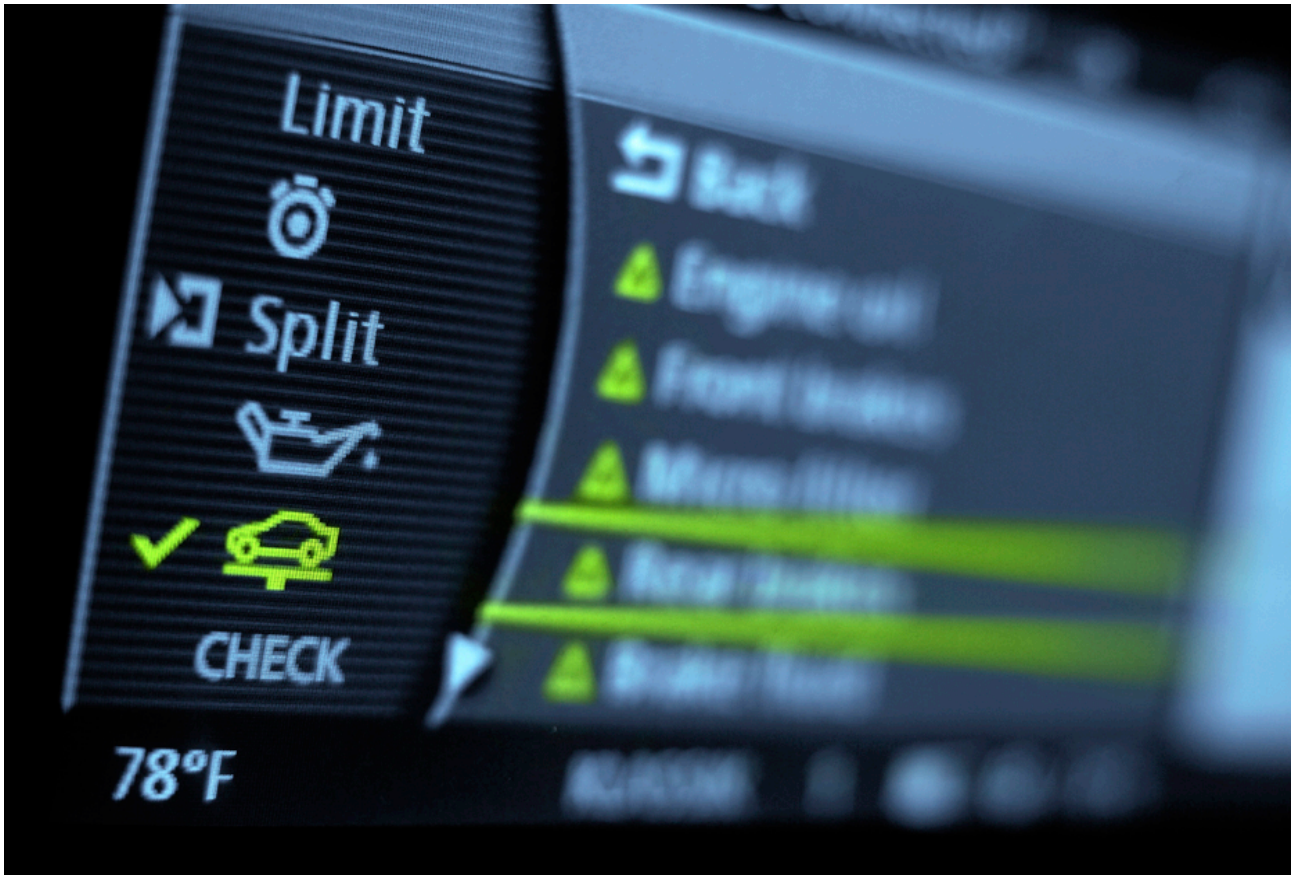
Elektrobit

EB tresos® AutoCore OS architecture notes Cortex-M

product release 6.1



Elektrobit Automotive GmbH
Am Wolfsmantel 46
91058 Erlangen, Germany
Phone: +49 9131 7701 0
Fax: +49 9131 7701 6333
Email: info.automotive@elektrobit.com

Technical support

https://www.elektrobit.com/support

Legal disclaimer

Confidential information.

Table of Contents

Begin here

1. Overview

These architecture notes provide details of the target hardware as it is used by EB tresos AutoCore OS. It is assumed that you are already familiar with the contents of the related user documentation [ASCOS_DOCUMENTATION].

In the following list, you find a summary of each chapter in this document with recommendations for reading:

- Chapter 1, “Overview of the Cortex-M architecture“ provides an overview of the aspects of the target hardware architecture that are relevant for EB tresos AutoCore OS. You should read this chapter in full.
- Chapter 2, “Supported derivatives“ provides details about the individual derivatives that EB tresos AutoCore OS supports. Each derivative lists the memory protection and timer hardware that is supported. You should read the sections of this chapter that are relevant for the target hardware that you use in your project.
- Chapter 3, “Timers“ describes the timers supported by EB tresos AutoCore OS. You should read the sections for the timers that are listed for your derivative.
- Chapter 4, “Memory protection“ describes the memory protection hardware and implementation. You should read the sections that are listed for your derivative.
- Chapter 5, “Implementation details“ provides details of the implementation of EB tresos AutoCore OS on the target hardware. You should read the parts of this chapter that are relevant for your needs.
- Chapter 6, “Reference manuals used for development“ lists the hardware reference manuals that were used during the implementation of EB tresos AutoCore OS. The bibliography gives further details about the documents, including revision and publication date where available.

2. Typography and style conventions

The signal word *WARNING* indicates information that is vital for the success of the configuration.

WARNING	**Source and kind of the problem** What can happen to the software? What are the consequences of the problem? How does the user avoid the problem?

The signal word *NOTE* indicates important information on a subject.

NOTE **Important information**

Gives important information on a subject

The signal word *TIP* provides helpful hints, tips and shortcuts.

TIP **Helpful hints**

Gives helpful hints

Throughout the documentation, you find words and phrases that are displayed in **bold**, *italic*, or `monospaced` font.

To find out what these conventions mean, see the following table.

All default text is written in Arial Regular font.

Font	Description	Example
Arial italics	Emphasizes new or important terms	The *basic building blocks* of a configuration are module configurations.
Arial boldface	GUI elements and keyboard keys	1. In the **Project** drop-down list box, select Project_A. 2. Press the **Enter** key.
Monospaced font (Courier)	User input, code, and file directories	The module calls the `BswM_Dcm_RequestSessionMode()` function. For the project name, enter `Project_Test`.
Square brackets [ ]	Denotes optional parameters; for command syntax with optional parameters	`insertBefore [<opt>]`
Curly brackets {}	Denotes mandatory parameters; for command syntax with mandatory parameters	`insertBefore {<file>}`
Ellipsis …	Indicates further parameters; for command syntax with multiple parameters	`insertBefore [<opt>…]`
A vertical bar |	Indicates all available parameters; for command syntax in which you select one of the available parameters	`allowinvalidmarkup {on|off}`

1. Overview of the Cortex-M architecture

This chapter gives a short overview of the Cortex-M architecture. It is limited to the points that are relevant for EB tresos AutoCore OS. For detailed information, see www.arm.com.

1.1. CPU modes

Cortex-M family CPUs execute code in two distinct modes of operation: thread mode and handler mode. Thread mode is the normal mode of operation and the mode the processor starts up in. It can be configured to be either *privileged* or *non-privileged*. When an interrupt or exception occurs, the processor switches to handler mode. Handler mode is always *privileged*.

1.2. Register set

The Cortex-M architecture contains 13 general purpose registers `R0`-`R12`. Three additional registers have a special usage model:

`R13`
: `R13` is the stack pointer and is also referred to as `SP`. It is used implicitly in instructions that access the stack, such as `push`. The stack pointer is a *banked* register.

`R14`
: `R14` is the link register and is also referred to as `LR`. It is used implicitly in function call instructions such as `bl` and `blx`. After executing a function call, `LR` contains the address of the instruction immediately after the function call instruction.

`R15`
: `R15` is the program counter and is also referred to as `PC`. It contains the address of the instruction to be executed. The CPU advances the program counter during execution of an instruction. The program counter is used implicitly in flow control instructions such as `b` and `bl`.

There are additional registers that contain the program status:

`xPSR`
: `xPSR` is the Program Status Register. It is a composite of three subregisters:

  - `APSR` (Application Program Status Register) mainly contains the flags used for conditional execution and conditional branches.
  - `IPSR` (Interrupt Program Status Register) contains the currently active exception, or 0 when executing in thread mode.

- `EPSR` (Execution Program Status Register) contains the T bit to indicate that the processor executes Thumb instructions as well as ICI/IT fields to implement interrupt-continuable load/store instructions and the `it` instruction.

1.3. The system registers

The Cortex-M architecture contains several system registers. Those that are relevant to EB tresos AutoCore OS are discussed here. See the Cortex-M architecture manuals for a full description.

`PRIMASK`
: `PRIMASK` is a 1-bit register. Setting it to 1 raises the processor's execution priority to 0.

`FAULTMASK`
: `FAULTMASK` is a 1-bit register. Setting it to 1 raises the processor's execution priority to -1.

`BASEPRI`
: `BASEPRI` is an 8-bit register. It changes the priority level required for exception preemption. However, it only is active when `BASEPRI` contains a lower value than the unmasked priority level of the currently executing software.

`CONTROL`
: `CONTROL` determines the following:

  - Which privilege level is used when in thread mode.
  - Which stack pointer is used when in thread mode.
  - Whether the FPU is enabled or disabled.

`VTOR`
: `VTOR` holds the vector table address.

`ICSR`
: The Interrupt Control and State Register `ICSR` provides software control of the NMI, PendSV, and SysTick exceptions, and provides interrupt status information.

`SHCSR`
: System Handler Control and State Register `SHCSR` controls and provides the active and pending status of system exceptions.

`CFSR`
: Configurable Fault Status Register `CFSR` provides information about UsageFault, BusFault and MemManage exception.

`HFSR`
: HardFault Status Register `HFSR` provides cause of HardFault.

`MMFAR`
: MemManage Fault Address Register `MMFAR` shows the address of the memory location that caused an MPU fault.

`BFAR`
: BusFault Address Register `BFAR` shows the address associated with a precise data access fault.

When a floating point unit is available, additional registers are used:

`CPACR`
: Coprocessor Access Control Register `CPACR` specifies the access privileges for coprocessors.

`FPCCR`
: Floating Point Context Control Register `FPCCR` holds control data for the floating point unit.

1.4. Exceptions

The Cortex-M family implements eight different exceptions:

`Reset`
: The Reset exception is triggered if the CPU is reset.

`NMI`
: The NMI is the non-maskable interrupt which can be triggered both by hardware and by software.

`HardFault`
: The HardFault exception denotes severe and often unrecoverable failures, for example an exception that occurs within the execution of an exception handler.

`MemManage`
: If a memory protection violation occurs, a MemManage exception is triggered. MemManage accounts for violations of both data and instruction memory transactions.

`BusFault`
: The BusFault exception is triggered if faults are detected during memory transactions, for example on the system buses.

`UsageFault`
: The UsageFault exception denotes errors during the execution of an instruction. Possible causes include division by zero, executing an undefined instruction or misaligned memory accesses.

`DebugMonitor`
: The DebugMonitor exception is generated if a `BKPT` (software breakpoint) instruction is executed.

`SVCall`
: The execution of an `SVC` (supervisor call) instruction triggers an SVCall exception.

Additionally, Cortex-M family CPUs support two system level-interrupts:

`PendSV`

PendSV can be used for system calls. In contrast to the SVCall exception, this interrupt is triggered by setting a bit in a system register `ICSR`.

`SysTick`

The SysTick interrupt is generated by the system timer, a 24-bit counter integrated into the Cortex-M processor core.

2. Supported derivatives

EB tresos AutoCore OS supports the Cortex-M derivatives listed in this chapter.

2.1. Broadcom BCM89107

The Broadcom BCM89107 SoC features one Cortex-M7 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Fixed at 400 MHz. TIM module runs at 50 MHz and CCT module runs at 10 kHz.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	► One Free running Timer Unit (TIM) module. The module provides 1 AUTOSAR counter. ► Four Compare/Capture Timer (CCT) modules (CCT0, CCT1, CCT2 and CC3). Each module provides 1 AUTOSAR counter.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	BroadR-Eye Camera/Display Endpoint Microcontroller Technical Reference Manual [BCM89107_RM103]

Table 2.1. BCM89107 hardware summary

2.2. Cypress Semiconductor CYT2B75

The Cypress Semiconductor CYT2B75 SoC features one Cortex-M4 core for primary processing and one Cortex-M0+ for peripheral and security processing. The CYT2B75 variant of AutoCore OS is designed to run on the general-purpose Cortex-M4 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv4 with 32 single-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	CYT2B75 has one TCPWM module (TCPWM0). TCPWM0 module has three timer groups (0-2), each having multiple channels. For AutoCore Os, timer group 2 is supported. It provides four AUTOSAR counters using channels 0-3.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 8 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	CYT2B7_Datasheet_32-bit_Arm_Cortex-M4F_Microcontroller_TRAVEO_T2G_-Family [002-18043]

Table 2.2. CYT2B75 hardware summary

2.2.1. TCPWM interrupt mapping on CYT2B75

When you configure a TCPWM timer in EB tresos AutoCore OS, its corresponding system interrupt is mapped to a CPU interrupt of the Cortex-M4 core. Depending on which of the TCPWM timers you configure a different CPU interrupt will be used. This mapping of TCPWM system interrupts to CPU interrupts is handled automatically by EB tresos AutoCore OS according to Table 2.5, “TCPWM system interrupt mapping”. CPU interrupts of configured timers are reserved for EB tresos AutoCore OS.

Timer	System Interrupt	CPU Interrupt Vector
`TCPWM0_2_00`	349 (`tcpwm_0_interrupts_512_IRQn`)	`I00_EXT0`
`TCPWM0_2_01`	350 (`tcpwm_0_interrupts_513_IRQn`)	`I01_EXT1`
`TCPWM0_2_02`	351 (`tcpwm_0_interrupts_514_IRQn`)	`I02_EXT2`
`TCPWM0_2_03`	352 (`tcpwm_0_interrupts_515_IRQn`)	`I03_EXT3`

Table 2.3. TCPWM system interrupt mapping

2.3. Cypress Semiconductor CYT3DL

The Cypress Semiconductor CYT3DL SoC features one Cortex-M7 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	One TCPWM module (TCPWM0) using timer group 2. It provides eight AUTOSAR counters using channels 0-7.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual [002-25800]

Table 2.4. CYT3DL hardware summary

2.3.1. TCPWM interrupt mapping on CYT3DL

When you configure a TCPWM timer in EB tresos AutoCore OS, its corresponding system interrupt is mapped to a CPU interrupt of the Cortex-M7 core. Depending on which of the TCPWM timers you configure a different CPU interrupt will be used. This mapping of TCPWM system interrupts to CPU interrupts is handled automatically by EB tresos AutoCore OS according to Table 2.5, “TCPWM system interrupt mapping”. CPU interrupts of configured timers are reserved for EB tresos AutoCore OS.

Timer	System Interrupt	CPU Interrupt Vector
`TCPWM0_2_00`	680 (`tcpwm_0_interrupts_512_IRQn`)	`I00_EXT0`
`TCPWM0_2_01`	681 (`tcpwm_0_interrupts_513_IRQn`)	`I01_EXT1`
`TCPWM0_2_02`	682 (`tcpwm_0_interrupts_514_IRQn`)	`I02_EXT2`
`TCPWM0_2_03`	683 (`tcpwm_0_interrupts_515_IRQn`)	`I03_EXT3`
`TCPWM0_2_04`	684 (`tcpwm_0_interrupts_516_IRQn`)	`I04_EXT4`
`TCPWM0_2_05`	685 (`tcpwm_0_interrupts_517_IRQn`)	`I05_EXT5`
`TCPWM0_2_06`	686 (`tcpwm_0_interrupts_518_IRQn`)	`I06_EXT6`

Timer	System Interrupt	CPU Interrupt Vector
TCPWM0_2_07	687 (tcpwm_0_interrupts_519_IRQn)	I07_EXT7

Table 2.5. TCPWM system interrupt mapping

2.4. Cypress Semiconductor CYT4BB

The Cypress Semiconductor CYT4BB SoC features two Cortex-M7 cores and a Cortex-M0+. The CYT4BB variant of EB tresos AutoCore OS is designed to run on the general-purpose Cortex-M7 cores. Currently only a single-core configuration is supported.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	CYT4BB has one TCPWM module (TCPWM0). TCPWM0 module has three timer groups (0-2), each having multiple channels. For EB tresos AutoCore OS, timer group 2 is supported. It provides eight AUTOSAR counters using channels 0-7.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	TRAVEO™ T2G Automotive Body Controller High Family Architecture Technical Reference Manual [002-24401]

Table 2.6. CYT4BB hardware summary

2.4.1. TCPWM interrupt mapping on CYT4BB

When you configure a TCPWM timer in EB tresos AutoCore OS, its corresponding system interrupt is mapped to a CPU interrupt of the Cortex-M7 core. Depending on which of the TCPWM timers you configure a different CPU interrupt will be used. This mapping of TCPWM system interrupts to CPU interrupts is handled automatically by EB tresos AutoCore OS according to Table 2.7, “TCPWM system interrupt mapping”. CPU interrupts of configured timers are reserved for EB tresos AutoCore OS.

Timer	System Interrupt	CPU Interrupt Vector
`TCPWM0_2_00`	426 (`tcpwm_0_interrupts_512_IRQn`)	`I00_EXT0`
`TCPWM0_2_01`	427 (`tcpwm_0_interrupts_513_IRQn`)	`I01_EXT1`
`TCPWM0_2_02`	428 (`tcpwm_0_interrupts_514_IRQn`)	`I02_EXT2`
`TCPWM0_2_03`	429 (`tcpwm_0_interrupts_515_IRQn`)	`I03_EXT3`

Timer	System Interrupt	CPU Interrupt Vector
TCPWM0_2_04	430 (tcpwm_0_interrupts_516_IRQn)	I04_EXT4
TCPWM0_2_05	431 (tcpwm_0_interrupts_517_IRQn)	I05_EXT5
TCPWM0_2_06	432 (tcpwm_0_interrupts_518_IRQn)	I06_EXT6
TCPWM0_2_07	433 (tcpwm_0_interrupts_519_IRQn)	I07_EXT7

Table 2.7. TCPWM system interrupt mapping

2.5. Cypress Semiconductor CYT4BF

The Cypress Semiconductor CYT4BF SoC features two Cortex-M7 cores and a Cortex-M0+. The CYT4BF variant of EB tresos AutoCore OS is designed to run on the general-purpose Cortex-M7 cores. Currently only a single-core configuration is supported.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	CYT4BF has two TCPWM modules, each with multiple groups and channels. For EB tresos AutoCore OS, one TCPWM module (TCPWM1) using timer group 2 is supported. It provides eight AUTOSAR counters using channels 0-7.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	TRAVEO™ T2G Automotive Body Controller High Family Architecture Technical Reference Manual [002-24401]

Table 2.8. CYT4BF hardware summary

2.5.1. TCPWM interrupt mapping on CYT4BF

When you configure a TCPWM timer in EB tresos AutoCore OS, its corresponding system interrupt is mapped to a CPU interrupt of the Cortex-M7 core. Depending on which of the TCPWM timers you configure a different CPU interrupt will be used. This mapping of TCPWM system interrupts to CPU interrupts is handled automatically by EB tresos AutoCore OS according to Table 2.9, “TCPWM system interrupt mapping”. CPU interrupts of configured timers are reserved for EB tresos AutoCore OS.

Timer	System Interrupt	CPU Interrupt Vector
`TCPWM1_2_00`	537 (`tcpwm_1_interrupts_512_IRQn`)	`I00_EXT0`
`TCPWM1_2_01`	538 (`tcpwm_1_interrupts_513_IRQn`)	`I01_EXT1`
`TCPWM1_2_02`	539 (`tcpwm_1_interrupts_514_IRQn`)	`I02_EXT2`
`TCPWM1_2_03`	540 (`tcpwm_1_interrupts_515_IRQn`)	`I03_EXT3`
`TCPWM1_2_04`	541 (`tcpwm_1_interrupts_516_IRQn`)	`I04_EXT4`

Timer	System Interrupt	CPU Interrupt Vector
TCPWM1_2_05	542 (tcpwm_1_interrupts_517_IRQn)	I05_EXT5
TCPWM1_2_06	543 (tcpwm_1_interrupts_518_IRQn)	I06_EXT6
TCPWM1_2_07	544 (tcpwm_1_interrupts_519_IRQn)	I07_EXT7

Table 2.9. TCPWM system interrupt mapping

2.6. Marvell MV88Q5050

The Marvell MV88Q5050 SoC features one Cortex-M7 core.

ARM architecture	ARMv7-M
Floating point support	No floating point support.
CPU clock speed	Fixed at 250 MHz. TimerN module runs at 25 MHz
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	One TimerN module. The module provides 3 AUTOSAR counters.
Time stamp	Configurable; uses a timer driver.
Memory protection	No support for memory protection.
Atomics support	Exclusive load and store
Hardware manual	► MV88Q5050 Functional Specification [FS_MV88Q5050] ► MV88Q5050 Register Specification [RS_MV88Q5050]

Table 2.10. MV88Q5050 hardware summary

2.7. Marvell MV88Q5072

The Marvell MV88Q5072 SoC features one Cortex-M7 core.

ARM architecture	ARMv7-M
Floating point support	No floating point support.
CPU clock speed	Fixed at 350 MHz. TimerN module runs at 250 MHz
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	One TimerN module. The module provides 3 AUTOSAR counters.
Time stamp	Configurable; uses a timer driver.
Memory protection	No support for memory protection.
Atomics support	Exclusive load and store
Hardware manual	► MV88Q5072 Functional Specification [MV-S111784-00] ► MV88Q5050 Register Specification [MV-S111857-00]

Table 2.11. MV88Q5072 hardware summary

2.8. Marvell MV88Q6113

The Marvell MV88Q6113 SoC features one Cortex-M7 core.

ARM architecture	ARMv7-M
Floating point support	No floating point support.
CPU clock speed	Fixed at 350 MHz. TimerN module runs at 250 MHz
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	One TimerN module. The module provides 3 AUTOSAR counters.
Time stamp	Configurable; uses a timer driver.
Memory protection	No support for memory protection.
Atomics support	Exclusive load and store
Hardware manual	MV88Q6113 Register Specification [MV-S111787-00]

Table 2.12. MV88Q6113 hardware summary

2.9. Microchip SAME5X

The Microchip SAME5X features one Cortex-M4 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Configurable using a PLL.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 7 (lowest).
Timers	Six Timer/Counter modules (TC0/TC1, TC2/TC3 and TC4/TC5). A pair of adjacent modules provides one AUTOSAR 32-bit counter.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 8 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	SAM D5X/E5X Family Data sheet [SAM_D5XE5X_DS]

Table 2.13. SAME5X hardware summary

2.10. NXP IMX8DXLM4

The NXP IMX8DXL SoC features the following hardware:

- A cluster of two Cortex-A35 cores.
- A general-purpose Cortex-M4 core.
- A Cortex-M4 core as part of the System Control Unit (SCU).

The IMX8DXLM4 variant of EB tresos AutoCore OS is designed to run on the general-purpose Cortex-M4 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv4 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup; timer frequencies are controlled by the SCU firmware.
Interrupt controller	Nested vectored interrupt controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	- One LPIT module with four channels. It provides three AUTOSAR counters. - One TPM module with six channels. It provides six AUTOSAR counters.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7 PMSA with 8 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	i.MX 8DualXLite Applications Processor Reference Manual [IMX8DXLRM]

Table 2.14. IMX8DXLM4 hardware summary

2.11. NXP IMX8QMM4

The NXP IMX8QM SoC features the following hardware:

- A cluster of two Cortex-A72 cores.
- A cluster of four Cortex-A53 cores.
- A general-purpose Cortex-M4 core.
- A Cortex-M4 core as part of the System Control Unit (SCU).

The IMX8QMM4 variant of EB tresos AutoCore OS is designed to run on the general-purpose Cortex-M4 core in single-core configuration.

ARM architecture	ARMv7-M
Floating point support	ARM FPv4 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup; timer frequencies are controlled by the SCU firmware.
Interrupt controller	Nested vectored interrupt controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	- One LPIT module with four channels. It provides three AUTOSAR counters. - One TPM module with six channels. It provides six AUTOSAR counters.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7 PMSA with 8 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	i.MX 8QuadMax Applications Processor Reference Manual [IMX8QMRM]

Table 2.15. IMX8QMM4 hardware summary

2.12. NXP IMX8QXPM4

The NXP IMX8QXP SoC features the following hardware:

- A cluster of four Cortex-A35 cores.
- A general-purpose Cortex-M4 core.
- A Cortex-M4 core as part of the System Control Unit (SCU).

The IMX8QXPM4 variant of EB tresos AutoCore OS is designed to run on the general-purpose Cortex-M4 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv4 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup; timer frequencies are controlled by the SCU firmware.
Interrupt controller	Nested vectored interrupt controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	- One LPIT module with four channels. It provides three AUTOSAR counters. - One TPM module with two channels. It provides two AUTOSAR counters.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7 PMSA with 8 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	i.MX 8DualX/8DualXPlus/8QuadXPlus Applications Processor Reference Manual [IMX8DQXPRM]

Table 2.16. IMX8QXPM4 hardware summary

2.13. NXP S32G27X

The NXP S32G274A SoC features the following hardware:

- Two clusters of two Cortex-A53 cores.
- Three Cortex-M7 cores that each feature a permanent lock-step checker core.

The S32G27X variant of EB tresos AutoCore OS is designed to run on the Cortex-M7 cores in multi-core configuration.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest). NXP MSCM is used as an interrupt router.
Timers	- Eight STM modules (STM0 to STM7). Each module provides one AUTOSAR counter. - Two PIT modules (PIT0 and PIT1). Each module provides one ticker.
Time stamp	It is implemented using the PIT timer utilizing channels 1 and 2 of module 0.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Inter-core interrupts	- Core 0: I001C0_SGI0 - Core 1: I001C1_SGI0 - Core 2: I001C2_SGI0
Atomics support	NXP Semaphores2 (SEMA42)
Hardware manual	S32G2 Reference Manual [S32G2RM]

Table 2.17. S32G27X hardware summary

The S32G27X variant of EB tresos AutoCore OS uses following system registers:

- Processor X Number Register (`MSCM_CPXNUM`)
- Interrupt Router Interrupt Generation Registers (`MSCM_IRCPxIGR0`)
- Interrupt Router Interrupt Status Registers (`MSCM_IRCPxISR0`)

2.13.1. Additional startup preconditions for S32G27X

Before you call `StartCore()`:

- Configure different domain IDs for each of the Cortex-M7 cores in NXP's Miscellaneous System Control Module (MSCM).
- Define `OS_SEMA42_DOMAIN_ID_MAP` as an array initializer for a core-indexed array. It shall contain the domain IDs that you configured in the previous step for each of the three cores.
- Start up all remaining cores before calling `StartCore()` on the initialization core[1]. On all cores except the initialization core poll for the value `OS_TRUE` in the core's entry in `OS_coreStarted`. Ensure that there is no race condition between the cores during data initialization, clock setup, and domain setup and other steps typically performed by the initialization core. One possible implementation is to perform these steps on the initialization core before starting up the other cores. Suitable sample code is provided with the application template.

[1] The initialization core is the core with core id `OsInitCoreId`. If `OsInitCoreId` is not configured, it defaults to 0.

2.14. NXP S32G399

The NXP S32G399A SoC features the following hardware:

- Two clusters of four Cortex-A53 cores.
- Four Cortex-M7 cores that each feature a permanent lock-step checker core.

The S32G399 variant of EB tresos AutoCore OS is designed to run on the Cortex-M7 cores in multi-core configuration.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest). NXP MSCM is used as an interrupt router.
Timers	- Thirteen STM modules (STM0 to STM12). Each module provides one AUTOSAR counter. **NOTE** **Unsupported STM modules** STM8 to STM12 are not supported by EB tresos AutoCore OS for S32G399 - Two PIT modules (PIT0 and PIT1). Each module provides one ticker.
Time stamp	It is implemented using the PIT timer utilizing channels 1 and 2 of module 0.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Inter-core interrupts	- Core 0: I001C0_SGI0 - Core 1: I001C1_SGI0 - Core 2: I001C2_SGI0 - Core 3: I001C3_SGI0
Atomics support	NXP Semaphores2 (SEMA42)
Hardware manual	S32G3 Reference Manual [S32G3RM]

Table 2.18. S32G399 hardware summary

The S32G399 variant of EB tresos AutoCore OS uses following system registers:

- Processor X Number Register (`MSCM_CPXNUM`)
- Interrupt Router Interrupt Generation Registers (`MSCM_IRCPxIGR0`)

- Interrupt Router Interrupt Status Registers (`MSCM_IRCPxISR0`)
- Interrupt Router Shared Peripheral Routing Control Registers (`MSCM_IRSPRCx`) that correspond to those interrupt sources which are configured to be used with EB tresos AutoCore OS

2.14.1. Additional startup preconditions for S32G399

Before you call `StartCore()`:

- Configure different domain IDs for each of the Cortex-M7 cores in NXP's Miscellaneous System Control Module (MSCM).
- Define `OS_SEMA42_DOMAIN_ID_MAP` as an array initializer for a core-indexed array. It shall contain the domain IDs that you configured in the previous step for each of the four cores.
- Start up all remaining cores before calling `StartCore()` on the initialization core[2]. On all cores except the initialization core poll for the value `OS_TRUE` in the core's entry in `OS_coreStarted`. Ensure that there is no race condition between the cores during data initialization, clock setup, and domain setup and other steps typically performed by the initialization core. One possible implementation is to perform these steps on the initialization core before starting up the other cores. Suitable sample code is provided with the application template.

[2] The initialization core is the core with core id `OsInitCoreId`. If `OsInitCoreId` is not configured, it defaults to 0.

2.15. NXP S32K14X

The NXP S32K14X family of SoCs features one Cortex-M4 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv4 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	Eight FTM modules (FTM0 to FTM7). This is the maximum number supported by this EB tresos AutoCore OS derivative. The exact number of FTM modules depends on the version of the chip used. Each module contains eight channels. The four even channels provide four AUTOSAR counters per module.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using NXP SMPU with 8 or 16 region descriptors, depending on the exact revision of the hardware.
Atomics support	Exclusive load and store
Hardware manual	S32K1xx Series Reference Manual [S32K1XXRM]

Table 2.19. S32K14X hardware summary

2.16. NXP S32K31X

The NXP S32K31X family of SoCs features one Cortex-M7 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	► One STM module (STM0). The module provides one AUTOSAR counter. ► Two PIT modules (PIT0 and PIT1). Each module provides one AUTOSAR counter or one ticker.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	S32K3xx Reference Manual [S32K3XXRM]

Table 2.20. S32K31X hardware summary

2.17. NXP S32K34X

The NXP S32K34X family of SoCs features one Cortex-M7 core.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	► Two STM modules (STM0 and STM1). Each module provides one AUTOSAR counter. ► Three PIT modules (PIT0 to PIT2). Each module provides one ticker.
Time stamp	Configurable; uses a timer driver.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	S32K3xx Reference Manual [S32K3XXRM]

Table 2.21. S32K34X hardware summary

2.18. NXP S32R45X

The NXP S32R45 SoC features the following hardware:

- Two clusters of two Cortex-A53 cores.
- Three Cortex-M7 cores that each feature a permanent lock-step checker core.

The S32R45X variant of EB tresos AutoCore OS is designed to run on the Cortex-M7 cores in multi-core configuration.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest). NXP MSCM is used as an interrupt router.
Timers	- Eight STM modules (STM0 to STM7). Each module provides one AUTOSAR counter. - Two PIT modules (PIT0 and PIT1). Each module provides one ticker.
Time stamp	It is implemented using the PIT timer utilizing channels 1 and 2 of module 0.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Inter-core interrupts	- Core 0: I001C0_SGI0 - Core 1: I001C1_SGI0 - Core 2: I001C2_SGI0
Atomics support	NXP Semaphores2 (SEMA42)
Hardware manual	S32R45 Reference Manual [S32R45RM]

Table 2.22. S32R45X hardware summary

The S32R45X variant of EB tresos AutoCore OS uses following system registers:

- Processor X Number Register (`MSCM_CPXNUM`)
- Interrupt Router Interrupt Generation Registers (`MSCM_IRCPxIGR0`)
- Interrupt Router Interrupt Status Registers (`MSCM_IRCPxISR0`)

2.18.1. Additional startup preconditions for S32R45X

Before you call `StartCore()`:

- Configure different domain IDs for each of the Cortex-M7 cores in NXP's Miscellaneous System Control Module (MSCM).
- Define `OS_SEMA42_DOMAIN_ID_MAP` as an array initializer for a core-indexed array. It shall contain the domain IDs that you configured in the previous step for each of the three cores.
- Start up all remaining cores before calling `StartCore()` on the initialization core[3]. On all cores except the initialization core poll for the value `OS_TRUE` in the core's entry in `OS_coreStarted`. Ensure that there is no race condition between the cores during data initialization, clock setup, and domain setup and other steps typically performed by the initialization core. One possible implementation is to perform these steps on the initialization core before starting up the other cores. Suitable sample code is provided with the application template.

[3] The initialization core is the core with core id `OsInitCoreId`. If `OsInitCoreId` is not configured, it defaults to 0.

2.19. NXP S32Z27XM33

The NXP S32Z2 SoC features the following hardware:

- Two clusters of four Cortex-R52 cores that may operate in split-lock configuration or as lockstep pairs.
- One Cortex-M33 core that features a permanent lock-step checker core in the System Manager Unit.
- Two Cortex-M33 cores that each feature a permanent lock-step checker core in the FlexLLCE Unit.

The S32Z27XM33 variant of EB tresos AutoCore OS is designed to run on the Cortex-M33 core of the System Manager Unit in single-core configuration.

ARM architecture	ARMv8-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	- Two STM modules (STM0 and STM2). Each module provides one AUTOSAR counter. They are called STM0 and STM1 in the configurator of EB tresos AutoCore OS. - Four PIT modules (PIT0, PIT1, PIT4 and PIT5). Each module provides one ticker. They are called PIT0 to PIT3 in the configurator of EB tresos AutoCore OS.
Time stamp	It is implemented using the PIT timer utilizing channels 1 and 2 of module 0.
Memory protection	Memory protection unit using ARMv8-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	S32Z2 Reference Manual [S32Z2RM]

Table 2.23. S32Z27XM33 hardware summary

2.20. NXP SAF85XXM7

The NXP SAF85XX SoC features the following hardware:

- One Cortex-A53 core as Computation core.
- One Cortex-M7 core as RFE core.
- One Cortex-M7 core as Control core.

The SAF85XXM7 variant of EB tresos AutoCore OS is designed to run on the Cortex-M7 control core in a single-core configuration.

ARM architecture	ARMv7-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	- Two STM modules (STM0 and STM1). Each module provides one AUTOSAR counter. - One PIT module (PIT0). It provides one ticker.
Time stamp	It is implemented using the PIT timer utilizing channels 1 and 2 of module 0.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	SAF85XX Reference Manual [SAF85XX]

Table 2.24. SAF85XXM7 hardware summary

2.21. Realtek RTL90XXA

The Realtek RTL90XXA SoC features one proprietary ARMv8-M REAL-M500 core.

ARM architecture	ARMv8-M
Floating point support	ARM FPv5 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup. DesignWare Timer module runs at 62.5 MHz.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	One DesignWare Timer module. The module provides 7 AUTOSAR counters.
Time stamp	Configurable; uses a timer driver.
Memory protection	No support for memory protection.
Atomics support	Exclusive load and store
Hardware manual	► RTL9075AAD, RTL9075AAN, RTL9072AAD, RTL9072AAN, RTL9068AAD, RTL9086ABD, RTL9086ABN, RTL9054AN, RTL9068AN Single-chip Automotive Ethernet Switch Controller with 100Base-T1/1000Base-T1 Transceiver Datasheet [RTL90XXA_DS] ► RTL9075AAD, RTL9075AAN, RTL9072AAD, RTL9072AAN, RTL9068AAD, RTL9086ABD, RTL9086ABN, RTL9054AN, RTL9068AN Single-chip Automotive Ethernet Switch Controller with 100Base-T1/1000Base-T1 Transceiver Registers Datasheet [RTL90XXA_RDS] ► DesignWare DW_apb_timers Databook [DWTIMERS_DB]

Table 2.25. RTL90XXA hardware summary

2.22. ST SR6P6M4

The ST SR6P6M4 family of SoCs features multiple Cortex-M4 cores. EB tresos AutoCore OS supports running on the Cortex-M4 core of the DME subsystem.

ARM architecture	ARMv7-M
Floating point support	ARM FPv4 with 16 double-precision registers and support for single-precision arithmetic.
CPU clock speed	Depending on clock setup.
Interrupt controller	Nested Vectored Interrupt Controller (NVIC) with configurable priorities from 1 (highest) to 15 (lowest).
Timers	► One instance of GST providing ten ARC units. Each ARC unit provides one AUTOSAR counter.
Time stamp	It is implemented using the OTC0 module of the GST.
Memory protection	Memory protection unit using ARMv7-M PMSA with 16 region descriptors.
Atomics support	Exclusive load and store
Hardware manual	SR6P6 Reference Manual [SR6P6_RM]

Table 2.26. SR6P6M4 hardware summary

2.22.1. ARC timer interrupt mapping on SR6P6M4

The interrupts associated with the ARC units of the Global System Timer are not mapped directly to the Cortex-M4 cores. Instead, you need to map them into the interrupt vector list of the respective Cortex-M4 core using the Interrupt Broadcaster (IBCM) unit. For each configured ARC timer, EB tresos AutoCore OS expects the respective interrupt to be mapped to a particular Cortex-M4 interrupt vector. You need to setup this mapping for all configured ARC timers strictly before you call `StartOS`. For the mapping requirements, see Table 2.27, “ARC timer interrupt mapping via IBCM”.

Timer	Indirect IRQ mapping vector	CPU Interrupt Vector
`ARC0_0`	145	`I000_IBCM0`
`ARC1_0`	146	`I001_IBCM1`
`ARC2_0`	147	`I002_IBCM2`
`ARC3_0`	148	`I003_IBCM3`
`ARC4_0`	149	`I004_IBCM4`
`ARC5_0`	150	`I005_IBCM5`
`ARC6_0`	151	`I006_IBCM6`

Timer	Indirect IRQ mapping vector	CPU Interrupt Vector
ARC7_0	152	I007_IBCM7
ARC8_0	153	I008_IBCM8
ARC9_0	154	I009_IBCM9

Table 2.27. ARC timer interrupt mapping via IBCM

3. Timers

This chapter lists all timer units that are supported by EB tresos AutoCore OS on the Cortex-M architecture. See your derivative's description in Chapter 2, “Supported derivatives“ to determine which of these timers are available in your derivative. There, you will also find additional information about the configuration of the timers like number of timer instances and channels supported by EB tresos AutoCore OS

3.1. Broadcom Free Running Timer Unit (TIM)

EB tresos AutoCore OS provides a driver for the Broadcom Free Running Timer Unit (TIM).

The Free running Timer Unit has two identical timer channels with configurable counter size 16 bit or 32 bit. Both the channels are configured as 32-bit counters. Channel 0 is configured to operate in free running mode, which is used for reading the real time. Channel 1 is configured to operate in single shot mode, which is used by the hardware counter of EB tresos AutoCore OS to generate interrupts.

The driver makes use of the following registers of the TIM module:

- Load Timer Value for Timer N (`TIM_TimerNLoad`)
- Current Value for Timer N (`TIM_TimerNValue`)
- Timer N Control register (`TIM_TimerNControl`)
- Timer N Interrupt Clear register (`TIM_TimerNIntClr`)
- Timer N Raw Interrupt Status register (`TIM_TimerNRIS`)

3.2. Broadcom Compare/Capture Timer (CCT)

EB tresos AutoCore OS provides a driver for the Broadcom Compare/Capture Timer (CCT).

Each timer instance has a 16 bit counter which can be configured to operate in either uni-directional or bi-directional mode. EB tresos AutoCore OS configures the timer instances to operate in uni-directional compare mode.

The driver makes use of the following registers of the CCT module:

- CCT Timer Control Register A (`TMR_CCT_CTRLA`)
- CCT Prescaler Ratio Register (`TMR_CCT_PSC_RATIO`)
- CCT Prescaler Control Register (`TMR_CCT_PSC_CTRL`)
- CCT Up-Down Counter Control Register (`TMR_CCT_UDC_CTRL`)
- CCT Up-Down Counter Trigger Register (`TMR_CCT_UDC_TRIG`)
- CCT Capture/Compare Register Channel N (`TMR_CCT_CCRN_VAL`)
- CCT Capture/Compare Control Register Channel N (`TMR_CCT_CCRN_CTRL`)
- CCT Capture/Compare Trigger Register Channel N (`TMR_CCT_CCRN_TRIG`)
- CCT Capture/Compare Interrupt Enable Register (`TMR_CCT_IRQ_CTRL`)
- CCT Capture/Compare Interrupt Status Register (`TMR_CCT_IRQ_STAT`)

Other features of the CCT modules such as input capture are not used by the EB tresos AutoCore OS driver. You cannot use the other features of a CCT module that is configured for use in EB tresos AutoCore OS. However, you can use the CCT modules that are not used by the OS for other purposes.

3.3. Cypress Timer, Counter, and PWM

EB tresos AutoCore OS provides a driver for the Cypress Timer, Counter, and Pulse Width Modulator (TCPWM) module.

Each TCPWM module comprises a number of independent counter/compare channels of either 16- or 32-bit width. The driver uses the 32-bit counter/compare channels. Each channel can provide one hardware counter and associated interrupt triggering for alarm and schedule table expiry. The 16-bit channels are not supported.

The driver makes use of the following registers of the TCPWM modules:

- TCPWMx GRPy CNTz control register (`TCPWMx_GRPy_CNTz_CTRL`)
- TCPWMx GRPy CNTz counter register (`TCPWMx_GRPy_CNTz_COUNTER`)
- TCPWMx GRPy CNTz trigger command register (`TCPWMx_GRPy_CNTz_TR_CMD`)
- TCPWMx GRPy CNTz trigger output select register (`TCPWMx_GRPy_CNTz_TR_OUT_SEL`)
- TCPWMx GRPy CNTz interrupt request register (`TCPWMx_GRPy_CNTz_INTR`)
- TCPWMx GRPy CNTz interrupt mask register (`TCPWMx_GRPy_CNTz_INTR_MASK`)

In addition, the interrupt mapping registers are used for the timer IRQs:

- Cortex-M7 CPU subsystem interrupt mapping (`CPUSS_CM7_0_SYSTEM_INT_CTL_x`)

When you configure a hardware counter in EB tresos Studio you can select one of the channels `TCPWMx_y_z` as the trigger source. The letter `x` represents the TCPWM module number. The letter `y` represents the group in the chosen TCPWM module. The letter `z` represents the channel in the chosen TCPWM group.

3.4. Marvell TimerN

EB tresos AutoCore OS provides a driver for the Marvell TimerN module.

Each TimerN module has four separate counters. The driver can use three of the counters to generate interrupts for the expiry of alarms and schedule tables. When you configure a TimerN counter as a hardware counter, the driver configures the fourth counter as a free-running counter. The driver derives the counter value for all hardware counters from the free-running counter.

The driver makes use of the following registers of the TimerN module:

- TimerN Load Count register (`TimerNLoadCount`)
- TimerN Current Value register (`TimerNCurrentValue_n`)
- TimerN Control register (`TimerNControlReg`)
- TimerN End-of-Interrupt register (`TimerNEOI`)

When you configure a hardware counter in EB tresos Studio, you can select one of the channels `TIMER_Nx_1` to `TIMER_Nx_3` as the trigger source. The letter `x` represents the TimerN module number in cases where more than one TimerN group is available.

3.5. Microchip Timer/Counter

EB tresos AutoCore OS provides a driver for the Microchip Timer/Counter (TC).

Depending on the hardware configuration, a maximum of eight TC instances can be present. Each counter of a timer instance can be configured in 8-bit, 16-bit or 32-bit mode. In 32-bit mode, two adjacent TC instances operate as a pair. EB tresos AutoCore OS configures the timer instances to operate in 32-bit mode.

The driver makes use of the following registers of the TC modules:

- TCn Control A register (`TCn_CTRLA`)
- TCn Control B Clear register (`TCn_CTRLBCLEAR`)
- TCn Control B Set register (`TCn_CTRLBSET`)
- TCn Interrupt Enable Clear register (`TCn_INTENCLR`)
- TCn Interrupt Enable Set register (`TCn_INTENSET`)
- TCn Interrupt Flag Status and Clear register (`TCn_INTFLAG`)
- TCn Synchronization Busy register (`TCn_SYNCBUSY`)
- TCn Counter Value register (`TCn_COUNT`)
- TCn Channel x Compare/Capture Value register (`TCn_CCx`)

Other features of the TC modules such as input capture are not used by the EB tresos AutoCore OS driver. You cannot use the other features of a TC module that is configured for use in EB tresos AutoCore OS. However, you can use the TC modules that are not used by the OS for other purposes.

When you configure a hardware counter in EB tresos AutoCore OS the counters provided by the TC module are named after the even-numbered timers. The timer named `TCx` in the configuration is the 32-bit timer formed by cascading timer `x` and timer `x+1`.

3.6. NXP Flex Timer Module

EB tresos AutoCore OS provides a driver for the NXP Flex Timer Module (FTM).

Each FTM instance contains a single 16-bit counter and a number of 16-bit timer compare channels. The counter provides the value for all the channels in the FTM. Each pair of channels shares a common interrupt. The driver uses the even-numbered compare channels to trigger interrupts for alarm and schedule table expiry for a single hardware counter. The counter value is derived from the common counter register.

The driver makes use of the following registers of the FTM:

- FTM Status and Control Register (`FTM_SC`)
- FTM Counter Register (`FTM_CNT`)
- FTM Modulo Register (`FTM_MOD`)
- FTM Channel n Status and Control Register (`FTM_CnSC`)
- FTM Channel n Value Register (`FTM_CnV`)
- FTM Counter Initial Value Register (`FTM_CNTIN`)
- FTM Feature Mode Selection Register (`FTM_MODE`)
- FTM Quadrature Decoder Control And Status Register (`FTM_QDCTRL`)

When you configure a hardware counter in EB tresos AutoCore OS, you can select one of the channels `FTMx_y` as the trigger source. The letter *x* represents the FTM module number. The letter *y* represents the index of the used even-numbered channel divided by 2. To configure the internal prescaler of the FTM timer modules, you have to provide a macro in the form `OS_CORTEXM_NXP_FTMx_PRESCALER`, where *x* represents the FTM instance number.

Limitations of 16-bit timers

The limited wrap-around time of the hardware means that, if the timer has a high resolution, there is a high interrupt load. To avoid this problem, configure a sufficiently high prescaler value such that the wrap-around value of the hardware is more than double your shortest periodic expiry interval.

There is also a possibility that exclusive areas in the application might cause the counter to drift because of missed wrap-arounds. If you are affected, increase the prescaler value further.

3.7. NXP Low Power Periodic Interrupt Timer

EB tresos AutoCore OS provides a driver for the NXP Low Power Periodic Interrupt Timer (LPIT). On some derivatives it is documented as Low Power Interrupt Timer.

The driver programs the timer's internal prescaler to 1.

Each LPIT timer module provides a number of timer channels. When you configure a hardware counter using an LPIT instance, the channel that you choose triggers interrupts for alarm and schedule table expiry. The driver programs the highest-numbered timer in the LPIT instance as a free-running timer. The counter values of all hardware counters that use the same LPIT instance are derived from the common free-running timer.

The driver makes use of the following registers of the LPIT module:

- LPIT Module Control Register (`LPIT_MCR`)
- LPIT Module Status Register (`LPIT_MSR`)
- LPIT Module Interrupt Enable Register (`LPIT_MIER`)
- LPIT Clear Timer Enable Register (`LPIT_CLRTEN`)
- LPIT Timer n Value Register (`LPIT_TVALn`)
- LPIT Current Timer n Value Register (`LPIT_CVALn`)
- LPIT Timer n Control Register (`LPIT_TCTRLn`)

The channels of one LPIT share a common interrupt request line. EB tresos AutoCore OS generates an ISR that demultiplexes the interrupts.

When you configure a hardware counter in EB tresos AutoCore OS, you can select one of the channels `LPITx_y` as the trigger source. The letter `x` represents the LPIT module number. The letter `y` represents the channel of the chosen LPIT module.

NOTE **Usage Limitations**

- When a channel of an LPIT module is used to provide execution budget monitoring, you cannot use the remaining channels as hardware counters. This is because of the shared interrupt.
- The timestamp timer shall use a separate timer channel and shall not be configured to share its channel with a hardware counter.

3.8. NXP Periodic Interrupt Timer

EB tresos AutoCore OS provides a driver for the NXP Periodic Interrupt Timer (PIT).

Each PIT module comprises a number of separate channels. Depending on the hardware of the respective derivative, EB tresos AutoCore OS uses a variable number of these to provide hardware counters, to provide simple ticker timers that can be used to trigger interrupts to increment software counters, or to provide the timestamp feature.

The driver makes use of the following registers of the PIT module:

- PIT Module Control Register (`PIT_MCR`)
- PIT Timer n Control Register (`PIT_TCTRLn`)
- PIT Timer n Load Value Register (`PIT_LDVALn`)

When you configure a hardware counter using a PIT instance, the channel that you choose triggers interrupts for alarm and schedule table expiry. The driver programs the highest-numbered channel in the PIT instance as a free-running counter. The counter values of all hardware counters that use the same PIT instance are derived from the common free-running counter.

When you configure a hardware counter in EB tresos AutoCore OS, you can select one of the channels `TIMER_PITx_y` as the trigger source. The letter `x` represents the PIT module number. The letter `y` represents the channel of the chosen PIT module.

When you configure a software counter in EB tresos AutoCore OS you can configure a hardware incrementer by selecting one of the channels `PITx_y` as the hardware module. The letter `x` represents the PIT module number. The letter `y` represents the channel of the chosen PIT module.

The timestamp feature uses two consecutive channels of one PIT module. Which module and which channels EB tresos AutoCore OS uses to provide the feature depends on the respective derivative. The channels are cascaded to provide a 64-bit counter. During the startup phase, the OS configures this counter to increment by one for each tick of the PIT module's clock.

3.9. NXP System Timer Module

EB tresos AutoCore OS provides a driver for the NXP STM timer module.

Each STM timer instance contains a single counter and a number of compare register channels. The counter provides the counter value for all the channels in the STM. Each compare channel can be used to provide the expiry times for alarms and schedule tables of a single hardware counter. The counter values for all hardware counters using the same STM are derived from the common counter.

The driver makes use of the following registers of the STM:

- STM Control Register (`STM_CR`)
- STM Count Register (`STM_CNT`)
- STM Channel n Control Register (`STM_CCRn`)
- STM Channel n Interupt Register (`STM_CIRn`)
- STM Channel n Compare Register (`STM_CMPn`)

When you configure a hardware counter in EB tresos AutoCore OS, you can select one of the channels `STMx_y` as the trigger source. The letter `x` represents the STM module number. The letter `y` represents the channel of the chosen STM module.

3.10. NXP Timer/PWM Module

EB tresos AutoCore OS provides a driver for the NXP Timer/PWM Module (TPM).

The driver programs the timer's internal prescaler to 1.

Each TPM provides a 32-bit counter and a set of compare channels. Each compare channel provides the trigger for alarm and schedule table expiry for a hardware counter. The counter values for the hardware counters associated with a TPM instance are all derived from the common counter.

The driver makes use of the following registers of the TPM:

- TPM Status and Control Register (`TPM_SC`)
- TPM Counter Register (`TPM_CNT`)
- TPM Modulo Register (`TPM_MOD`)
- TPM Capture and Compare Status Register (`TPM_STATUS`)
- TPM Channel n Status and Control Register (`TPM_CnSC`)
- TPM Channel n Value Register (`TPM_CnV`)

All the channels share a common interrupt It is demultiplexed by EB tresos AutoCore OS to add the appropriate handling for each channel.

When you configure a hardware counter in EB tresos AutoCore OS, you can select one of the channels `TPMx_y` as the trigger source. The letter `x` represents the TPM module number. The letter `y` represents the channel of the chosen TPM module.

NOTE **Usage Limitation**

- When a channel of a TPM module is used to provide execution budget monitoring, you cannot use the remaining channels as hardware counters. This is because of the shared interrupt.
- The timestamp timer shall use a separate timer channel and shall not be configured to share its channel with a hardware counter.

3.11. ST Global System Timer

EB tresos AutoCore OS provides a driver for the Auto-Reload/Compare (ARC) units of the ST Global System Timer (GST).

The driver uses one comparator channel of each ARC unit to trigger interrupts for alarm and schedule table expiry. The counter value is derived from the common counter in each ARC unit.

The timer names in EB tresos Studio follow the schema `ARCm_0` denoting the usage of compare register `0` of ARC module `m`. If you configure an ARC unit for use by the OS, you must not use the channels unused by the OS of this ARC unit for other purposes.

The timestamp feature uses the first counter of the Operation Time Counter (OTC) of the GST. `OTC0` is a 64-bit counter that starts from zero at power-on and continues counting afterwards, even during functional and destructive resets. During the startup phase, the OS configures `OTC0` to increment by one for each tick of the GST module clock. This is the same clock as the ARC counters.

EB tresos AutoCore OS makes use of the following registers of the GST/ARC module:

- ARCm Debug Config register (`ARCm_DEBUG_CFG`)
- ARCm Config register (`ARCm_CFG`)
- ARCm Control register (`ARCm_CTRL`)
- ARCm Reload Config register (`ARCm_RELOAD_VALUE`)
- ARCm Expiry Config register (`ARCm_EXPIRY_CFG`)
- ARCm Compare x Config register (`ARCm_COMPARE_x_CFG`)
- ARCm Interrupt Or Pulse Enable Config register (`ARCm_INT_PULSE_EN_CFG`)
- ARCm Interrupt Clear register (`ARCm_INT_CLR`)
- ARCm Event Generator Config register (`ARCm_EVENT_GEN_CFG`)
- ARCm Read register (`ARCm_READ`)

EB tresos AutoCore OS makes use of the following registers of the GST/OTC module to provide timestamp functionality:

- OTC0 Config register (`OTC0_CFG`)
- OTC0 Control register (`OTC0_CTRL`)
- OTC0 Lower Word Reload register (`OTC0_LWORD_RELOAD`)
- OTC0 Higher Word Reload register (`OTC0_HWORD_RELOAD`)
- OTC0 Event Generator Config register (`OTC0_EVENT_GEN_CFG`)
- OTC0 Increment Step Config register (`OTC0_INCSTEP_CFG`)
- OTC0 Current Lower Word Read register Without Latch (`OTC0_LWORD_READ_NOLATCH`)

- OTC0 Current Higher Word Read register Without Latch (`OTC0_HWORD_READ_NOLATCH`)

3.12. Synopsys DWTimer

EB tresos AutoCore OS provides a driver for the Synopsys DWTimer module.

Each DWTimer module has eight separate counters. The driver can use seven of the counters to generate interrupts for the expiry of alarms and schedule tables. When you configure a DWTimer counter as a hardware counter, the driver configures the eighth counter as a free-running counter. The driver derives the counter value for all hardware counters from the free-running counter.

The driver makes use of the following registers of the DWTimer module:

- Value to be loaded into Timer N (`TimerNLoadCount`)
- Current value of Timer N (`TimerNCurrentValue`)
- Control Register for Timer N (`TimerNControlReg`)

When you configure a hardware counter in EB tresos Studio you can select one of the channels `DWTIMER_x_0` to `DWTIMER_x_6` as the trigger source. The letter `x` represents the DWTimer module number in cases where more than one DWTimer group is available.

4. Memory protection

This chapter lists the different hardware options for memory protection that are available for the Cortex-M architecture. To determine which of these schemes is relevant for your derivative, see your derivative's description in Chapter 2, “Supported derivatives“.

4.1. Memory protection using the ARMv7-M MPU

4.1.1. Overview

The Protected Memory System Architecture (PMSA) of the Cortex-M CPU family is based on a memory protection unit (MPU). The MPU supports eight or sixteen memory regions. For each region, the region size, the base address and the access rights can be configured.

For the region size, only the discrete values 2^{N+1} where N>3 and N <32 are possible, e.g.

- 32 bytes
- 64 bytes
- 128 bytes
- 256 bytes
- ...
- 4 GiB

The MPU of Cortex-M CPUs optionally supports a background region. EB tresos AutoCore OS leaves the enable bit of the background region untouched. Thus, the startup code may enable or disable the background region according to the project's requirements.

The MPU registers are memory mapped. The following registers are relevant for EB tresos AutoCore OS:

`MPU_TYPE`
: The MPU Type Register. Used to check if MPU is supported.

`MPU_CTRL`
: The MPU Control Register. Enabling of the MPU and control it's behavior.

`MPU_RNR`
: The region number selects the memory region that can be programmed by `MPU_RBAR` and `MPU_RASR` control registers.

`MPU_RBAR` and `MPU_RASR`
: The Region Base Address Register and the Region Attribute and Size Register.

4.1.2. Implementation of memory protection

The operating system executes ISRs and application hooks of trusted applications in handler mode, which is a *privileged* mode. Tasks of trusted applications are executed in privileged thread mode.

Tasks of non-trusted applications are executed in *unprivileged* thread mode. ISRs and application hooks of non-trusted applications are executed in privileged handler mode.

If there are only trusted applications in the EB tresos AutoCore OS's configuration, the MPU remains disabled. If there exist non-trusted applications, the MPU is permanently enabled, also during the execution of trusted entites. The MPU is configured by the operating system as follows. It uses seven region descriptors shown in the table below:

Number	Used for	Trusted access		Non-trusted access	
		Privileged	**Non-privileged**	**Privileged**	**Non-privileged**
0	Global ROM region	Read, execute	Read, execute	Read, execute	Read, execute
1	Global RAM region	Read, write	Read, write	Read	Read
2	Global kernel-stack region	(disabled)	(disabled)	Read, write	Read
3	Global I/O region	Read, write	Read, write	(disabled)	(disabled)
4	Dynamic stack	(unused)	(unused)	Read, write	Read, write
5	Dynamic task/ISR data	(unused)	(unused)	Read, write	Read, write
6	Dynamic application data	(unused)	(unused)	Read, write	Read, write

Table 4.1. MPU regions used by EB tresos AutoCore OS

During the execution of non-trusted ISRs and application hooks, the access rights for the kernel-stack region are reduced to read-only for both privileged and non-privileged mode. When ISR or hook execution finishes, the default rights as described in Table 4.1, “MPU regions used by EB tresos AutoCore OS” are restored.

During the execution of trusted entities, the dynamic regions may still contain the region descriptors of a previously running non-trusted entity. As all of those regions may only be parts of the region covered by the global RAM region and the access permissions and memory attributes are the same, this does not cause any problems.

All memory regions programmed by EB tresos AutoCore OS which map to RAM or ROM configure the memory attributes as follows:

- normal memory
- outer and inner write-back cacheable
- write and read allocate
- shareable

The global I/O region programmed by EB tresos AutoCore OS configures the memory attributes as:

- strongly-ordered
- shareable

EB tresos AutoCore OS leaves the MPU's optional background region as is, i.e. bit PRIVDEFENA in register MPU_CTRL remains untouched.

4.1.3. Configuring memory protection with the ARMv7-M MPU

Since the ARMv7-M MPU only supports discrete sizes for memory regions, you have to configure the region sizes before generating the operating system configuration.

If you use memory protection, you configure the global region sizes by setting the following configuration parameters:

- `/OsOS/OsCORTEXMMemoryRegions/OsCORTEXMGlobalRomSize`
- `/OsOS/OsCORTEXMMemoryRegions/OsCORTEXMGlobalRamSize`
- `/OsOS/OsCORTEXMMemoryRegions/OsCORTEXMGlobalIoSize`

Moreover, you may configure private data sections for applications, tasks and ISRs by setting the following optional parameters:

- `/OsApplication/OsCORTEXMMemoryRegions/OsCORTEXMPrivateDataRegionSize`
- `/OsTask/OsCORTEXMMemoryRegions/OsCORTEXMPrivateDataRegionSize`
- `/OsIsr/OsCORTEXMMemoryRegions/OsCORTEXMPrivateDataRegionSize`

EB tresos AutoCore OS uses the configured region sizes to initialize and configure the region sizes of the MPU.

WARNING

Region size and alignment must match the configuration

Ensure that:

- Your linker script aligns your memory regions to a multiple of the configured size.
- There is sufficient free space after each region to fill the configured size.

If your linker script places your regions incorrectly, you may find that unexpected access violations occur or that the hardware fails to detect access violations.

4.1.4. Memory access violations

When the MPU detects a memory protection violation it generates a MemManage exception.

The cause of an access violation can be derived from the values of the fault status and fault address registers. These are stored in the parameters of the global error information. For details, see the description of `OS_GetErrorStatus()` in [ASCOS_DOCUMENTATION] and Section 5.5, “Exception handling”.

4.2. Memory protection using the ARMv8-M MPU

4.2.1. Overview

The Protected Memory System Architecture version 8 (PMSAv-8) of the Cortex-M CPU family is based on a memory protection unit (MPU). The number of supported MPU regions is implementation defined. For each region, the base address, the limit address, the access rights and the memory attributes can be configured. The base address and size of a region must be aligned to 32 bytes. Regions must not overlap.

The MPU of Cortex-M CPUs optionally supports a background region. EB tresos AutoCore OS leaves the enable bit of the background region untouched. Thus, the startup code may enable or disable the background region according to the project's requirements.

The MPU registers are memory mapped. The following registers are relevant for EB tresos AutoCore OS:

`MPU_TYPE`
: The MPU Type Register. Used to check if MPU is supported.

`MPU_CTRL`
: The MPU Control Register. Enabling of the MPU and control it's behavior.

`MPU_RNR`
: The region number selects the memory region that can be programmed by `MPU_RBAR` and `MPU_RLAR` control registers.

`MPU_RBAR` and `MPU_RLAR`
: The Region Base Address Register and the Region Limit Address Register.

`MPU_MAIR0` and `MPU_MAIR1`
: The Memory Attribute Indirection Registers 0 and 1.

4.2.2. Implementation of memory protection

The operating system executes ISRs and application hooks of trusted applications in handler mode, which is a *privileged* mode. Tasks of trusted applications are executed in privileged thread mode.

Tasks of non-trusted applications are executed in *unprivileged* thread mode. ISRs and application hooks of non-trusted applications are executed in privileged handler mode.

If there are only trusted applications in the EB tresos AutoCore OS's configuration, the MPU remains disabled. If there exist non-trusted applications, the MPU is permanently enabled, also during the execution of trusted entities. The MPU is configured by the operating system as follows. It uses up to twelve region descriptors shown in the table below:

Number	Used for	Trusted access		Non-trusted access	
		Privileged	Non-privileged	Privileged	Non-privileged
0	Global I/O region	Read, write	Read, write	(disabled)	(disabled)
1	Global ROM region	Read, execute	Read, execute	Read, execute	Read, execute
2	Global RAM region	Read, write	Read, write	(disabled)	(disabled)
3	Dynamic kernel-stack region	(disabled)	(disabled)	Read, write	No access
4 - 11	Dynamic stack	(disabled)	(disabled)	Read, write	Read, write
	Dynamic task/ISR data	(disabled)	(disabled)	Read, write	Read, write
	Dynamic application data	(disabled)	(disabled)	Read, write	Read, write
	Dynamic padding region 0	(disabled)	(disabled)	Read	Read
	Dynamic padding region 1	(disabled)	(disabled)	Read	Read
	Dynamic padding region 2	(disabled)	(disabled)	Read	Read
	Dynamic padding region 3	(disabled)	(disabled)	Read	Read
	Dynamic padding region 4	(disabled)	(disabled)	Read	Read

Table 4.2. MPU regions used by EB tresos AutoCore OS

The memory protection layout as shown in Table 4.2, “MPU regions used by EB tresos AutoCore OS” only reflects the maximum number of dynamic regions. Depending on the entity and the actual memory layout, some of the regions, especially of the padding regions, may not be necessary. Likewise, the order of the dynamic regions 4 - 11 may vary and depends, among other things, on the placement of the entity's data, application data and stack regions.

Deviating from the default access rights as described in Table 4.2, “MPU regions used by EB tresos AutoCore OS”, the dynamic kernel-stack region of non-trusted ISRs and application hooks has read-only access for both privileged and non-privileged mode.

During the execution of trusted entities, the dynamic regions are all disabled, because the global RAM region grants read and write access to all data and stacks and there must not be any overlapping regions. Likewise, during the execution of non-trusted entities, an entity-specific set of dynamic regions is present and enabled in the MPU and the global RAM region is disabled. In this case, the padding regions provide read-only access to all RAM apart from the entity's own data and stack.

EB tresos AutoCore OS programs the shareability domain of all memory regions to "outer shareable" apart from the global I/O region, for which the hardware ignores the shareability bit.

EB tresos AutoCore OS uses three static sets of memory attributes which are programmed to the MPU_MAIR0 register at startup as shown in Table 4.3, “MPU attributes used by EB tresos AutoCore OS”.

Index	Outer	Inner	Device
0	-	-	nGnRnE
1	Write-through, non-transient, read-allocate	Write-through, non-transient, read-allocate	-
2	Write-through, non-transient, read-/write-allocate	Write-through, non-transient, read-/write-allocate	-

Table 4.3. MPU attributes used by EB tresos AutoCore OS

EB tresos AutoCore OS does not modify any other entry of register MPU_MAIR0 nor the complete register MPU_MAIR1.

Attribute set 0 is used for the global I/O region, attribute set 1 is used for the global ROM region, attribute set 2 is used for all RAM regions, i.e. the global RAM region and all data, stack and padding regions.

EB tresos AutoCore OS leaves the MPU's optional background region as is, i.e. bit PRIVDEFENA in register MPU_CTRL remains untouched.

4.2.3. Memory access violations

When the MPU detects a memory protection violation it generates a MemManage exception.

The cause of an access violation can be derived from the values of the fault status and fault address registers. These are stored in the parameters of the global error information. For details, see the description of `OS_GetErrorStatus()` in [ASCOS_DOCUMENTATION] and Section 5.5, “Exception handling”.

4.3. Memory protection using the NXP SMPU

4.3.1. Overview

The implementation of the memory protection architecture on Cortex-M CPUs is optional. Chip manufacturers, such as NXP, may leave the standard ARMv7-M MPU out and integrate different means of memory protection. This chapter describes relevant implementation details of the NXP System Memory Protection Unit (SMPU) and how it is integrated into the Cortex-M version of EB tresos AutoCore OS.

The distinguishing feature of the SMPU, in comparison to the ARMv7-M MPU, is its placement on the crossbar switch instead of being integrated into the CPU core. This allows the SMPU to control memory accesses from different bus masters: CPU core, DMA unit, Ethernet module or a debugger.

SMPU only protects Flash and SRAM memory regions. Access to peripherals, including the SMPU itself, is controlled by the Peripheral Bridge. In practice, this means that peripherals can be accessed only in privileged mode.

The SMPU registers are memory mapped. The following registers are relevant for EB tresos AutoCore OS:

`CESR`
: Control/Error Status Register

`RGDx_WORDy`
: Region Descriptor x, Word y

`RGDAACx`
: Region Descriptor Alternate Access Control x

Depending on the derivative, the SMPU supports 8 or 16 memory regions. Each memory region may span any range aligned to 32 Bytes up to 4 GB. A region descriptor used to configure a memory region comprises four 32-bit registers (`WORD0`, `WORD1`, `WORD2`, `WORD3`) which are written into `RGDx_WORDy`.

`WORD0`
: Contains the start address (lower bound) of the memory region, aligned to 32 bytes. The five least-significant bits are fixed at zero.

`WORD1`
: Contains the last address (upper bound) of the memory region.

`WORD2`
: Contains the access permission flags for each bus master (`BM0-BM7`). Permissions for bus masters `BM4-BM7` are represented by 4 bits. Bus masters `BM0-BM3` have an additional execute flag; their permissions are represented by 5 bits.

`WORD3`
: Enables or disables the region. The process identifier and process-identifier mask are not used by EB tresos AutoCore OS.

An access to a memory location which does not belong to a configured region results in an exception.

4.3.2. Implementation of memory protection

The operating system executes ISRs and application hooks of trusted applications in handler mode, which is a *privileged* mode. Tasks of trusted applications are executed in privileged thread mode. During the execution of trusted tasks, ISRs or application hooks, the SMPU is disabled. Memory types and access rights are thus governed by the default system address map.

Tasks of non-trusted applications are executed in *unprivileged* thread mode. ISRs and application hooks of non-trusted applications are executed in privileged handler mode. During the execution of non-trusted tasks, ISRs or application hooks, the SMPU is enabled.

NOTE

The SMPU is not able to protect the kernel stack from non-trusted ISRs and application hooks, because these entities use the kernel stack and run in handler mode.

For overlapping regions the SMPU hardware implements the following prioritization scheme: Granting permission is a higher priority than denying permission. Taking this rule into account, EB tresos AutoCore OS uses the following memory regions:

Number	Usage	Core	Debugger	DMA	Ethernet
0	Entire address space (global)	`--- ---`	`rw- RW-`	`--- ---`	`--- ---`
1	Global ROM region	`r-x R-X`	`--- ---`	`--- ---`	`--- ---`
2	Global RAM region	`r-- R--`	`--- ---`	`r-- R--`	`r-- R--`
3	Global kernel-stack region	`--- RW-`	`--- ---`	`--- ---`	`--- ---`
4	Dynamic task/ISR stack	`rw- RW-`	`--- ---`	`--- ---`	`--- ---`
5	Dynamic task/ISR data	`rw- RW-`	`--- ---`	`--- ---`	`--- ---`
6	Dynamic application data	`rw- RW-`	`--- ---`	`--- ---`	`--- ---`

Table 4.4. SMPU regions used by EB tresos AutoCore OS

In the context of the table above the following words have a special meaning.

`global`
: The memory region is programmed once and never changed.

`dynamic`
: The memory region is programmed every time the OS switches to another task or ISR.

`[r,w,x,R,W,X,-]`
: Access rights are represented using 6 characters. The first 3 characters (in lower case) denote unprivileged access flags for reading, writing and executing. The second 3 characters (in upper case) denote the corresponding flags for the privileged access. The minus sign (`-`), denotes that the corresponding access type is forbidden.

The memory regions have the following meaning:

- Region 0 allows the debugger to read/write any location in the memory.
- Region 1 allows the OS kernel and all OS applications (tasks/ISRs) to read and execute code in the whole flash.

- Region 2 allows the OS kernel, all OS applications (tasks/ISRs), as well as the DMA and the Ethernet controllers to read data in the whole RAM.
- Region 3 allows the OS Kernel, the privileged (trusted) OS applications, and all ISRs to read and write to the kernel stack.
- Region 4 allows a non-privileged task or ISR to read or write its private stack area.
- Region 5 allows a non-privileged task or ISR to read or write its global variables.
- Region 6 allows a non-privileged task or ISR to read or write the global variables of the associated OS application.

4.3.3. Memory access violations

When the SMPU detects a memory protection violation it generates a BusFault exception.

The cause for the access violation can be derived from the values of the fault status and fault address registers which are stored in the parameters of the global error information. For details, see the description of `OS_GetErrorStatus()` in [ASCOS_DOCUMENTATION] and Section 5.5, “Exception handling”.

5. Implementation details

This chapter provides information about EB tresos AutoCore OS that is specific to the implementation on the Cortex-M architecture.

5.1. Function call semantics

EB tresos AutoCore OS uses the standard ARM calling conventions as defined in the "Procedure Call Standard for the ARM architecture" (AAPCS) [IHI0042E].

- `R0-R3` and `R12` are volatile registers and are preserved by the function making a function call.
- `R4-R11` are non-volatile registers and are preserved by the function being called.
- `R0-R3` can be used to pass the first four arguments. Remaining arguments are passed using the stack.
- `R0` and `R1` are used as return registers.
- `SP` is used as stack pointer.
- `LR` is used as link register.

5.2. The stack pointer

Register `SP` is used as the stack pointer. Tasks use the process stack as the stack pointer. The operating system uses the main stack as the stack pointer. Category 1 and 2 interrupts as well as hooks use the kernel stack and thus also use the main stack pointer. Do not explicitly modify this register. EB tresos AutoCore OS and the compiler manage the stack pointer.

5.3. CPU modes used by the operating system

Cortex-M family CPUs can run in two different modes, see Section 1.1, “CPU modes”. The current implementation of EB tresos AutoCore OS supports *privileged* and *non-privileged* execution.

The following list gives an overview of how each CPU mode is used within EB tresos AutoCore OS:

Handler mode
: The CPU switches to handler mode whenever it takes an exception or services an interrupt. Handler mode is always privileged. The kernel itself, as well as all ISRs and hook functions execute in handler mode and are thus always privileged.

NOTE **Execution mode of user-defined exception handlers**

If you disable exception handling using the configuration item `OsExceptionHandling`, you need to provide custom exception handlers in your application. For details see [ASCOS_DOCUMENTATION]. The custom exception handlers are executed in handler mode.

Thread mode

Thread mode is active after system reset. This means that the board startup code runs in thread mode until control is transferred to EB tresos AutoCore OS by calling `StartOS()`.

All tasks are executed in thread mode.

Thread mode usually executes in privileged mode, except for non-trusted tasks. These execute in non-privileged mode.

5.4. Interrupt controller registers

The Cortex-M architecture uses the Nested Vectored Interrupt Controller (NVIC) for all derivatives. It is deeply integrated with the CPU core to ensure low-latency interrupt processing. Thus, it is mandatory for derivatives to use it to configure interrupt handling.

The NVIC implementation uses *Set* and *Clear* registers to modify the enable and pending status of individual interrupt sources. Interrupt priorities are configured using a set of byte accessible registers.

You must ensure that you do not access and modify either the pending and enable bits or the priority register of all interrupt sources that are configured for use in EB tresos AutoCore OS. If you modify registers in the interrupt controller, you must ensure that the software that modifies the registers does not cause race conditions, either in the hardware or with EB tresos AutoCore OS.

Multi-core interrupt handling

If your derivative includes multiple Cortex-M cores, every core contains its own NVIC. In addition to the NVIC instances there is usually a derivative-specific interrupt-router component. EB tresos AutoCore OS uses the interrupt router to assign interrupt sources to specific cores and to implement cross-core interrupts.

For interrupt sources that are configured for use with EB tresos AutoCore OS, ensure that you do not change their core assignment in the respective interrupt router. For interrupt sources that are not configured for EB tresos AutoCore OS, ensure that you do not set the core assignment to a core on which EB tresos AutoCore OS is configured to run.

5.5. Exception handling

Unless configured otherwise, the EB tresos AutoCore OS kernel handles all processor exceptions. The action taken depends on the severity of the exception, ranging from quarantining the task that caused the exception to complete system shutdown. To fully comply with the AUTOSAR requirements, the kernel must handle all CPU-generated exceptions.

For the NMI exception, EB tresos AutoCore OS provides a call-out mechanism which has a default implementation of entering an endless loop. This is because in some hardware the NMI is level triggered and the exception will be active until the level of NMI is pulled low. If the level is not pulled low, the NMI exception handler will be called continuously which will corrupt the kernel stack and the behaviour is undefined. For this reason a call back is provided by the NMI exception handler to handle the level triggered NMI via the `void boardNmiCallBack(void)` function. A default implemenation is given in `boards/BOARD_NAME/boardNmiCallBack.c` file. You may just modify this function to your project needs.

5.5.1. Error reporting

The function `OS_GetErrorInfo()` returns the error information structure. When an error or protection hook is called as a result of an exception, the three *`parameter`* members of the error information structure contain the following information:

`parameter[0]`
: the program location at which the exception occurred.

`parameter[1]`
: the value of the corresponding status register. Which status register is evaluated depends on the exception that has been taken:

- `SHCSR` (system handler control and status register) for NMI, Debug, PendSV and SysTick exceptions. The value of this register is also present in the unexpected case that a reserved slot in the exception table is jumped to.
- `HFSR` (hard fault status register) for HardFault.
- `CFSR` (configurable fault status) for MemManage, BusFault and UsageFault exceptions.

`parameter[2]`
: the value of the corresponding address register if applicable.

- `MMFAR` (MemManage Fault Address Register) for MemManage exceptions.
- `BFAR` (BusFault Address Register) for BusFault exceptions.
- For all other exceptions, the value is `0`.

For details about OS_GetErrorInfo(), see the EB tresos AutoCore OS documentation [ASCOS_DOCUMENTATION].

5.5.2. Handling of UsageFault exceptions with INVPC error condition

If you perform an illegal exception return as part of the implementation of an ISR, you may get a UsageFault with the INVPC error code set. In case you configured ISRs to be called via a wrapper, not only the offending ISR will be terminated, but also the task or ISR that was originally interrupted by the offending ISR. This condition can be detected by checking for the error code INVPC in the error information obtained by calling `OS_GetErrorInfo`. The error code of a UsageFault is contained in the `CFSR` register. See Section 5.5.1, “Error reporting” for further information.

WARNING **Avoid erroneous exception returns in ISRs**

ISRs are executed in handler mode. Thus, the hardware does not prevent you from performing exception return instructions. Take care to not perform any exception returns inside of your implementation of ISRs to ensure proper system operation.

5.6. Floating-point support

The EB tresos AutoCore OS kernel controls the access and handles the context of the floating-point unit (FPU), if available. See your derivative's description in Chapter 2, “Supported derivatives“ to determine whether FPU support is included for your derivative.

The EB tresos AutoCore OS kernel only permits tasks with OsTaskUse_Hw_Fp to access the FPU. It ensures that the context is saved and restored correctly for such tasks. Any other access to the FPU, e.g. by a non-floating-point task, ISR, hook or trusted function results in a UsageFault.

Some compilers might use floating-point registers to store temporary values. For all but floating-point tasks, this will trigger a UsageFault. There are two possible ways to avoid the use of FPU registers for integer operations:

1. Use a compiler option to disable the use of the FPU. This option is only possible if your application does not use floating-point variables.
2. Add an attribute to disable FPU use to every function that does not use floating-point variables.

Refer to the compiler manual for further information.

5.7. Idle task

If there is no task ready to run, the system starts an internal idle task. The idle function results in an endless loop with global interrupts enabled. Rescheduling is triggered as soon as an interrupt or exception has been signaled.

Besides the default idle mode `IDLE_NO_HALT`, there is an additional idle mode `OS_IDLE_WFI` for the Cortex-M architecture. When you set a core's idle mode to `OS_IDLE_WFI`, the endless idle loop executes the `wfi` instruction after it has enabled the interrupts.

You can control the behavior of the idle task using `ControlIdle()`. For further information, see the API reference for `ControlIdle()` in [ASCOS_DOCUMENTATION].

5.8. Initialization before calling `main()`

AUTOSAR specifies that some OS services can be called before `StartOS()`. To allow these services to function correctly the startup code must perform the following initialization:

- Initialize the CPU registers. This is especially important for lock-step hardware.
- Initialize the stack pointer to the top of the kernel stack for the core.
- Ensure that the CPU mode is in privileged thread mode.
- Initialize the `.data` and `.bss` sections - both core-local and global.
- If present, disable the MPU.
- Initialize the `VTOR` register to the address of the kernel's exception table. This is particularly important as Cortex-M requires EB tresos AutoCore OS to be configured with a trapping kernel.
- If caches are present and shall be used, initialize and enable them. Do not use data caches in conjunction with memory protection. On multi-core hardware, do not use data caches at all.

5.9. Atomic functions support

Use of atomic functions for EB tresos AutoCore Generic is supported by the atomics module that is provided by EB tresos AutoCore Generic Base. For a standalone delivery of EB tresos AutoCore OS the underlying atomics functions may be used directly.

5.9.1. Exclusive load/store instructions

When the atomic functions use the `LDREX` and `STREX` instructions, the Cortex-M hardware places restrictions on the objects for which exclusivity is required. You must ensure that `os_atomic_t` objects are correctly aligned, sufficiently separated from other objects and are placed in memory that has all of the following attributes:

- normal memory
- inner or outer shareable

- inner write-back
- outer write-back
- read and write allocate hints
- not transient

For the ARMv7M architecture, see section "A3.4.5 Load-Exclusive and Store-Exclusive usage restrictions" in [DDI0403E_c] for details.

5.9.2. SEMA4 hardware module

The SEMA4 hardware module only works when the software that uses it executes in a *privileged* mode. If a *non-privileged* executable attempts to use the SEMA4 module, the hardware raises an exception.

Consequently, use of the hardware-specific atomics functions is restricted to tasks, ISRs, hook functions and trusted functions belonging to *trusted* OS applications. If you intend to use atomics functions in non-trusted OS applications, you must use the atomics module provided by EB tresos AutoCore Generic Base to ensure freedom from interference.

5.9.3. Hardware support for `EB_FAST_LOCK`

For Cortex-M, EB tresos AutoCore OS features an optimized implementation to ease the overhead incurred by entering and leaving exclusive areas. On Cortex-M hardware, any attempt to disable interrupts when the CPU is running in a *non-privileged* mode fails silently. Interrupts remain enabled, and the CPU does not raise an exception. To overcome this restriction, the `EB_FAST_LOCK` implementation checks if the CPU is in a *privileged* mode as a pre-condition to perform direct interrupt locking. If called from non-privileged mode, the implementation uses `SuspendAllInterrupts()` to perform interrupt locking.

`EB_FAST_LOCK` optimization is effective only for a privileged mode and doesn't affect program execution in a non-privileged mode.

For general constraints on exclusive areas, see also the EB tresos AutoCore OS documentation [ASCOS_DOCUMENTATION].

5.10. The linker script

When you write your linker script, make sure that the linker symbols mentioned in this section are correctly defined. The operating system evaluates these linker-defined symbols to set up the MPU. The symbols may be used in your startup code to initialize the global `.data` and `.bss` sections.

The linker script provides definitions of the following symbols:

- `__GLBL_ROM_START` defines the start address of the global text and const region.
- `__GLBL_ROM_END` defines the end address of the global text and const region.
- `__GLBL_RAM_START` defines the start address of the global RAM region.
- `__GLBL_RAM_END` defines the end address of the global RAM region.
- `__GLBL_IO_START` defines the start address of the global I/O region.
- `__GLBL_IO_END` defines the end address of the global I/O region.
- `__DATA_<name>` defines the start address of the application/task/ISR-specific data region. <name> is the name of the application/task/ISR.
- `__DEND_<name>` defines the end address of the application/task/ISR-specific data region. <name> is the name of the application/task/ISR.
- `__IDAT_<name>` defines the start load address of the initialization data for the application/task/ISR-specific initialized data section. <name> is the name of the application/task/ISR.
- `__IEND_<name>` defines the end load address of the initialization data for the application/task/ISR-specific initialized data section. <name> is the name of the application/task/ISR.
- `__STARTDATA` defines the start address of the non-application-specific initialized data section.
- `__ENDDATA` defines the end address of the non-application-specific initialized data section.
- `__INITDATA` defines the load address of the initialization data for the non-application-specific initialized data section.
- `__STARTBSS` defines the start address of the non-application-specific uninitialized data section.
- `__ENDBSS` defines the end address of the non-application-specific uninitialized data section.

The `__GLBL_RAM_START` / `__GLBL_RAM_END` region must at least comprise all of the RAM which is used by EB tresos AutoCore OS and all configured runnable entities.

The `__GLBL_IO_START` / `__GLBL_IO_END` region must at least comprise all of the memory-mapped I/O address space which is used by EB tresos AutoCore OS and all configured trusted runnable entities.

If you use the KEIL ARM® toolchain for your project, see the EB tresos AutoCore OS user's guide section *Support for the KEIL ARM® toolchain* for a detailed description of required linker symbols.

6. Reference manuals used for development

The following documents were used as a basis for porting the OS for Cortex-M:

- ARMv7-M Architecture Reference Manual [DDI0403E_e]
- ARMv8-M Architecture Reference Manual [DDI0553B_o]
- ARM Cortex-M7 Processor [DDI0489F]
- ARM Cortex-M4 Processor [100166-0001-04]
- ARM Cortex-M3 Processor [100165-0201-02]
- ARM Cortex-M33 Processor [10023_0100_03_en]
- CYT2B7_Datasheet_32-bit_Arm_Cortex-M4F_Microcontroller_TRAVEO_T2G_Family [002-18043]
- Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual [002-25800]
- TRAVEO™ T2G Automotive Body Controller High Family Architecture Technical Reference Manual [002-24401]
- i.MX 8DualXLite Applications Processor Reference Manual [IMX8DXLRM]
- i.MX 8QuadMax Applications Processor Reference Manual [IMX8QMRM]
- i.MX 8DualX/8DualXPlus/8QuadXPlus Applications Processor Reference Manual [IMX8DQXPRM]
- MV88Q5050 Functional Specification [FS_MV88Q5050]
- MV88Q5050 Register Specification [RS_MV88Q5050]
- MV88Q5072 Functional Specification [MV-S111784-00]
- MV88Q5050 Register Specification [MV-S111857-00]
- MV88Q6113 Register Specification [MV-S111787-00]
- S32G2 Reference Manual [S32G2RM]
- S32G3 Reference Manual [S32G3RM]
- S32K1xx Series Reference Manual [S32K1XXRM]
- S32K3xx Reference Manual [S32K3XXRM]
- S32Z2 Reference Manual [S32Z2RM]
- SAF85XX Reference Manual [SAF85XX]
- SAM D5X/E5X Family Data sheet [SAM_D5XE5X_DS]
- SR6P6 Reference Manual [SR6P6_RM]

- RTL9075AAD, RTL9075AAN, RTL9072AAD, RTL9072AAN, RTL9068AAD, RTL9086ABD, RTL9086ABN, RTL9054AN, RTL9068AN Single-chip Automotive Ethernet Switch Controller with 100Base-T1/1000Base-T1 Transceiver Datasheet [RTL90XXA_DS]
- RTL9075AAD, RTL9075AAN, RTL9072AAD, RTL9072AAN, RTL9068AAD, RTL9086ABD, RTL9086ABN, RTL9054AN, RTL9068AN Single-chip Automotive Ethernet Switch Controller with 100Base-T1/1000Base-T1 Transceiver Registers Datasheet [RTL90XXA_RDS]
- DesignWare DW_apb_timers Databook [DWTIMERS_DB]

Glossary

banked register	A register that has several instances that are accessed by means of a single name. The instance that is actually used depends on some hardware influence such as processor mode or core.
non-privileged	A mode of the CPU in which access to the CPU's control registers is forbidden by the hardware. The mode may also play a role in the memory access rights.
privileged	A mode of the CPU in which access to the CPU's control registers is permitted by the hardware. The mode may also play a role in the memory access rights.

Bibliography

[002-18043] Infineon:*CYT2B7_Datasheet_32-bit_Arm_Cortex-M4F_Microcontroller_TRAVEO_-T2G_Family*, Rev. I, October 13 2021

[002-24401] Cypress:*TRAVEO™ T2G Automotive Body Controller High Family Architecture Technical Reference Manual*, Rev. F, October 06 2021

[002-25800] Cypress:*Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual*, Rev. B, May 22 2020

[100165-0201-02] ARM:*ARM Cortex-M3 Processor*: Technical Reference Manual, 0201-02, 09 Nov 2016

[100166-0001-04] ARM:*ARM Cortex-M4 Processor*: Technical Reference Manual, 0001-04, 15 May 2020

[10023_0100_03_-en] ARM:*ARM Cortex-M33 Processor*: Technical Reference Manual, 0100-03, 19 June 2020

[ASCOS_DOCUMENTATION] *EB tresos AutoCore OS documentation*: product release 6.1, 3.1_AutoCore_OS_-documentation.pdf

[BCM89107_-RM103] Broadcom:*BroadR-Eye Camera/Display Endpoint Microcontroller Technical Reference Manual*, September 17 2019

[DDI0403E_c] ARM:*ARMv7-M Architecture Reference Manual*: ARMv7-M edition, E.c, May 2017
https://developer.arm.com/documentation/ddi0403/ec/

[DDI0403E_e] ARM:*ARMv7-M Architecture Reference Manual*: ARMv7-M edition, E.e, 15 Feb 2021
https://developer.arm.com/documentation/ddi0403/ee/

[DDI0489F] ARM:*ARM Cortex-M7 Processor*: Technical Reference Manual, F, 15 Nov 2018

[DDI0553B_o] ARM:*ARMv8-M Architecture Reference Manual*: ARMv8-M edition, B.o, 31 Mar 2021
https://developer.arm.com/documentation/ddi0553/bo/

[DWTIMERS_DB] Synopsys:*DesignWare DW_apb_timers Databook*, Revision 2.09a, June 2014

[FS_MV88Q5050] Marvell:*MV88Q5050 Functional Specification*: Internal CPU and Low-speed Bus Unit Sections, January 23 2017

[IHI0042E] ARM:*Procedure Call Standard for the ARM® Architecture*, 30th November 2012
https://developer.arm.com/documentation/ihi0042/e

[IMX8DQXPRM] NXP:*i.MX 8DualX/8DualXPlus/8QuadXPlus Applications Processor Reference Manual*, Revision 0, May 2020

[IMX8DXLRM] NXP:*i.MX 8DualXLite Applications Processor Reference Manual*, Revision C, July 2020

[IMX8QMRM] NXP:*i.MX 8QuadMax Applications Processor Reference Manual*, Revision F, October 2019

[MV-S111784-00] Marvell:*MV88Q5072 Functional Specification*: 11-Port Automotive Ethernet Switch with XFI and PCIe Interfaces and 7 Integrated 10BASE-T1 PHYs , May 08 2020

[MV-S111787-00] Marvell:*MV88Q6113 Register Specification*: 11-Port Automotive Ethernet High Bandwidth Aggregation Switch with XFI and PCIe Interface , Dezember 19 2019

[MV-S111857-00] Marvell:*MV88Q5050 Register Specification*: 11-Port Automotive Ethernet Switch with XFI and PCIe Interfaces and 7 Integrated 10BASE-T1 PHYs , May 08 2020

[RS_MV88Q5050] Marvell:*MV88Q5050 Register Specification*: Low-speed Bus Unit Section, January 23 2017

[RTL90XXA_DS] Realtek:*RTL9075AAD, RTL9075AAN, RTL9072AAD, RTL9072AAN, RTL9068AAD, RTL9086ABD, RTL9086ABN, RTL9054AN, RTL9068AN Single-chip Automotive Ethernet Switch Controller with 100Base-T1/1000Base-T1 Transceiver Datasheet*, Revision 1.5, May 2022

[RTL90XXA_RDS] Realtek:*RTL9075AAD, RTL9075AAN, RTL9072AAD, RTL9072AAN, RTL9068AAD, RTL9086ABD, RTL9086ABN, RTL9054AN, RTL9068AN Single-chip Automotive Ethernet Switch Controller with 100Base-T1/1000Base-T1 Transceiver Registers Datasheet*, Revision 1.6, January 2022

[S32G2RM] NXP:*S32G2 Reference Manual*: Supports S32G274A, S32G254A, S32G233A, S32G234M, Revision 2, April 2021

[S32G3RM] NXP:*S32G3 Reference Manual*: Supports S32G399A, S32G398A, S32G379A, S32G378A, Revision 1, November 2021

[S32K1XXRM] NXP:*S32K1xx Series Reference Manual*: Supports S32K116, S32K118, S32K142, S32K142W, S32K144, S32K144W, S32K146, and S32K148 , Revision 12.1, February 2020

[S32K3XXRM] NXP:*S32K3xx Reference Manual*: Supports S32K311, S32K312, S32K314, S32K322, S32K324, S32K328, S32K338, S32K341, S32K342, S32K344, S32K348, and S32K358 , Revision 3, October 2021

[S32R45RM] NXP:*S32R45 Reference Manual*, Revision 3, December 2021

[S32Z2RM] NXP:*S32Z2 Reference Manual*, Revision 2 Draft D, September 2022

[SAF85XX] NXP:*SAF85XX Reference Manual*, Revision 1.12, November 2022

[SAM_D5XE5X_DS] Microchip:*SAM D5X/E5X Family Data sheet*, Revision F, May 2020

[SR6P6_RM] ST Microelectronics:*SR6P6 Reference Manual*: SR6P6x 32-bit ARM® Cortex®-R52+ architecture microcontroller for automotive ASILD applications, RM0496_Rev 1, December 2022