no workaround is available anyway.

2/ If the OS user configures trapping on invalid FPU operation, the CSE trap (Class 2 Trap), generated due to the invalid condition, is handled by the OS as an unhandled exception if the user does not define his own exception handler.

In this case, the unhandled exception handler calls the ProtectionHook.

The OS user can implement the proposed CPU_TC.134 workaround inside the ProtectionHook. In case a return to the normal program flow shall occur, the usage of exception context manipulation (see [1]) may be necessary/helpful.

3/ If the OS user configures trapping on invalid FPU operation and defines his own exception handler for Class 2 traps, he can implement the proposed CPU_TC.134 workaround in his handler. Please refer to [1] regarding user defined exception handlers.

6 Contact

Visit our website for more information on

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

www.vector.com