MICROSAR OS HAL

Technical Reference

TriCore

Version 1.06.00

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.

Reference Documents

No.	Source	Title	Version
[1]	Vector	MICROSAR OS Technical Reference	See delivery information

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 OS” on the TriCore derivative family. General information about “MICROSAR OS” can be found in [1].

1.1 Hardware Overview

The following table summarizes information about MICROSAR 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 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: TC3xx_um_v1.4.0_part1.pdf TC3xx_um_v1.4.0_part2.pdf	V1.4.0

	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	V0.7 2020-08-15

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
WindRiver Diab	V4.9.6.4
GreenHills (GHS)	2019.1.5

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 PSW Register Handling

PSW.S bit handling	MICROSAR OS sets the safety task identifier bit to 1 for trusted software parts and to 0 for non-trusted software parts.
PSW.IS bit handling	MICROSAR OS sets the interrupt stack bit to 1. Thus automatic hardware stack switch is not supported.
PSW.GW bit handling	MICROSAR 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 OS sets the call depth enable bit to 1 upon start of a thread. Call depth counting is enabled.
PSW.CDC bits handling	MICROSAR OS sets the call depth counter to 1 upon start of a thread.
PSW.USB bits handling	MICROSAR OS does not manipulate the USB flags.
PSW.PRS bits handling	MICROSAR 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 OS sets these bits during context switch (of Tasks, ISRs, non-trusted functions etc) to the configured value of the owning application. Configuration parameter: Trusted (OsTrusted) If “Trusted” is set to 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”.

2.4 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).

2.5 SYSCON register handling

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 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.

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	Always	SYSCON 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 initialize the priority (SRPN) and type of service (TOS) during OS initialization. It does not enable the interrupt source.	The application software may access the register in order to enable the interrupt source or to trigger the ISR
INT_SRC0 – INTSRC1023 (Aurix TC3xx)		
INT_SRC0 – INTSRC2047 (Aurix TC4xx)		

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		

3.2 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

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, 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

4 Platform Restrictions

4.1 Exception Handlers

- The exception handler for trap class 1 is implemented by the OS
- 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 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

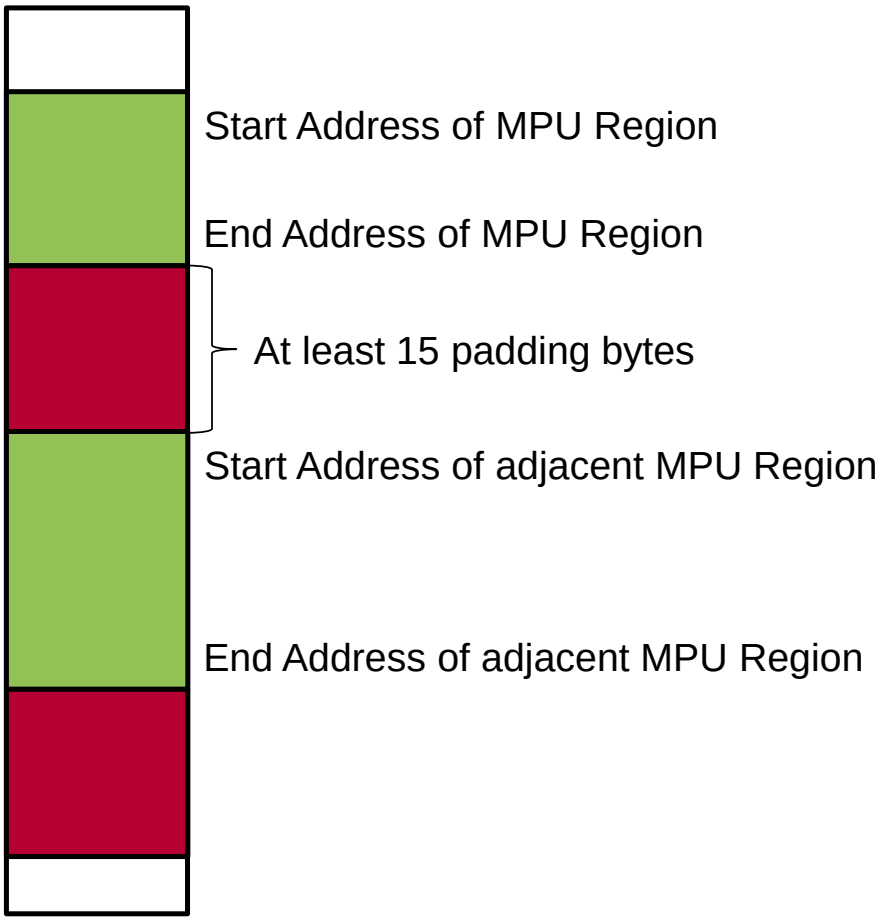

Figure 4-1 Padding bytes between MPU regions

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

MICROSAR 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 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 with HighTec (GNU) Compiler
The interrupt vector table (used in BIV) and exception vector table (used in BTV) must be aligned manually in the user linker script.

The following example shows how the interrupt vector table (of Core0) can be included and aligned to a 0x2000 byte boundary:

```
. = ALIGN(8192);
#define OS_LINK_INTVEC_CODE
#include "Os_Link_Core0.ld"
```

The following example shows how the exception vector table (of Core0) can be included and aligned to a 0x100 byte boundary:

```
. = ALIGN(256);
#define OS_LINK_EXCVEC_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 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.

4.6.1 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

4.6.2 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

5 Contact

Visit our website for more information on

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

www.vector.com