

MCAL User Manual General

32-bit TriCore™ AURIX™ TC3xx microcontroller

About this document

Scope and purpose

This User Manual is intended to enable users to integrate the Microcontroller Abstraction Layer (MCAL) software for the TriCore™ AURIX™ family of 32-bit microcontrollers.

This document describes responsibilities of integrator in charge of integrating MCAL software with the basic software (BSW) stack. This document also provides detailed information on package installation, project creation, safety and other generic information that are common across all the modules.

Note: Detailed information about each module are provided in module specific user manuals.

Intended audience

This document is intended for anyone using the TC3xx MCAL software.

Document conventions

Table 1 Conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, table and figure captions, screen names, windows, dialog boxes, menus, sub-menus
<i>Italics</i>	Denotes variable(s) and reference(s)
Courier	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets
New	
>	Indicates that a cascading sub-menu opens when you select a menu item
[cover parentID=<alpha numeric value>]	Used for traceability completeness. Reader should ignore these.

Glossary of terms

Table 2 Glossary

Term	Description
ADC	Analog-to-digital converter
ATOM	ARU connected TOM (functional block of a microcontroller peripheral)
AUTOSAR	Automotive Open System Architecture
Bank	A Flash module contains separate banks. In the DFlash, there are one or more banks. Banks support concurrent operations with some limitations due to common logic.
BIFACES	BIFACES stands for Build and Integration Framework for Automotive Controller Embedded Software. It is an internally developed build framework for Infineon automotive microcontrollers software development.
BSW	Basic software
Burst write	The maximum amount of data that can be written with one command. The programming throughput is higher than for programming single pages. For DFlash, it is 4 pages (32 bytes).

(table continues...)

About this document
Table 2 (continued) Glossary

Term	Description
CAN	Controller Area Network
CAN FD	CAN flexible data rate
CC	Communication controller
CCM	Cluster control module (functional block of GTM)
Channel	A channel is a software exchange medium for data that are defined with the same criteria: configuration parameters, number of data elements with same size and data pointers (source, destination) or location.
CHI	Communication host interface
CMU	Clock management unit (functional block of GTM)
CPU	Central processing unit
CRC	Cyclic redundancy check
DEM	Diagnostics event manager (MCAL module)
DER	Destination error
DET	Development error tracer
DF_EEPROM	Data Flash dedicated for EEPROM emulation
DFlash	Data Flash
DIO	Digital input output
DMA	Direct memory access
DMU	Data memory unit
Driver	A driver is a BSW module located in the MCAL layer and contains the functionality to control and access an internal or an external device.
EB	Externally defined buffer for QSPI
ECC	Error correction code
EEPROM	Electrically erasable and programmable ROM (read only memory)
ERAY	FlexRay IP module (hardware)
ERU	External request unit
ESR	External service request (microcontroller pin)
ETH	Ethernet
EVADC	Enhanced versatile analog-to-digital converter
EVER	Bit indicating erase verify error
ex_r	Exclusive read access
ex_rw	Exclusive read and write access
ex_w	Exclusive write access
Fast-Mode	Triggering the watchdog hardware has to be done with a short timeout period. This mode can be used during normal operations of the ECU. For example, the watchdog hardware is configured for the Windows mode (watchdog triggering should happen within certain minimum/maximum boundaries within the timeout period) and a timeout period of 5 ms.
FCE	Flexible CRC engine

(table continues...)

About this document
Table 2 (continued) Glossary

Term	Description
FCFS	First come first serve
FEE	Flash EEPROM emulation
FEE sector	The DF_EEPROM area is logically divided into two FEE sectors for implementing the double sector algorithm. If the QS region exists then the part of DF_EEPROM is used for QS blocks and remaining is used for double sector algorithm.
FIFO	First in first out
FLS	Flash
GC	Garbage collection
GEth	Gigabit Ethernet MAC
GTM	Generic timer module (hardware)
HOH	Hardware object (transmit/receive) handle
HRH	Hardware receive handle
HSCT	High-speed communications tunnel
HSPDM	High-speed pulse density modulation module
HTH	Hardware transmit handle
I/O	Input/Output
IB	Driver defined internal buffer / channel
IFX	Infineon Technologies
Internal job	FEE driver internal management activities, which are not explicitly triggered by the upper layer
ISR	Interrupt service routine
Job	A job is composed of one or several channels with the same chip select (is not released during the processing of job). A job is considered atomic and therefore cannot be interrupted by another job. A job has an assigned priority.
(Logical) block	Smallest erasable unit (4 K) as seen by the modules user. Consists of one or more virtual pages.
LPdu	Datalink layer protocol data unit
LPM	Low power mode
MAC	Media access control
MC-ISAR	Microcontroller Infineon Software Architecture
MCU	Microcontroller unit
MII	Media independent interface
Module	The element is composed of various software units called modules. Typically each software driver is referred to as module. More explicitly, it is also referred to as software module.
MRST	Master receive slave transmit
MTL	MAC transaction layer
MTSR	Master transmit slave receive

(table continues...)

About this document
Table 2 (continued) Glossary

Term	Description
(Normal) write mode	In this mode, the maximum amount of data that can be written with one command is 8 byte (1 Page).
NVM	AUTOSAR NVRAM manager
NVRAM	Non-volatile RAM (random access memory)
NVRAM block	Management unit as seen by the NVRAM manager
OCU	Output compare unit
Off-Mode	Watchdog hardware is disabled / shut-down
OS	Operating system
Page	A page is an aligned group of data double words plus an ECC extension. It is the smallest unit that can be programmed. DFlash: 1 data double word (8 bytes) plus 22-bit ECC extension.
Peripheral	Hardware module used by a driver. A driver can use one or more peripherals.
PHY	Physical layer device (Ethernet transceiver)
Physical address	Address information in device-specific format (depending on the underlying Flash driver and device) that is used to access a logical block.
PLL	Phase lock loop
Pn_xxxxx	Port n register (xxxxx is register name)
POC	Protocol operation control
PORST	Power-on reset
PVER	Bit indicating program verify error
QS	Quasi-static
QSPI	Queued serial peripheral interface
r	Read access
RGMII	Reduced Gigabit Media Independent Interface
RMII	Reduced Media Independent Interface
rw	Read and write access
SchM	Scheduler manager (AUTOSAR module)
SCU	System control unit
Sequence	A sequence is a number of consecutive Jobs to transmit but it can be rescheduled between jobs using a priority mechanism. A sequence transmission is interruptible (by another sequence transmission) or not depending on a static configuration.
SER	Source error
Slow-Mode	Triggering the watchdog hardware can be done with a long timeout period. This mode can be used during system start-up / initialization phase. For example, the watchdog hardware is configured for toggle mode (no constraints on the point in time at which the triggering is done) and a timeout period of 10 ms.
SLSO	Programmable slave select outputs
SMU	Safety management unit
SPB	System peripheral bus

(table continues...)

About this document
Table 2 (continued) Glossary

Term	Description
SRC	Service request control register
SRI	System resource interconnect
SRN	Service request node
STC	Set trigger condition API
STM	System timer (hardware)
TBU	Time base unit (functional block of GTM)
TECQED	Triple-bit error correction and quad-bit error detection
TIM	Timer input module (functional block of a microcontroller peripheral)
TOM	Timer output module (functional block of a microcontroller peripheral)
Unconfigured block	Data block which is stored in the DFlash (DF_EEPROM), but is not contained in the currently active configuration
User Job	User requested read/write/invalidate job
VariantLT	This variant allows a mix of pre-compile time, link-time configuration parameters. The intention of this variant is to optimize the parameters configuration for an object code delivery.
VariantPB	This variant allows a mix of pre-compile time, post-build-time and link time configuration parameters. The intention of this variant is to optimize the parameters configuration for a re-loadable binary
VarinatPC	This variant allows only pre-compile configuration parameters. The intention of this variant is to optimize the parameters configuration for a source code delivery.
Virtual address	Consisting of 16-bit block number and 16-bit offset inside the logical block
Virtual page	May consist of one or several physical pages to ease handling of logical blocks and address calculation
w	Write access
WDG	Watchdog
WDGx	Watchdog timer for CPUx
WDT	Watchdog timer (hardware)
Word-line (WL)	An aligned group of bytes. In DFLASH, 512 bytes are in the single-ended mode and 256 bytes are in the complement-sensing mode.
WUT	WakeUp timer

Reference documents

This User Manual should be read in conjunction with the following documents:

- AURIX™TC3xx User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™TC39x-B Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™ TC38x Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™ TC37x Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™ TC37xEXT Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™ TC36x Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™ TC35x Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™ TC33xEXT Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- AURIX™ TC33x/TC32x Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG

About this document

- AURIX™TC3Ex Appendix to User Manual, V2.0.0, 2021-02, Infineon Technologies AG
- MC-ISAR_TC3xx_HWErrataAnalysis.xlsx, v24.0

Table of contents

Table of contents

	About this document	1
	Table of contents	7
1	Generic information	9
1.1	Application integration	9
1.1.1	Package installation and project creation	9
1.1.1.1	Installing MCAL package	9
1.1.1.2	EB tresos™: license handling	12
1.1.1.3	EB tresos: Eclipse handling	13
1.1.1.4	Creating an EB tresos project	14
1.1.2	Typical module parameter configuration	19
1.1.2.1	Configuring parameters	19
1.1.2.2	Generating configuration files	21
1.1.3	Building and linking to generate final application binary	22
1.1.3.1	Building package	23
1.1.3.2	Release folder structure	23
1.1.3.3	Makefile adaptations	25
1.1.3.4	Linker script adaptations	26
1.1.3.5	Initiation of compiling and linking	27
1.1.3.6	Flashing the image	29
1.2	MCAL Initialization	36
1.3	Multicore support	36
1.3.1	Multicore initialization	38
1.3.2	Multicore de-initialization	39
1.3.3	Multicore runtime	40
1.3.4	Multicore interrupts	41
1.3.5	Resource Manager (ResourceM)	42
1.3.5.1	Description	42
1.3.5.2	Configuration containers and parameters	42
1.3.5.2.1	Container: ResourceM	42
1.3.5.2.2	Container: ResourceMAllocation	43
1.3.5.2.3	Container: ResourceMMcalConfig	44
1.3.5.2.4	Container: ResourceMMcalCore	45
1.3.5.3	Code generator plugin files	45
1.3.5.4	Example usage	46
1.3.5.4.1	Configuring master core	46
1.3.5.4.2	Configuring cores	47
1.3.5.4.3	Configuring resources to core	47
1.4	User mode execution	48
1.4.1	Atomic update of the Supervisor mode register when CPU is in the User-1 mode	48

Table of contents

1.4.2	Protected Supervisor mode register update when CPU is in the Supervisor mode	49
1.4.3	Protected Supervisor mode register update when CPU is in the User-1 mode	50
1.4.4	Read Supervisor mode register when CPU is in the User-1 mode	51
1.4.5	Write to Supervisor mode register when CPU is in the User-1 mode	52
1.4.6	Optimization technique for supervisor mode registers access	53
1.5	Execution context of functions	54
1.6	Memory mapping	55
1.6.1	Memory mapping for code	56
1.6.2	Memory mapping for callout code	56
1.6.3	Memory mapping for configuration data	57
1.6.4	Memory mapping for constants	57
1.6.5	Memory mapping for variables	58
1.7	Variation point support	58
1.7.1	Configuring the Sidebar view	58
1.7.2	Configuring EcuC module	61
1.7.3	Configuring modules	62
1.7.4	Post-build variant multiplicity and Multiplicity configuration class	65
1.7.5	Variation point command line code generation	65
1.7.5.1	Importing and exporting configuration data	65
1.7.5.2	Generating project via command line	67
2	Safety view	69
2.1	Safety objectives	69
2.2	Common Assumptions of Use (AoU)	69
2.3	Mapping of MCAL to external safety mechanism (ESM)	73
2.3.1	Reporting of unintended service requests	75
3	Deviations and limitations	77
3.1	Software specification deviations	77
3.2	Limitations	77
	Revision history	78
	Disclaimer	82

1 Generic information

1.1 Application integration

This section describes the typical workflow required to integrate MCAL modules with the user environment.

Note: *This Generic information chapter is applicable to BASIC, CD, COM-E and DEMO.*

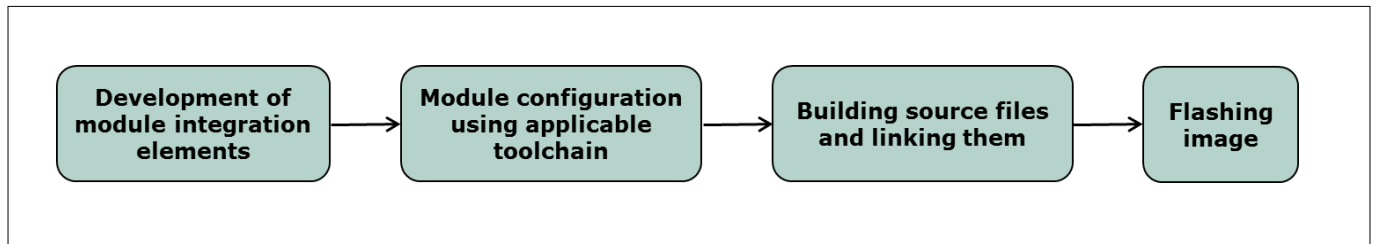


Figure 1 Application integration flow

- Being module specific, details of integration elements are provided in sections from *Integration hints* found in module chapters.
- Rest of the workflow topics being generic, are described in the subsequent sections.

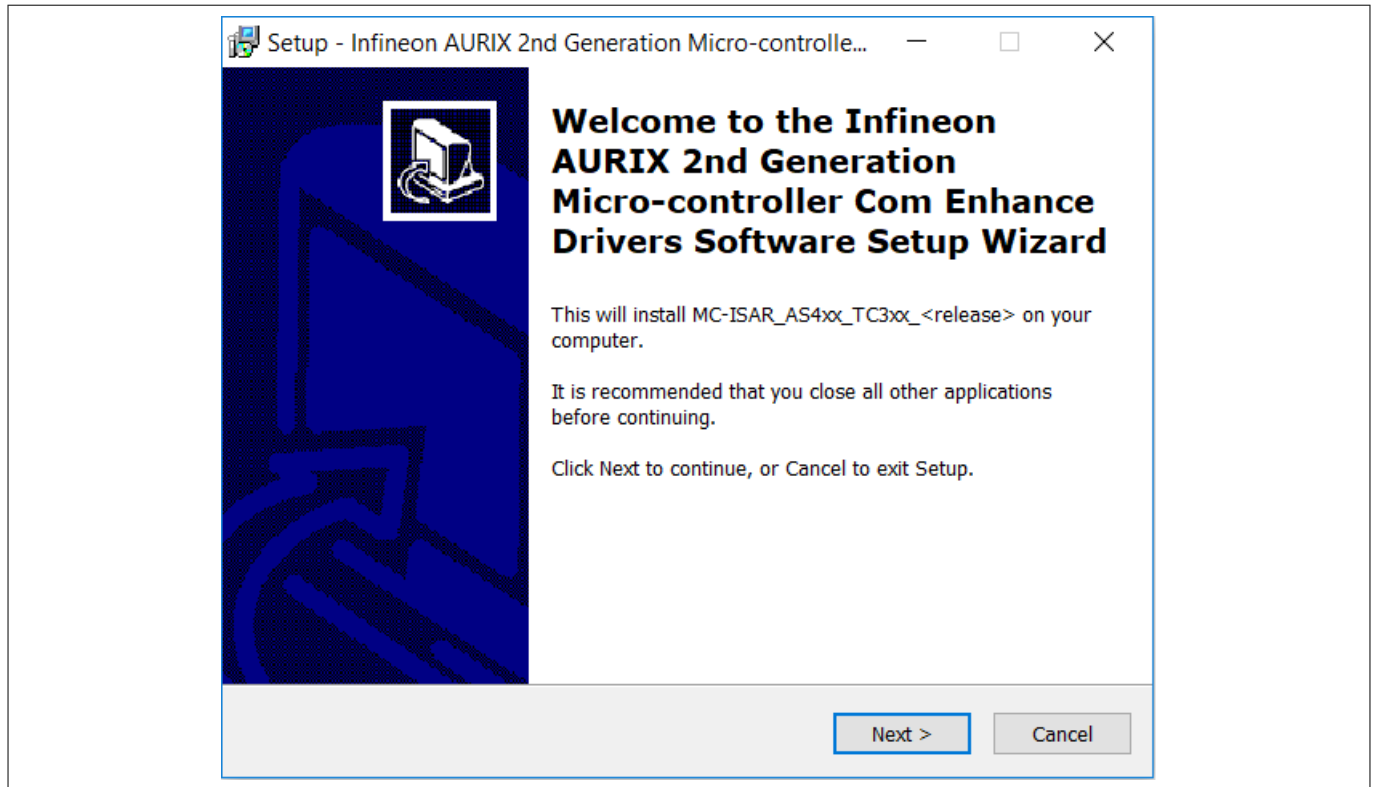
1.1.1 Package installation and project creation

This section describes how the modules may be installed and projects created.

1.1.1.1 Installing MCAL package

Note: *In this chapter, the options shown in the images are only for the purpose of illustration. The actual list/names should be referred in the tool.*

The MCAL code may be extracted to any directory, from the archive. The procedure for installing the software is as follows:

**Figure 2** **Setup window**

1. Run the exe file provided with the package.

Note: Similarly, you can select CD, Basic and/or DEMO executable files from the respective package folders.

2. Click **Next**. The **Setup** window is displayed.
3. Select a target to install. Multiple devices can be selected at the same time.
4. Click **Next**.

Note: Selecting the option **All** installs the software on all the listed Targets.

Note: The list of devices shown in the above image is only an example.

Note: AS4xx shown in the above image represents AS422 or/and AS440 based on the autosar version supported in the package.

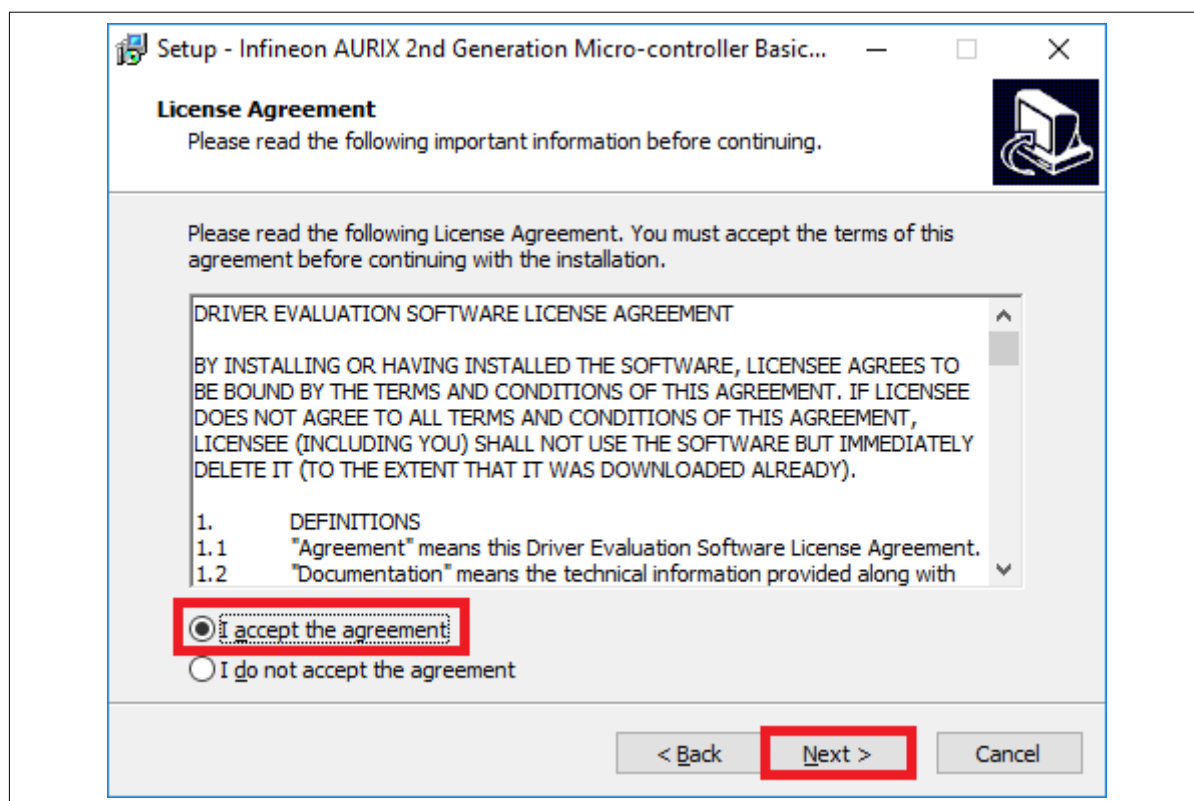


Figure 3 License agreement

5. Select the **I accept the agreement** option to accept the terms and conditions. Note that the installation does not take place if this option is not selected.
6. Click **Next**.

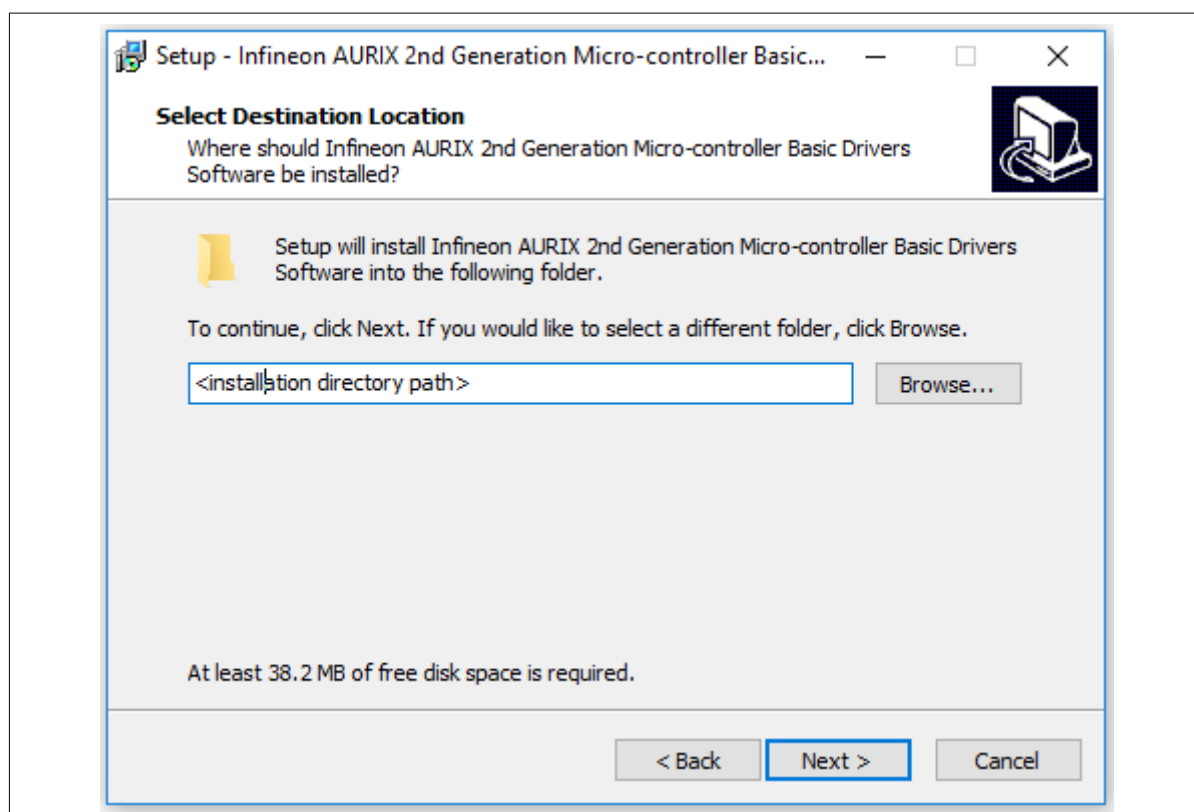


Figure 4 Select destination location

Generic information

7. Click **Browse** to navigate to a folder to install the drivers and the plug-in files.
8. Click **Next**.

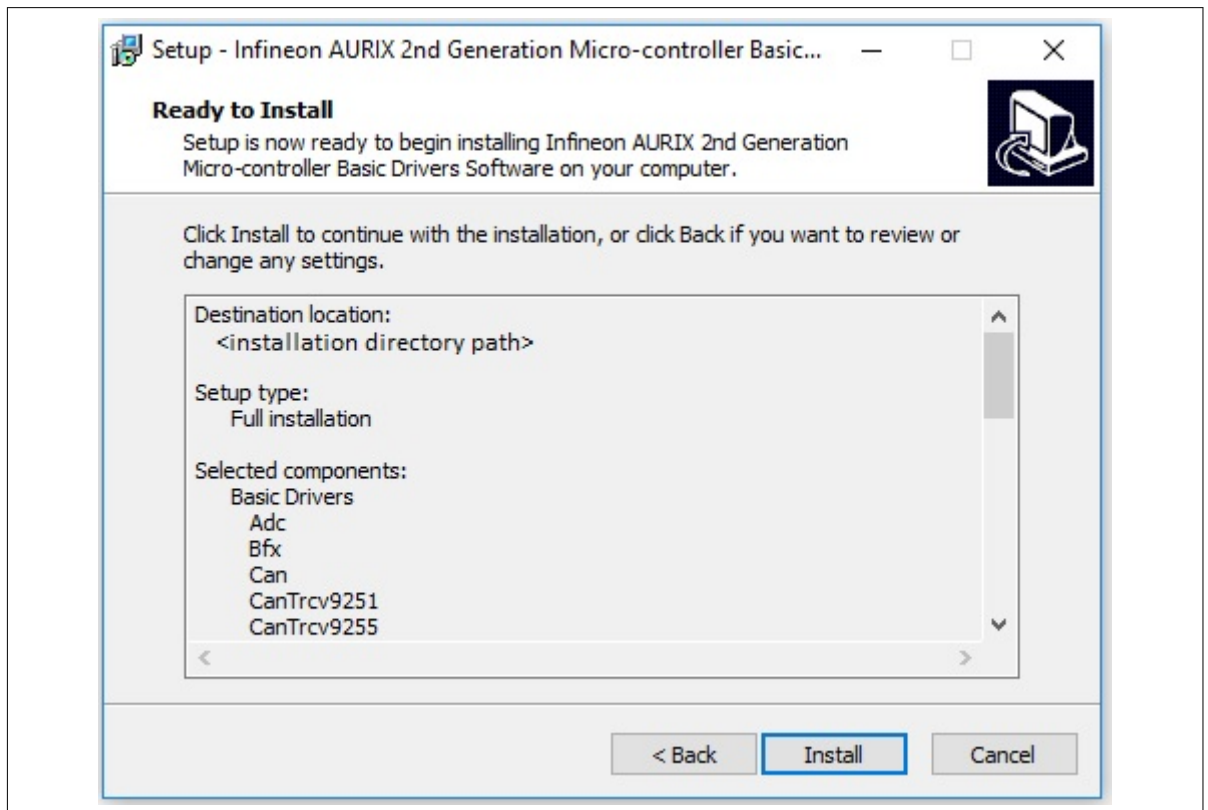


Figure 5 Ready to install

9. Click **Install**. This will initiate the installation process.
10. Copy all the plug-ins located at <Installed-Package>\McIsar\PluginsTresos\eclipse\plugins to the EB tresos™ plug-in path.

Note: Previous installation should be uninstalled before a new package is installed, unless you intend to overwrite the existing installation.

1.1.1.2 EB tresos™: license handling

The steps below must be followed to activate the EB tresos™ license:

1. Install the **EB_Client_License_Administrator** tool.
2. Open the <Install Path>\EB_Client_License_Administrator\bin\ClientLicenseAdministrator.exe.
3. Enter the activation code. Click **Activate**.

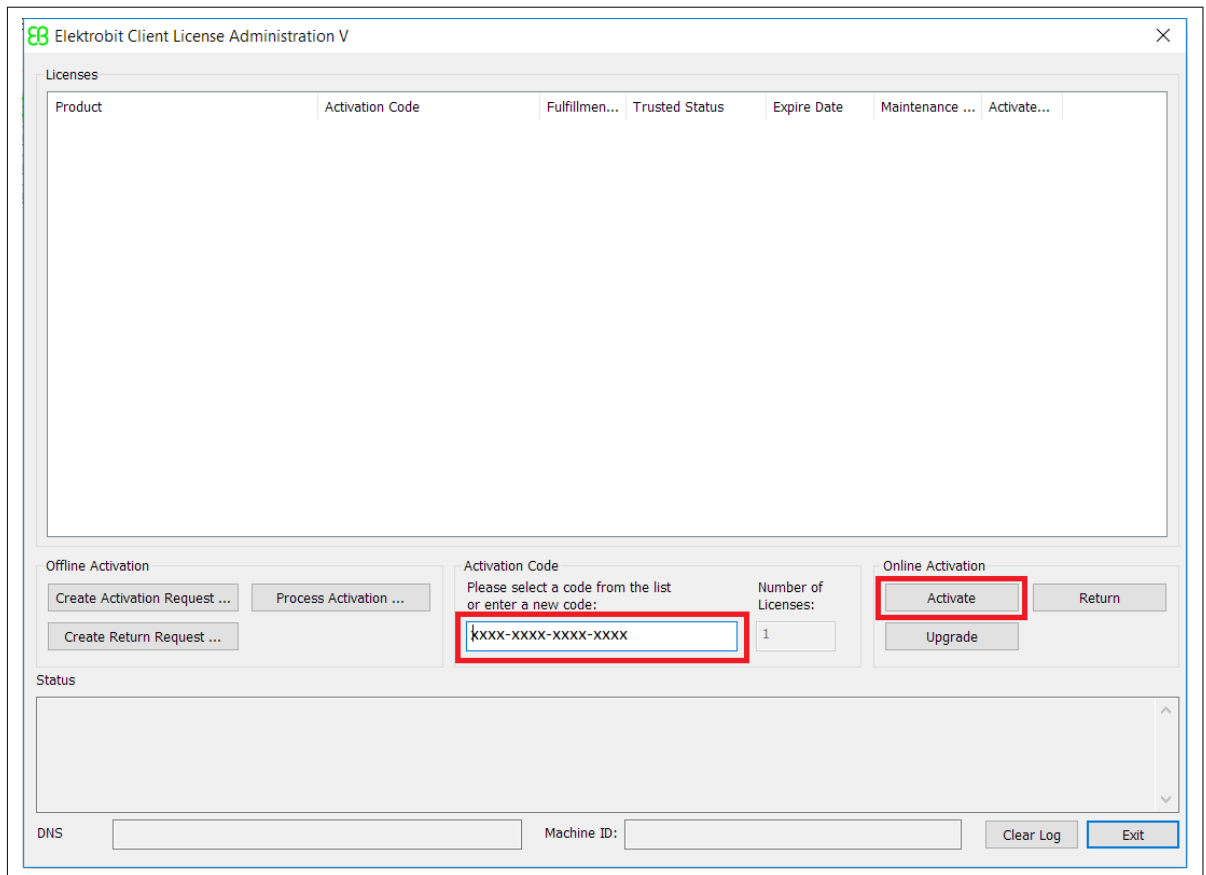


Figure 6 Tresos license activation

4. Status message **SUCCESSFULLY SEND ACTIVATION REQUEST** indicates the license has been activated successfully.

1.1.1.3 EB tresos: Eclipse handling

Due to a bug in the Eclipse tool, all directories named `org.eclipse.*` underneath the configuration directory of the EB tresos™ Studio installation have to be removed.

The `org.eclipse.*` directories contain cached information about installed plugins. Not deleting them can cause several undefined effects.

The new plugin will be available after restarting the EB tresos™ Studio.

If you have purchased EB tresos™ AutoCore together with EB tresos™ Studio, you can also integrate the plug-in with the user build environment. You will find the instructions on how to do this in the EB tresos™ AutoCore documentation: [EB tresos AutoCore documentation/EB tresos](#)

[AutoCore user's guide/Build environment](#).

1.1.1.4 Creating an EB tresos project

Prerequisites

- The EB tresos tool works only with the 64 bit version of Windows.
- The User must have a license/key for the EB tresos tool.

Note: Autosar version and tool version mentioned in this section is for the representation purpose only. The actual version shall be selected based on the package.

Note: User shall not delete any files present in the Tresos tools installed plugin directory.

The procedure for creating an EB tresos project is as follows:

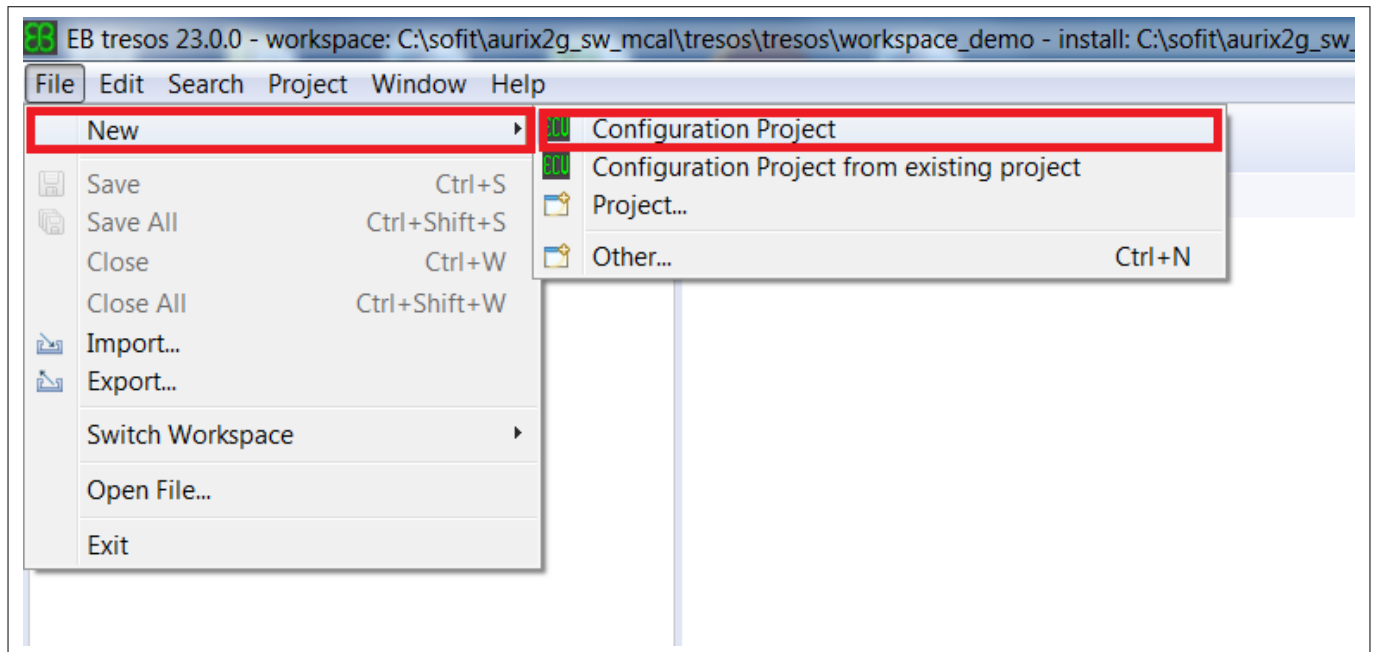


Figure 7 Configuration Project option

1. Click **File > New > Configuration Project**. The **New Project Wizard** will be displayed.

Generic information

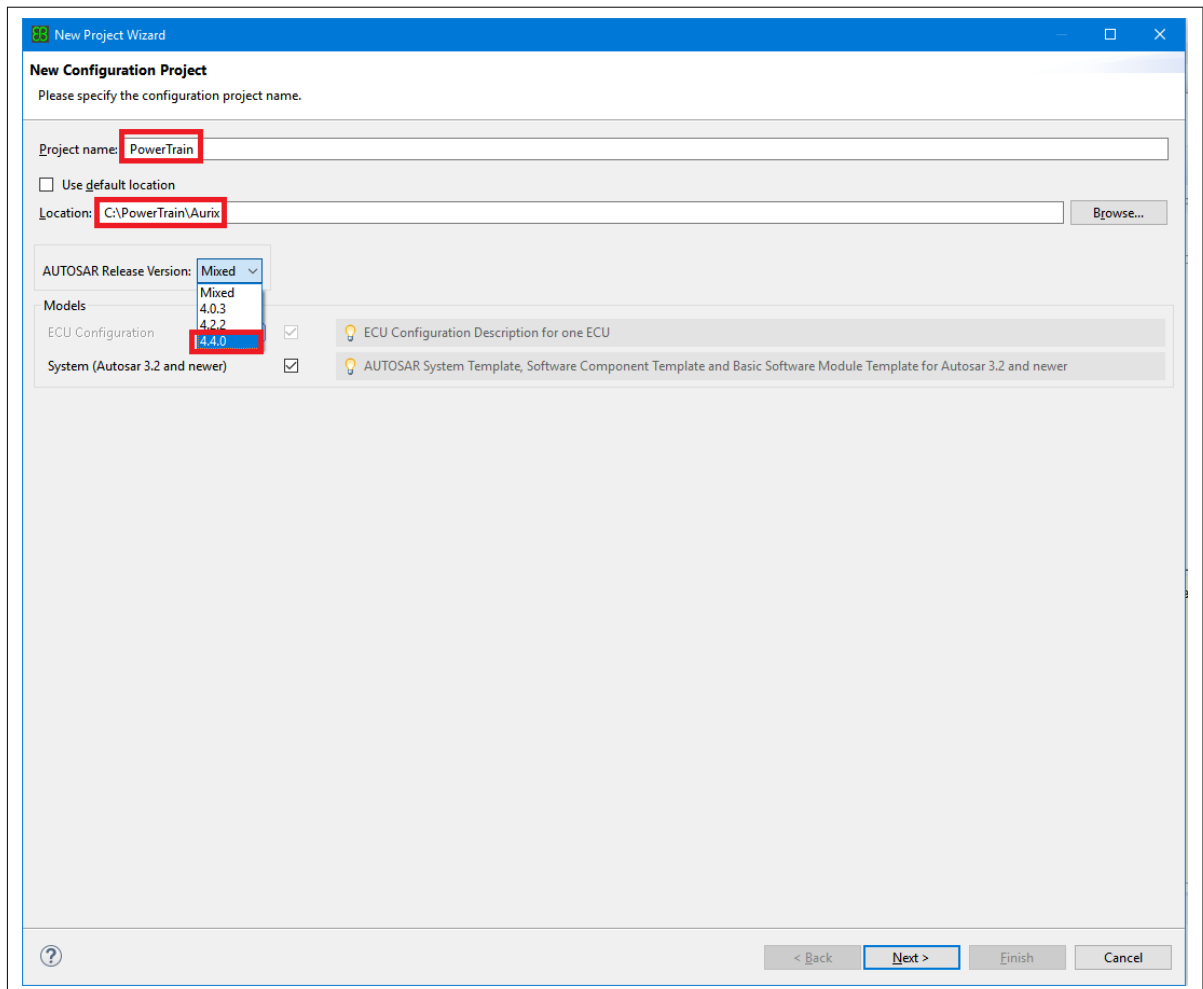


Figure 8 New Project Wizard

2. In the New Project Wizard, do the following:

- In the **Project name** field, specify the project name.
- The project that you create will map to a directory structure in the file system. The default file system location is displayed in the **Location** field. If you want to create the project and its contained resources in a different location, uncheck the **Use default location** field and browse to a location where you want to save the project.
- From the **AUTOSAR Release Version** drop-down list, select a version.
- Click the **Next** button.

Note: *Compiler_TS_TxDxM1I0R0 and Platforms_TS_T19D1M3I0R0 module plugins should be always placed in the tresos tool installed plugin location. Compiler_TS_TxDxM1I0R0 and Platforms_TS_T19D1M3I0R0 plugins are required for project creation in EB tresos studio, but Compiler_TS_TxDxM1I0R0 and Platforms_TS_T19D1M3I0R0 are not part of MCAL package delivery. By default Compiler_TS_TxDxM1I0R0 and Platforms_TS_T19D1M3I0R0 modules are part of tresos plugin installation.*

Note: *In case mixed mode option is selected during project creation, then by default added compiler module configuration can be removed from the project configuration.*

Generic information

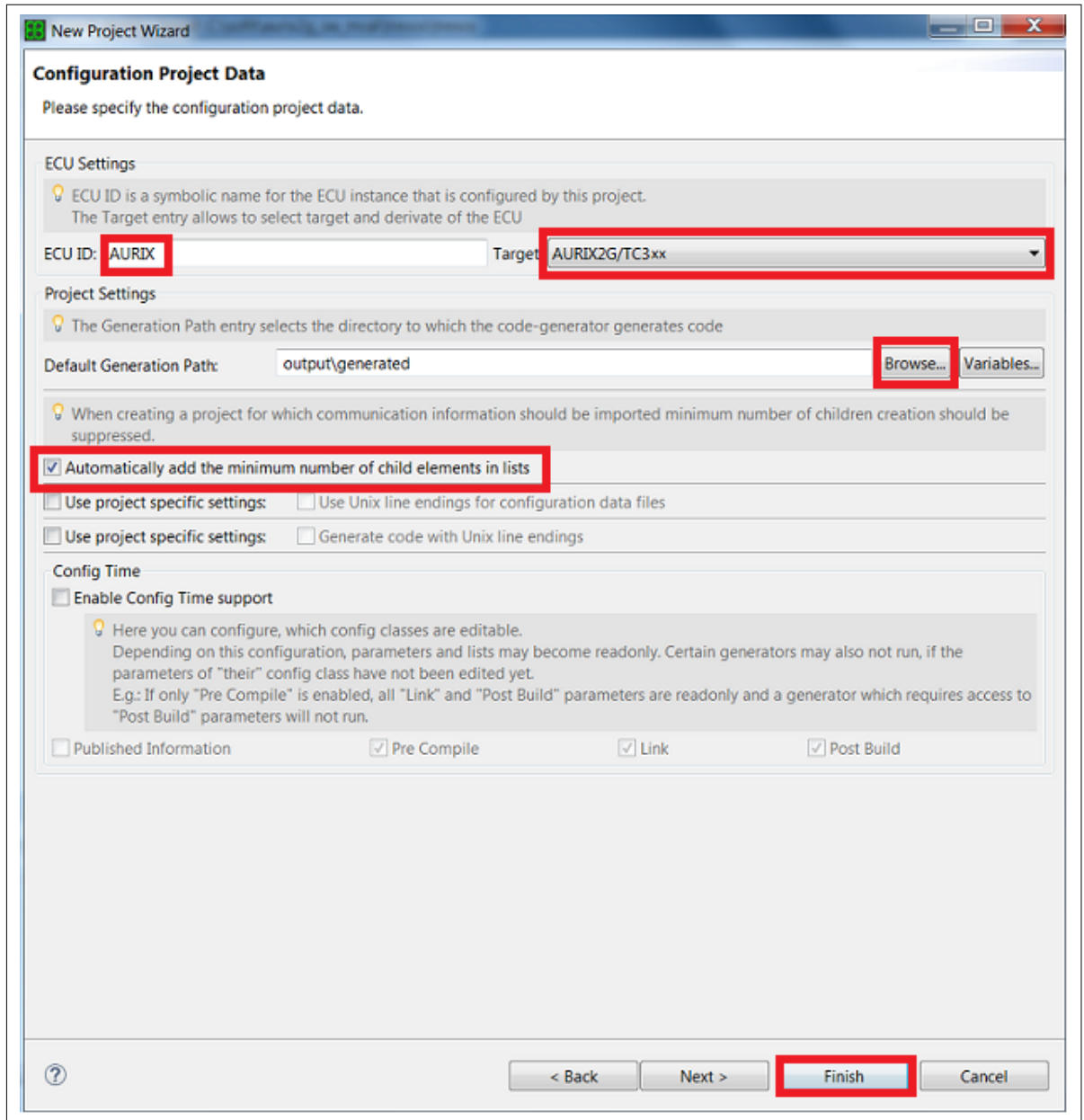


Figure 9 Configuration Project Data

3. Do the following:

- Specify a name for the ECU instance in the **ECU ID** field.
- From the **Target** drop-down list, select a target.
- In the **Default Generation Path** field, browse to a location in your computer where you want to store the output.
- Check the **Automatically add the minimum number of child elements in lists** option.
- Click the **Finish** button.

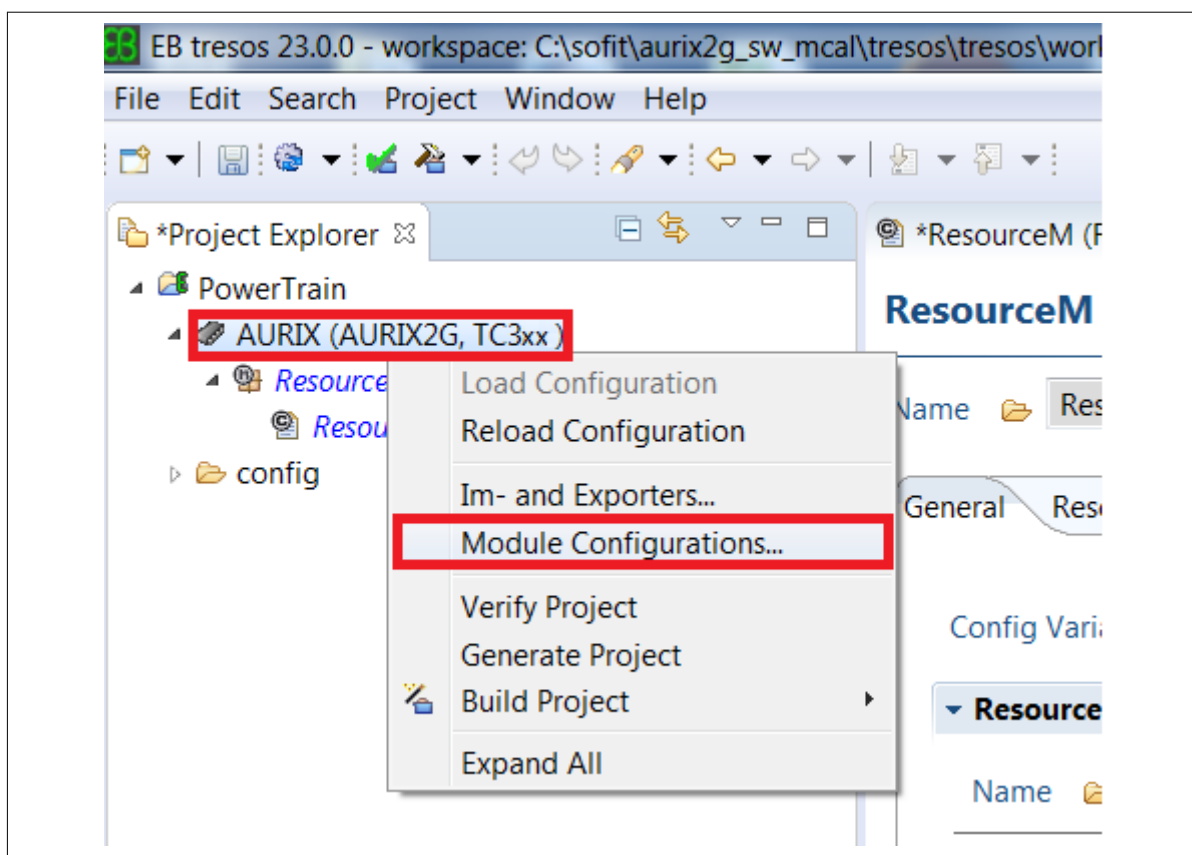


Figure 10 Module Configurations selection

4. Right-click the **ECU ID** and select **Module Configurations....**

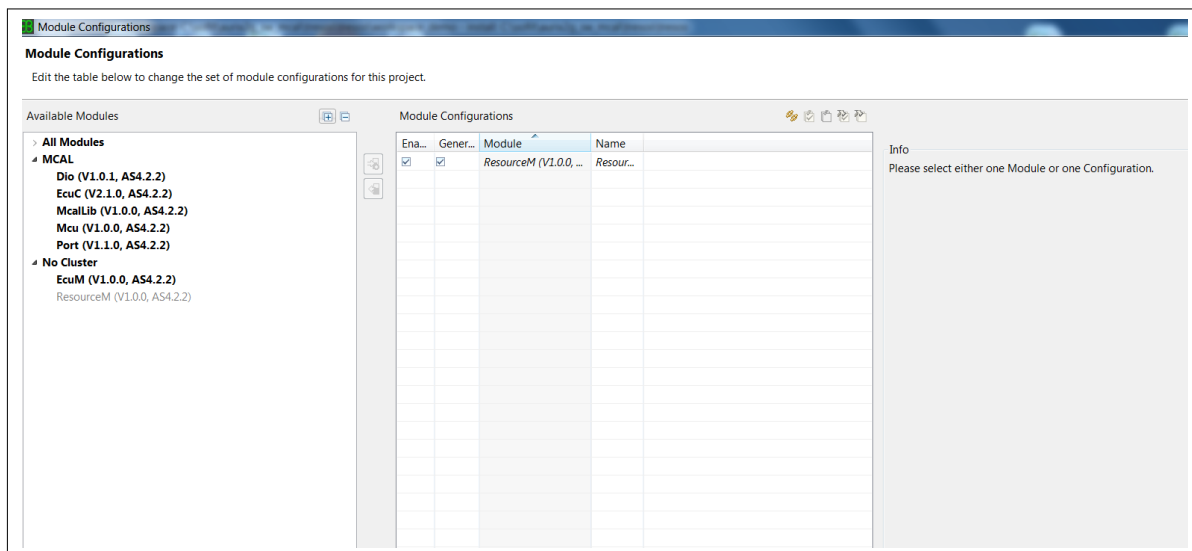


Figure 11 Module Configurations window

5. The **Module Configurations** window displays.

Generic information

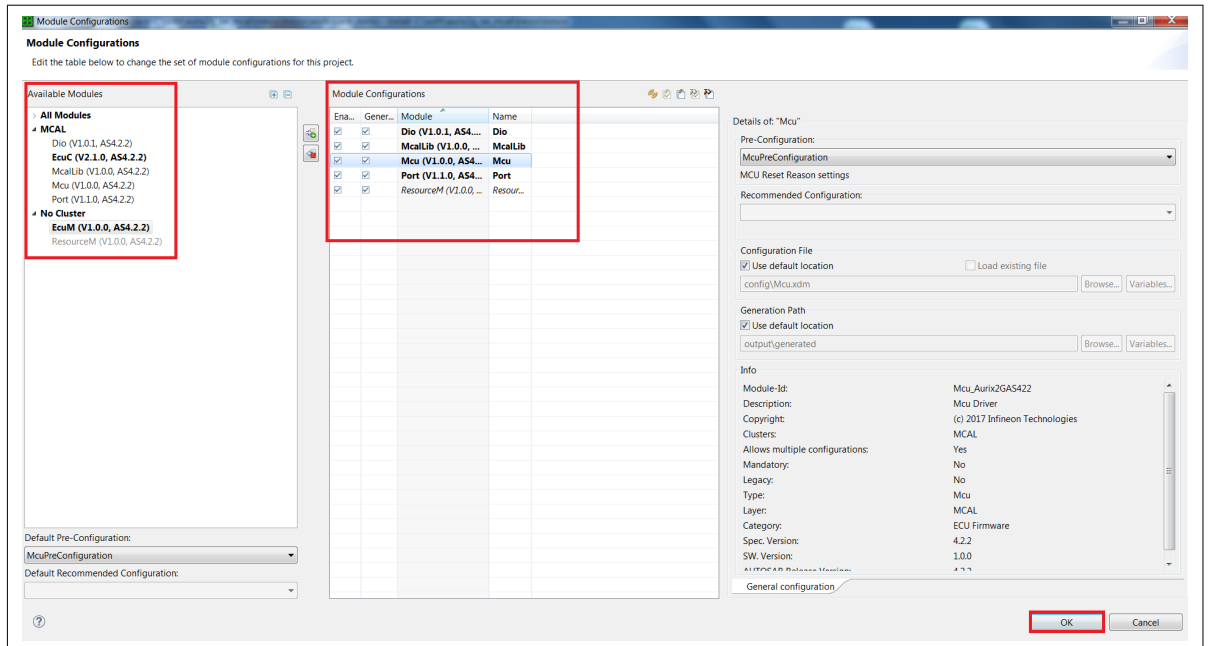


Figure 12 Modules populated

6. In the **Module Configurations** window, in the **Available Modules** section, double-click a module to move it to the **Module Configurations** section.
7. Click the **OK** button to finish the will be displayed.

1.1.2 Typical module parameter configuration

This section describes how a user may configure a typical parameter and generate the configuration files.

Note: *Autosar version and tool version mentioned in this section is for the representation purpose only. The actual version shall be selected based on the package.*

1.1.2.1 Configuring parameters

Once the EB tresos project is created, one needs to configure the module parameters.

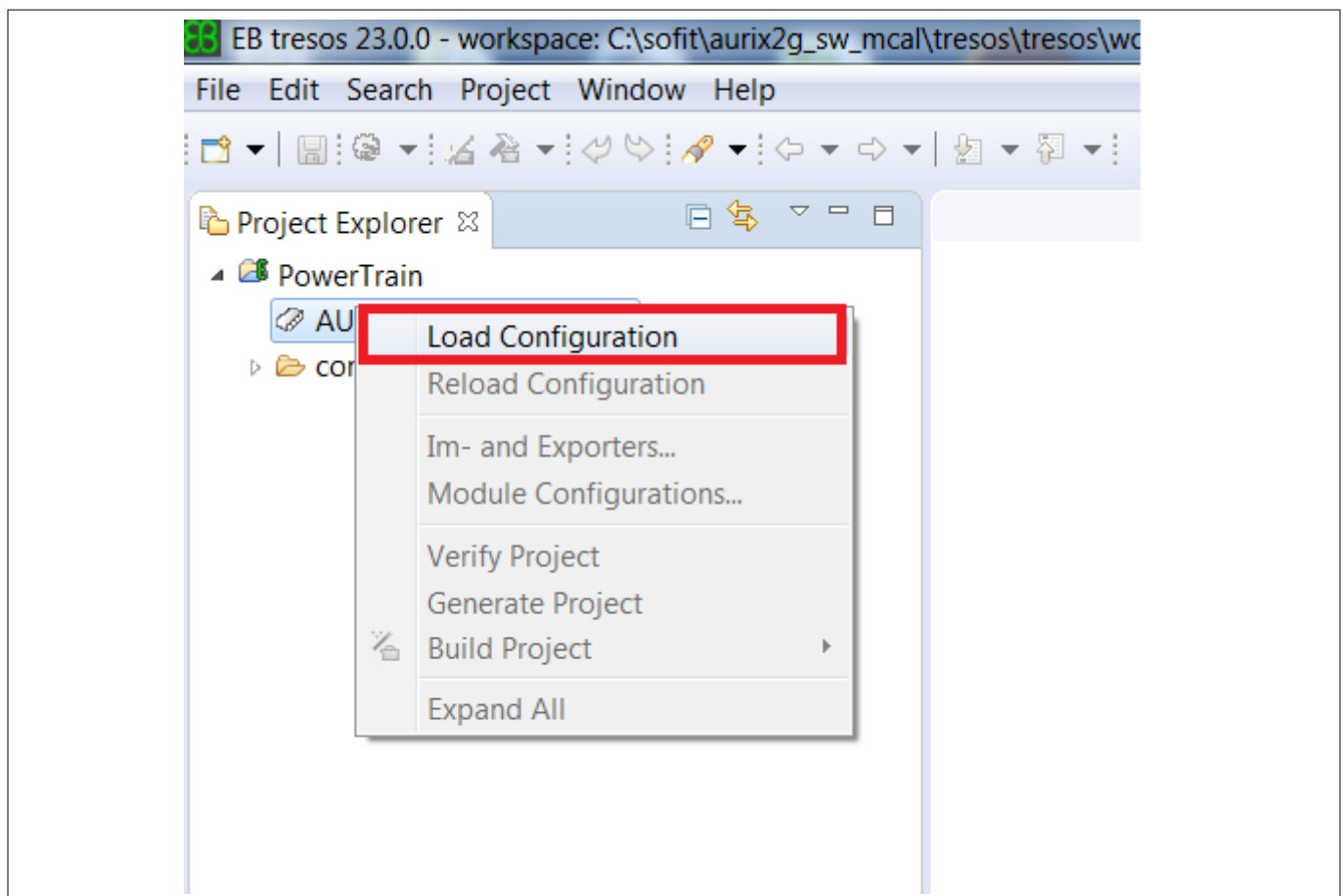


Figure 13 Loading configuration

1. Right-click the **ECU ID** and select the **Load Configuration** option. Alternatively, you can double-click the **ECU ID** to display the editor.

Generic information

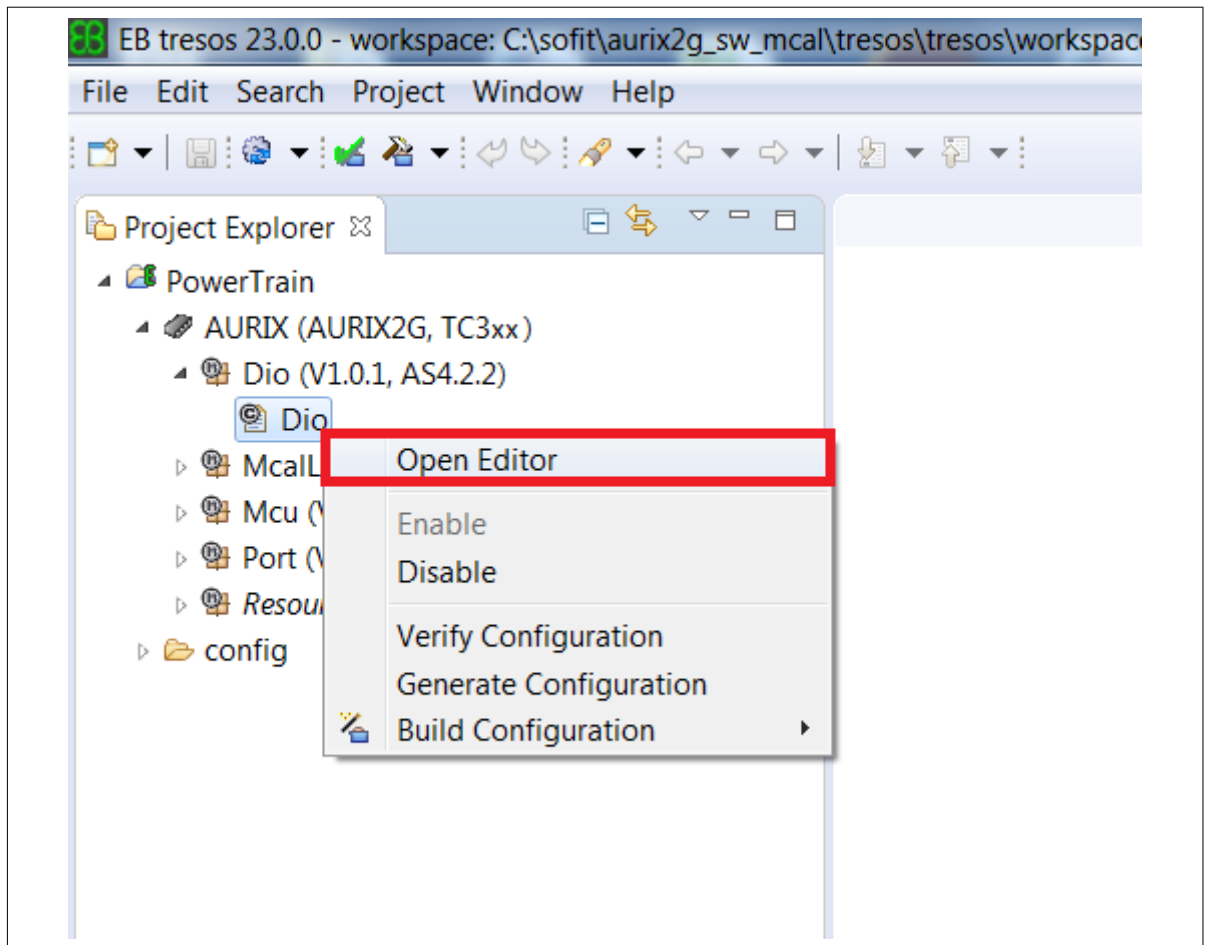


Figure 14 Open Editor option

2. After loading the configuration, right-click a module that you want to configure and select the **Open Editor** option. Alternatively, you can double-click the module to display the editor.

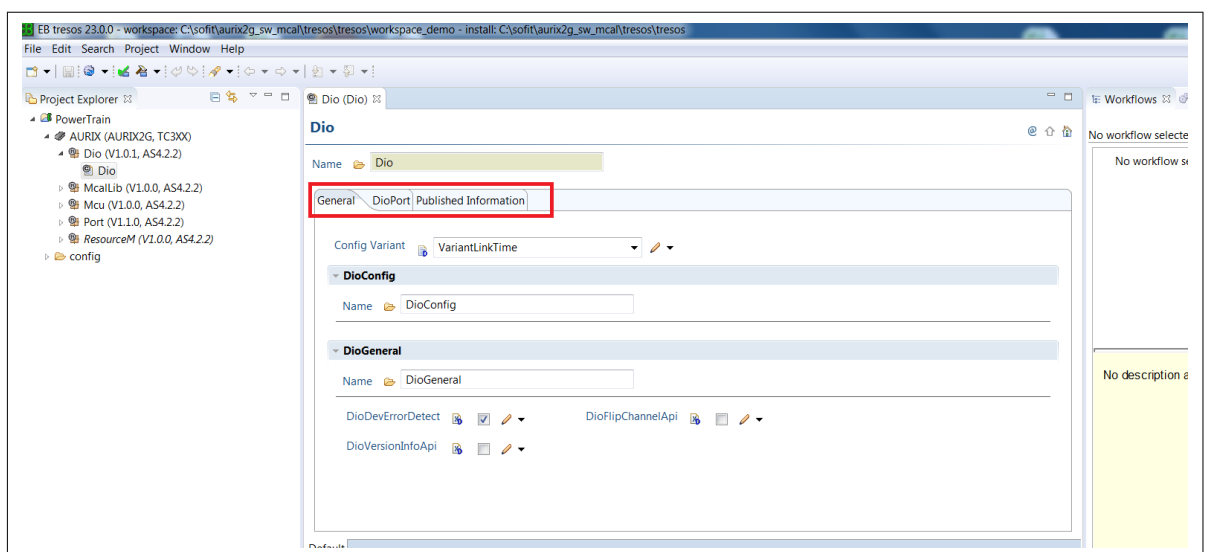


Figure 15 Parameter selection

3. In the editor section, click the respective parameter tab you want to configure.

Generic information

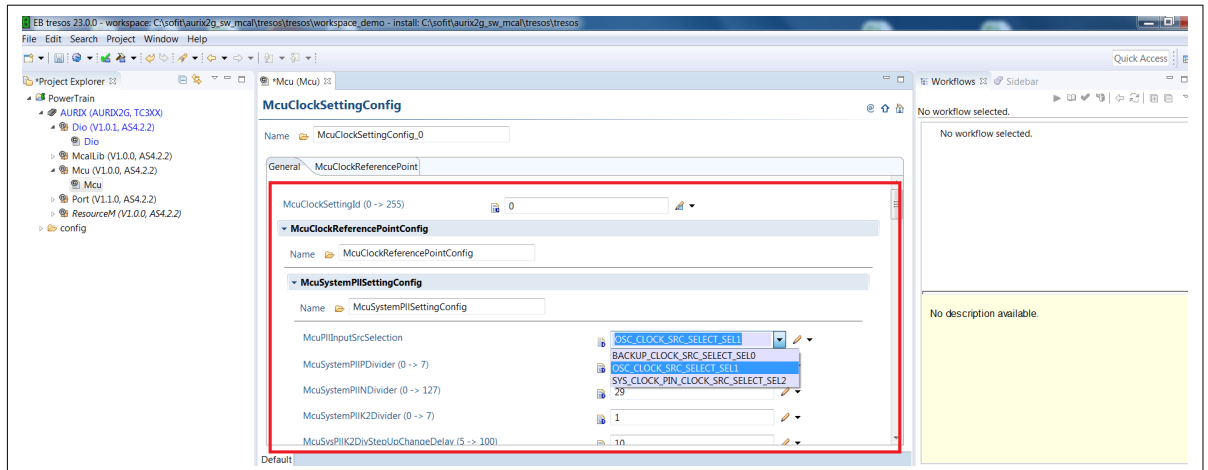


Figure 16 Configuring parameter

- Configure/edit the required parameters as applicable.

Note: The default value for all the configuration parameters are chosen for demo purpose only. The user shall configure each parameter as per the intended application requirement.

- Click the **Save** icon at the top left of the screen.

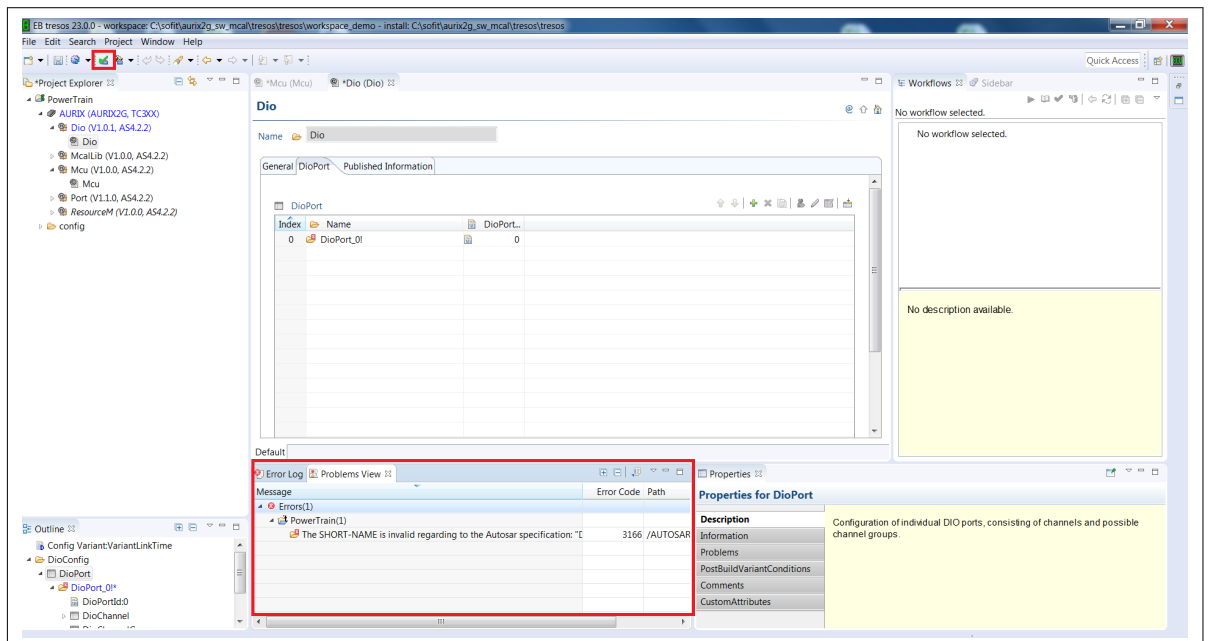
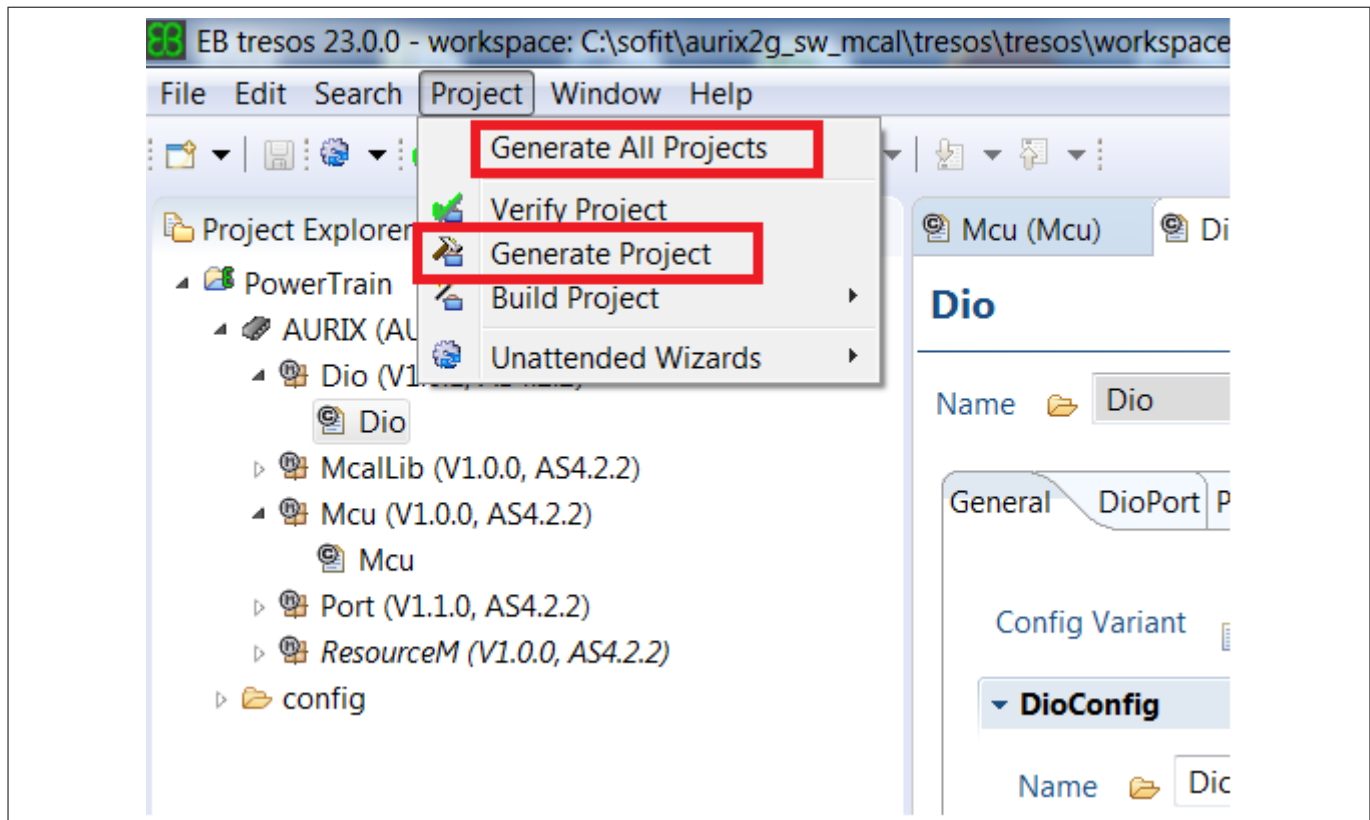


Figure 17 Parameter validation

- Click the **Verify** icon on the top of the screen. If there are any configuration errors, they are displayed at the bottom of the screen. Click the **Error Log** and **Problems View** tabs to view and resolve the error(s).

1.1.2.2 Generating configuration files

The steps to generate configuration files are as follows:

Generic information

Figure 18 Generating project

1. From the menu options, select **Project**.
2. Select either **Generate All Projects** or **Generate Project** option.
 - **Generate All Projects:** selecting this option will generate the configured XDM, <module>_PBcfg.c, <module>_PBcfg.h, <module>_Cfg.h and <module>_Bswmd.xml files for all the projects.
 - **Generate Project:** selecting this option will generate the configured XDM, <module>_PBcfg.c, <module>_PBcfg.h, <module>_Cfg.h and <module>_Bswmd.xml files for the selected project.

You can locate the generated files in the following location in your computer:

\tresos\tresos\workspace\PowerTrain\output\generated.

1.1.3 Building and linking to generate final application binary

This section describes how to build all source files and link them together to generate the ELF/HEX file(s).

This package leverages BIFACES, the standard build framework developed by Infineon. Any other package(s), supplied by Infineon, supporting BIFACES can be integrated with this package, if relevant.

Note: Autosar version and tool version mentioned in this section is for the representation purpose only. The actual version shall be selected based on the package.

Note: The platform software files are provided in MCAL package under the folder \Src\Infra_Prod\Platform. If the integrated system software uses the platform software files other than released by Infineon, the integrator must ensure that the all data types and macros are same as defined in the files delivered by Infineon.

Generic information

1.1.3.1 Building package

For the list of derivatives supported, refer the release notes provided with this package.

Note: Before proceeding with the following steps, to build a variant, make sure to copy all the plug-ins located at <Installed-Package>\Mclsrc\PluginsTresos\eclipse\plugins to the EB tresos plug-in path.

The steps to build a variant are as follows:

1. Open the command line shell.
 - To build <device>, go to C:\<package>\DemoWorkspace\McalDemo\<device>.
2. In the shell, invoke the DemoAppBuild.bat with different parameters. The user is provided with the following options.
 - Parameter 1 is compiler.
 - The compiler can be *TASKING*, *GNU* or *DCC* or *GHS*. If no parameter is passed, *TASKING* is selected by default.
 - Parameter 2 is the option to build with or without TRESOS generation.
 - User can pass *WITH_TRESOS* or *WITHOUT_TRESOS* or *WITHOUT_TRESOS_WITHOUT_FR*. If no parameter is passed, the build system takes *WITH_TRESOS* as the default parameter.
 - Parameter 3 is the option to set the TRESOS path.
 - User has to pass the full TRESOS path (using Unix slashes) including the Drive. If no parameter is passed, the build system takes
 - C:/sofit/aurix2g_sw_mcal/tresos/<tresos-version>/tresos/tresos/bin as the default path.
 - Parameter 4 is the option to set the COMPILER path.
 - User has to pass the full COMPILER path including the Drive. If no parameter is passed, the build system takes the default COMPILER path (that is, *TASKING*, *GNU*, *GHS* or *DCC*) from SOFIT.

Example commands to build DemoApp

- Double-click the DemoAppBuild.bat file.
 - This will build for the TASKING compiler and will generate the TRESOS code as well.
- To build for the GNU/DCC/GHS/TASKING compiler without generating TRESOS code:
 - DemoAppBuild.bat GNU/DCC/GHS/TASKING WITHOUT_TRESOS

Note: This example tries to modify only first and second parameters. Hence, the third and fourth parameters are not required to pass.

- To build for the WindRiver compiler with the TRESOS code generated and with different TRESOS and Compiler paths:
 - DemoAppBuild.bat DCC WITH_TRESOS C:/EB/Tresos/bin
C:/WindRiver/Compiler/diab-5.9.7.0/WIN32

Note: Even if only fourth parameter to be modified, the user has to pass first three parameters compulsorily.

1.1.3.2 Release folder structure

The following diagram depicts the organization of the release folders along with Makefile distribution.

Note: Similar folder structure will be created for all the supported derivatives.

Generic information

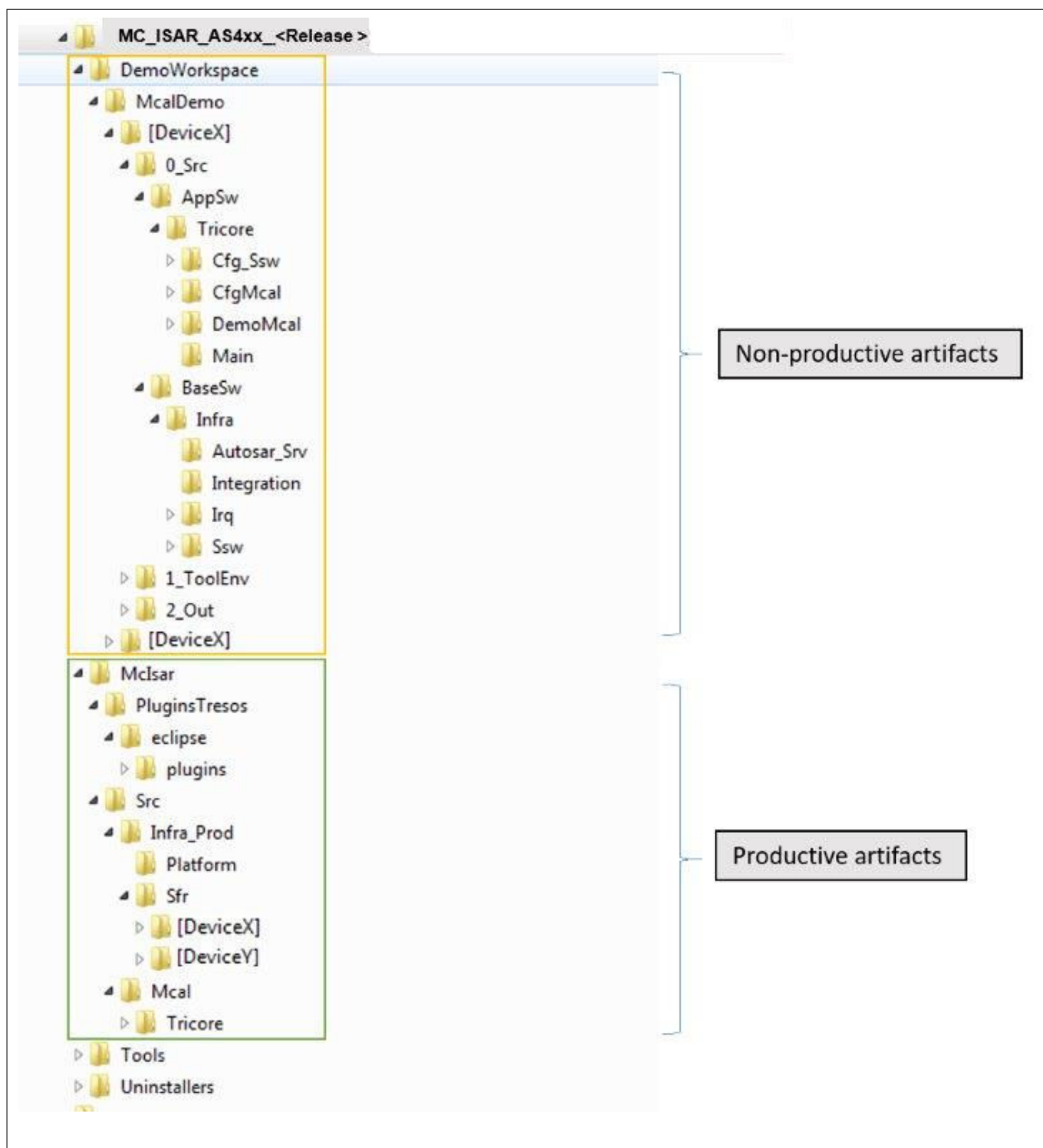


Figure 19 Release folder structure

Note: The **2_out** folder is generated after compilation.

Note: The availability of the modules for the released product must be referred from the Release Notes.

Generic information

Note: *The module specific Irq files provided in the folder '`\DemoWorkspace\McalDemo\<device>\0_Src\AppSw\Tricore\DemoMcal\Demo_Irq`' are non-productive source files and these files contains references to test code used by Infineon. The integrator shall take care if any of the code in respective Irq files are not applicable for the end application.*

1.1.3.3 Makefile adaptations

To make the process easier, Infineon have developed a wrapper over the Makefile in the form of a batch file. When this batch file is run, it automatically modifies the Makefile.

The following Makefiles are updated:

- `ConfigPrjExtn.mk`
- `Tresos.mk`

To modify Makefile(s) to suit the user environment, one needs to edit Makefile(s) manually.

The procedure to manually edit Makefile is as follows:

1. Open the `Tresos.mk` file (located at `<package>\DemoWorkspace\McalDemo\<derivative>\1_ToolEnv\0_Build\1_Config\Extensions`) in a text editor. By default, the DemoApp basic build is selected. The `Tresos.mk` file allows to configure the following parameter(s):
 - EB tresos bin path
 - EB tresos command
 - DemoApp configuration path

Note: *By default, all modules are selected for build.*
2. The XML files are placed in the `<package/Tools>`. Here, there are several files based on the compiler/derivative/FR_rebuild. These files can be modified for:
 - additional external paths for extra code to be built
 - discard file patterns
3. In the `Config_<compiler>.mk` file (located at `<package>\DemoWorkspace\McalDemo\<derivative>\1_ToolEnv\0_Build\1_Config\Config_Tricore_<compiler>`), edit the following parameter(s):
 - Compiler path
 - Compiler options
 - Assembler options
 - User includes

Editing batch file(s)

The procedure to manually edit a batch file to adapt to user environment is as follows. However, Infineon recommends using only the batch file (located at `<package>\DemoWorkspace\McalDemo\<derivative>`) instead of editing the Makefiles.

1. Open the `DemoAppBuild.bat` file in a text editor. The user is allowed to build the code for the supported compiler(s).
2. The following parameter(s) can be edited in the `DemoAppBuild.bat` file.
 - Parameter 1: compiler (for example, GNU)
 - Parameter 2: can have three possible combinations as follows:
 - with Tresos generation for both FR and Normal builds (this option is selected by default)
 - Values: empty, WITH_TRESOS

Generic information

- without Tresos generation for both FR and Normal build
 - Value: WITHOUT_TRESOS
- without Tresos generation for Normal build and no FR build
 - Value: WITHOUT_TRESOS_WITHOUT_FR
- Parameter 3: EB tresos bin path
- Parameter 4: Compiler path

1.1.3.4 Linker script adaptations

Linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file. The linker program uses a linker script. The main purpose of the linker script is to describe how the sections in the input files should be mapped into the output file, and to control the memory layout of the output file.

The following linker scripts are provided by Infineon for the linking purpose:

- Lcf_Tasking_Tricore.lsl for the Tasking compiler
- Lcf_Gnuc_Tricore.lsl for the GNU compiler
- Lcf_Dcc_Tricore.lsl for the WindRiver compiler
- Lcf_Ghs_Tricore.lsl for the GreenHills compiler

Each linker script can be modified by the user as per their environment requirements.

MCAL modules use <Module>_MemMap.h file for defining compiler pragma, which controls the output sections such as placing the code and global variables. It is entirely up to the integrator to define a linking scheme that places the output sections at desired addresses in the processor's memory map.

Note: *The linker script provided is compiler dependent.*

For example, the ADC driver ensures that global variables and text segments are allocated to appropriate output sections.

Generic information

A sample for using compiler pragma for a global variable from `Adc.c` is as follows:

```
#define ADC_START_SEC_VAR_CLEARED_ASIL_B_CORE2_UNSPECIFIED
#include "Adc_MemMap.h"
    static Adc_GlobalDataType Adc_KernelData_Core2[ADC_KERNEL_USED_COUNT_CORE2];
#define ADC_STOP_SEC_VAR_CLEARED_ASIL_B_CORE2_UNSPECIFIED
#include "Adc_MemMap.h"
```

The user may define the compiler pragmas in `Adc_MemMap.h` as below:

```
#elif defined ADC_START_SEC_VAR_CLEARED_ASIL_B_CORE2_UNSPECIFIED
    #ifdef _TASKING_C_TRICORE_
        #pragma section_name_with_symbol
        #pragma section all "ClearedData_Cpu2.Unspecified"
        #pragma align 4
    #elif defined _GNU_C_TRICORE_
        #pragma section "ClearedData_Cpu2.Unspecified" aw 4
    #endif
    #undef ADC_START_SEC_VAR_CLEARED_ASIL_B_CORE2_UNSPECIFIED
    #undef MEMMAP_ERROR
#elif defined ADC_STOP_SEC_VAR_CLEARED_ASIL_B_CORE2_UNSPECIFIED
    #ifdef _TASKING_C_TRICORE_
        #pragma align restore
        #pragma section all
    #elif defined _GNU_C_TRICORE_
        #pragma section
    #endif
    #undef ADC_STOP_SEC_VAR_CLEARED_ASIL_B_CORE2_UNSPECIFIED
    #undef MEMMAP_ERROR
```

1.1.3.5 Initiation of compiling and linking

The DemoApp build is launched by invoking `DemoAppBuild.bat` present in the <Installed-
Package>\DemoWorkspace\McalDemo folder. The application (DemoApp files in this case), source
and configuration files are compiled and linked together. The output files, *.elf/*.hex, are generated after
completion of the build process in the <Installed-

Package>\DemoWorkspace\McalDemo\2_Out\<Tricore_Tasking\Tricore_Gnuc\Tricore_Dcc\Tricore_Ghs> folder.

The output `DemoApp_Node0.elf/DemoApp_Node0.hex` and `DemoApp_Node1.elf/DemoApp_Node1.hex` files are generated
if the FR module is selected for installation. The `DemoApp_Node0.elf/DemoApp_Node0.hex` files are used for
executing the DemoApp of all modules other than FR. The `DemoApp_Node1.elf/DemoApp_Node1.hex` files are only
used for executing the FR DemoApp. The output `DemoApp.elf/DemoApp.hex` files are generated if the FR module is
not selected for installation. The output files are generated in the <Installed-

Package>\DemoWorkspace\McalDemo\2_Out\<Tricore_Tasking\Tricore_Gnuc\Tricore_Dcc\Tricore_Ghs> folder.

Note: *All packages (BASIC, COM-E, CD, DEMO) with all provided drivers should be installed in the same
directory in order to build the DemoApp provided for MCAL drivers. Though it is possible to install
individual packages in a separate directory for use but the DemoApp cannot be built individually/
selectively.*

Generic information

Note: In case of selective installation (for example, if only ETH is to be installed from the COM-E package), remove the modules and their respective .xdm files from the EB tresos workspace and also remove the driver source code, demo code, IRQ and integration files.

Command to merge multiple configured XDM to single EPC file

Following is an example of merging XDM file to generate an EPC file:

```
C:/sofit/aurix2g_sw_mcal/tresos/tresos/bin/tresos_cmd.bat -DMapOptionalAsList=false
-Dtarget=AURIX2G -Dderivate=<device> legacy convert Port.xdm
Pwm_17_Gtm.xdm Spi.xdm Adc.xdm Dem.xdm Dio.xdm EcuM.xdm Gpt.xdm Icu_17_TimerIp.xdm
Irq.xdm Mcu.xdm ResourceM.xdm Combined.epc@asc:4.2.2
```

Note: -Dderivate should be changed as per the desired derivate to run the DemoApp.

Command to generate code from EPC file

Following is an example command to generate code from the EPC file:

```
C:/sofit/aurix2g_sw_mcal/tresos/tresos/bin/tresos_cmd.bat -Dtarget=AURIX2G
-Dderivate=<device> legacy generate Combined.epc@asc:4.2.2 -o
outputdir -g Irq_Aurix2GAS422 -g Mcu_Aurix2GAS422 -g Port_Aurix2GAS422 -g
Dem_Aurix2GAS422 -g EcuM_Aurix2GAS422 -g Adc_Aurix2GAS422 -g Dio_Aurix2GAS422 -g
Gpt_Aurix2GAS422 -g Icu_Aurix2GAS422 -g Pwm_Aurix2GAS422 -g Spi_Aurix2GAS422 -g
ResourceM_Aurix2GAS422
```

Utilities for building DemoApp

The following utilities are used for the build process:

Description	Location
The Makefile used for building the application contains rules for generating the executable	The file is located at <Installed-Package>\DemoWorkspace\<Device>\McalDemo\1_ToolEnv\0_Build

Note: Any error in compilation and linking displays in the DOS prompt.

Generic information
1.1.3.6 Flashing the image

Image is flashed using the UDE Memtool. The steps for flashing image using the UDE Memtool are as follows:

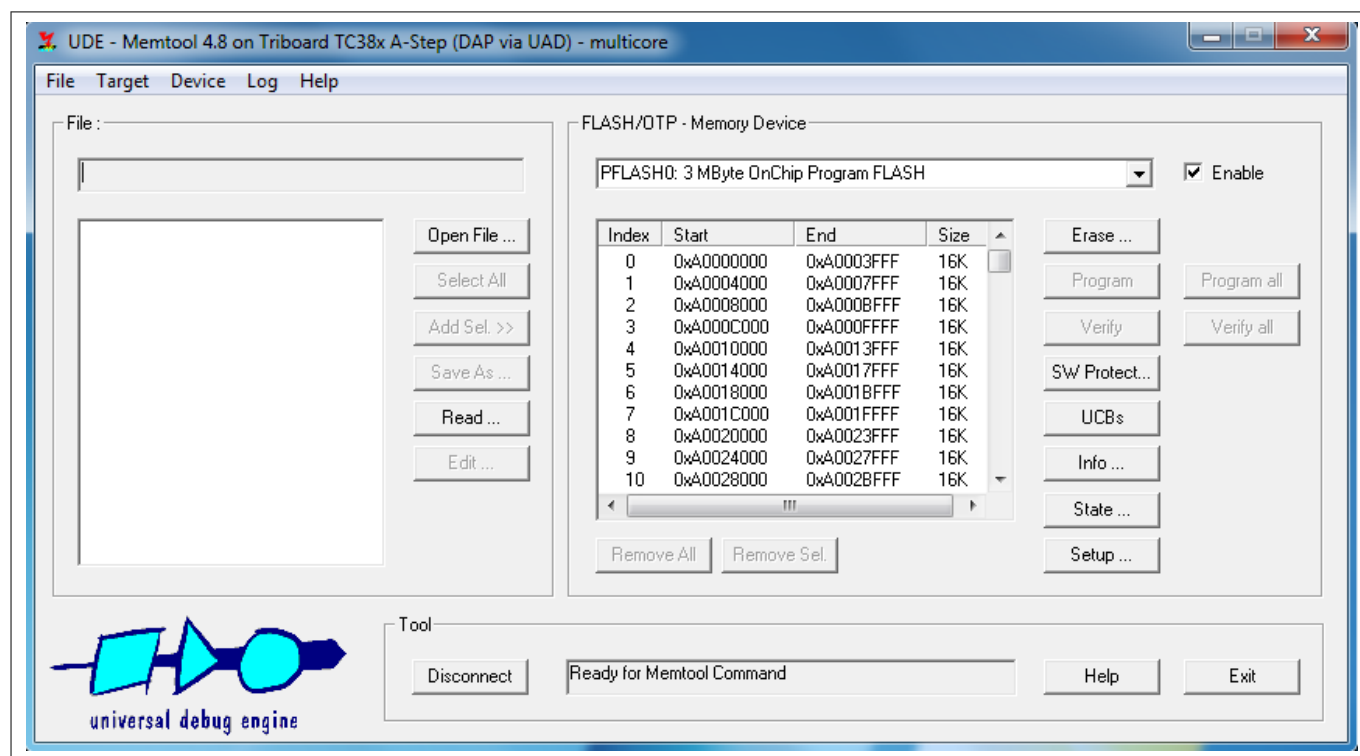


Figure 20 UDE Memtool interface

1. Double-click the Memtool1.exe file to open the UDE Memtool.

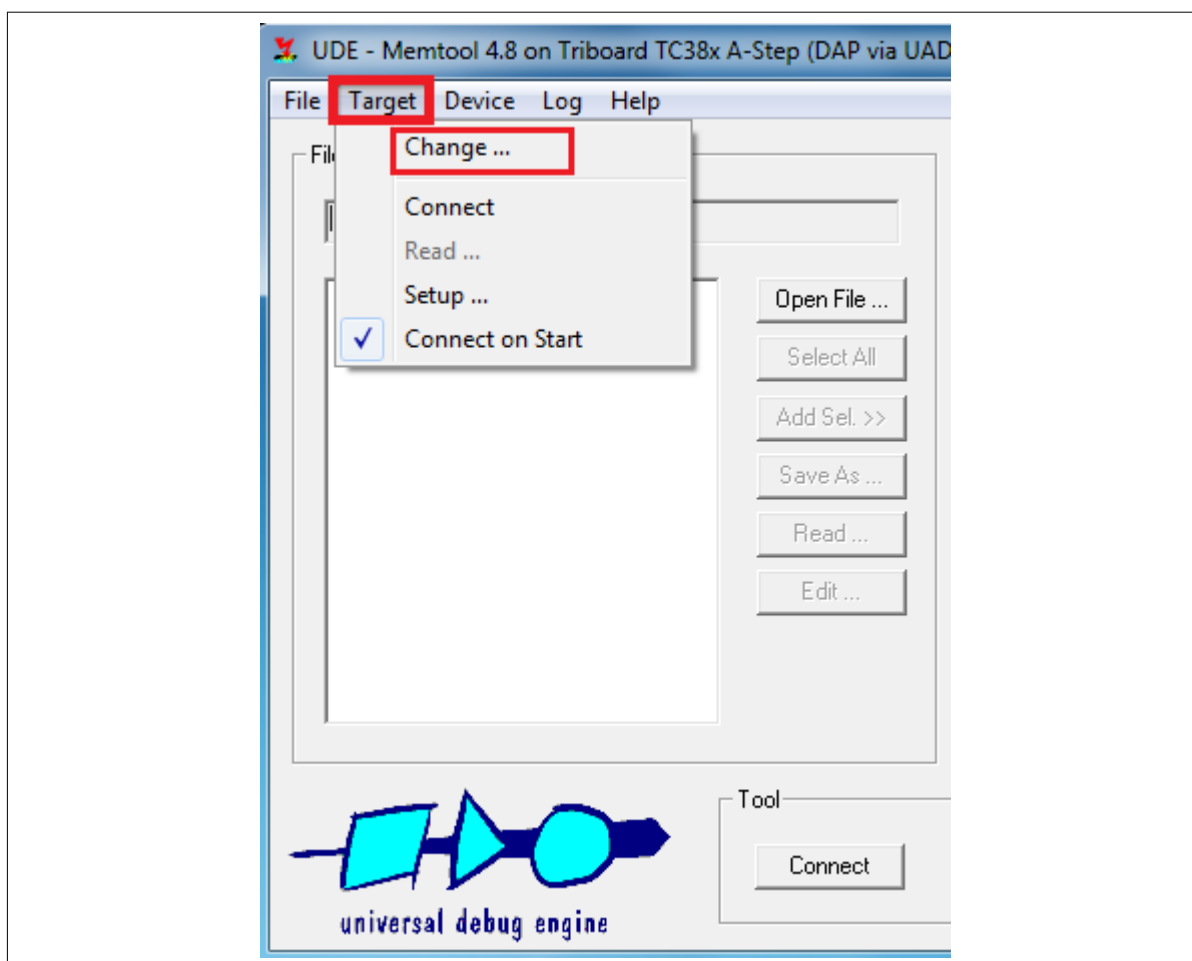


Figure 21 **Configuring target file**

2. Click **Target > Change** to configure the target file for the respective device.

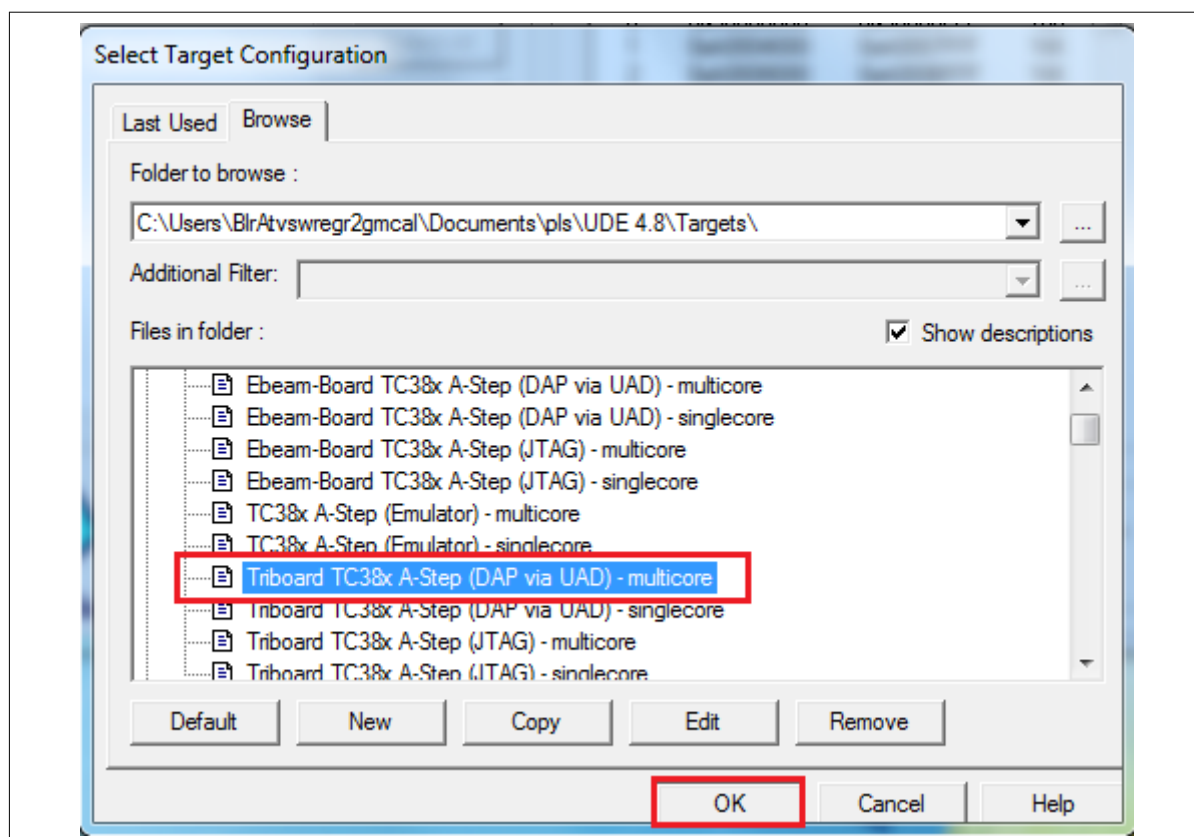


Figure 22 **Selecting target configuration**

3. Select a configuration file and click the **OK** button.

Generic information

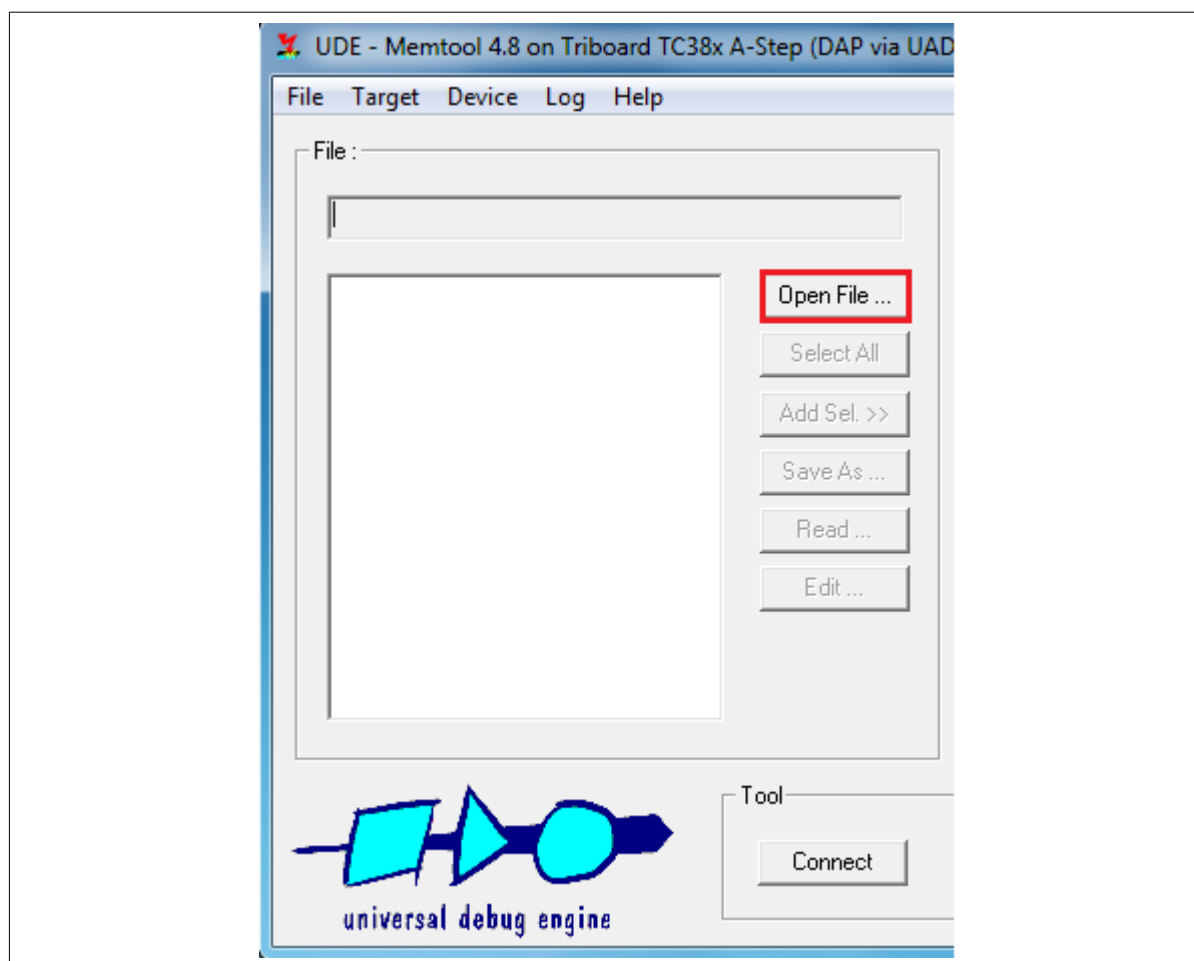


Figure 23 Opening a file

4. Click the **Open File** button.

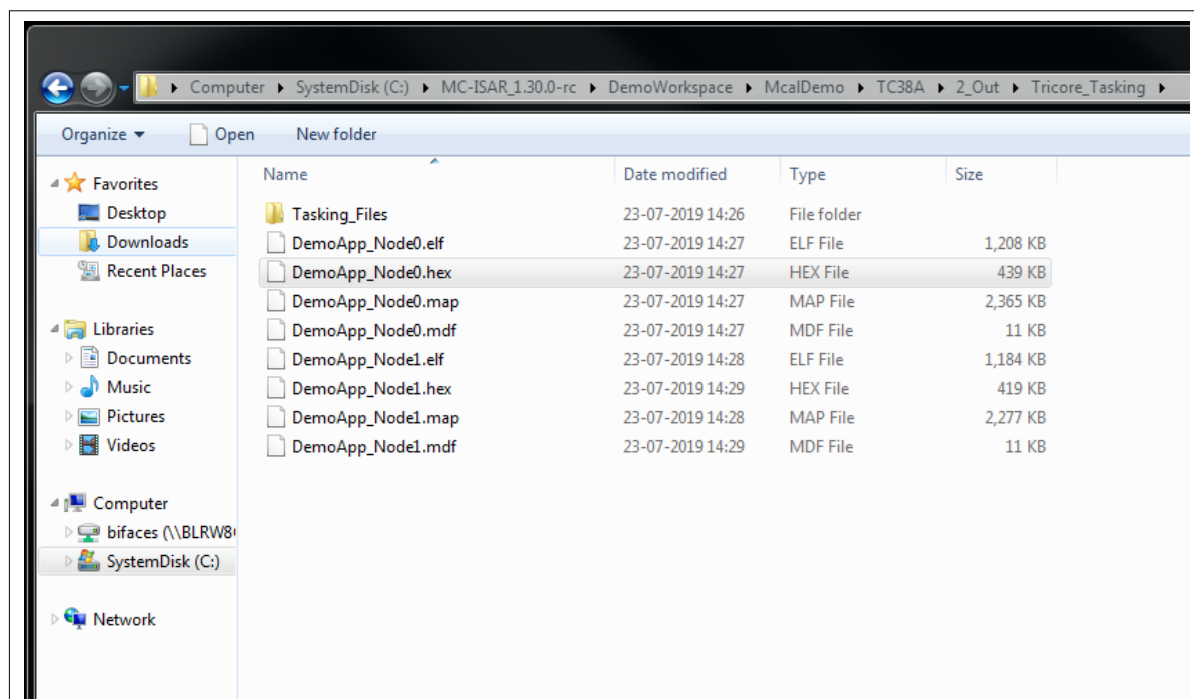


Figure 24 Selecting hex file

5. Select the hex file.

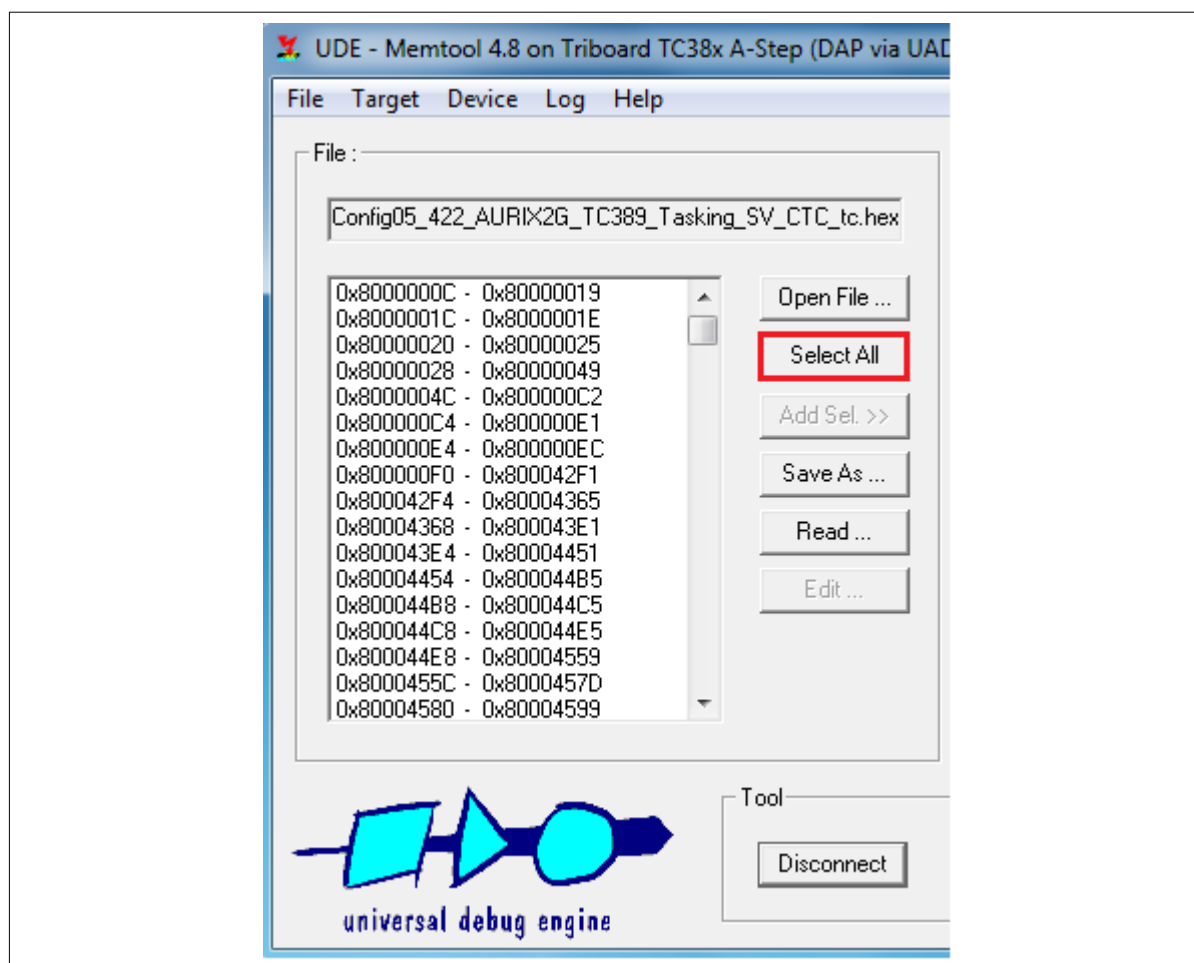
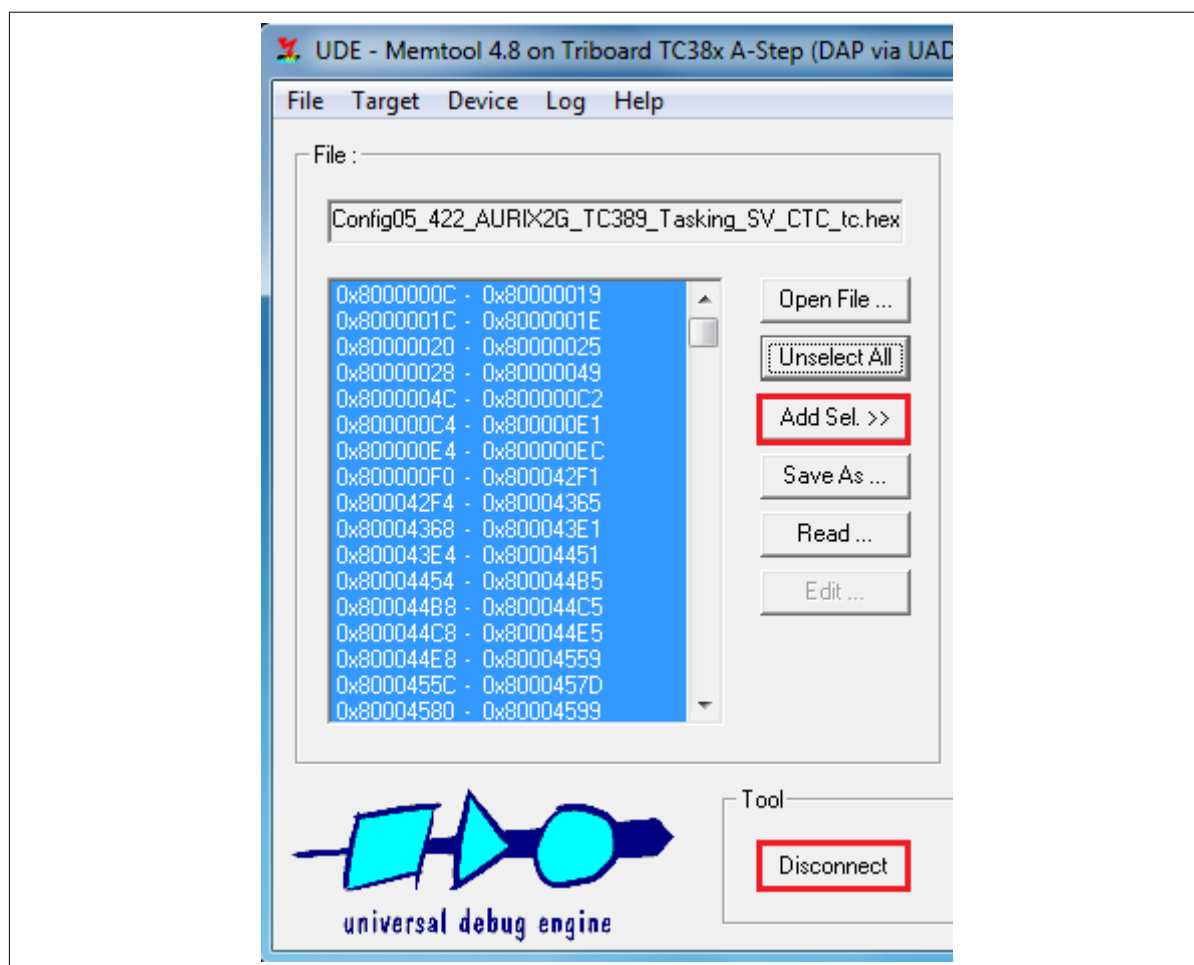


Figure 25 **Selecting flashing sectors**

6. Click the **Select All** button to select flashing all sectors.

**Figure 26** Flashing memory

7. Click the **Connect** button. Once connected successfully, the button turns to **Disconnect**.
8. Click the **Add Sel.** button to flash the memory.

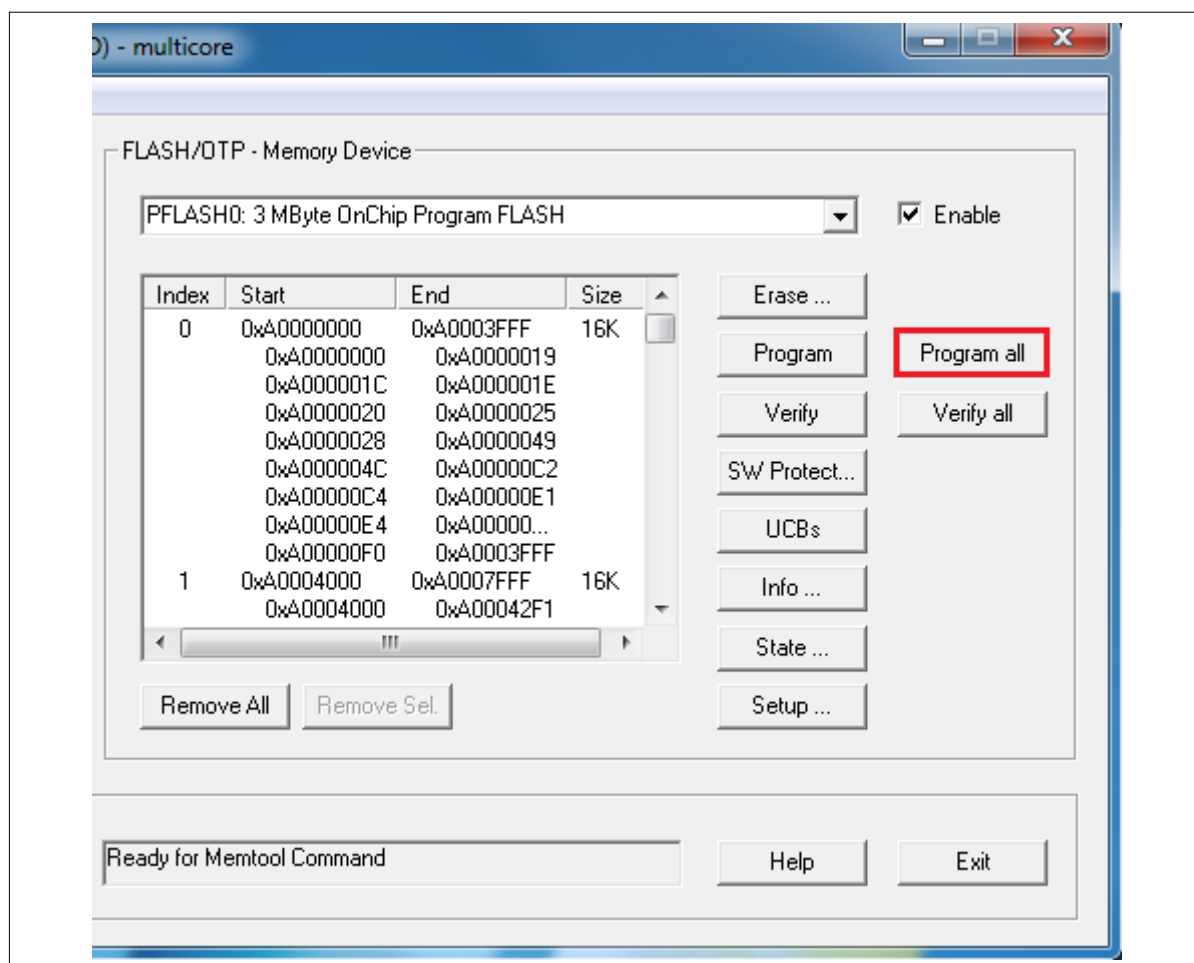


Figure 27 Selecting program all option

9. Click the **Program all** button to flash the hex image.

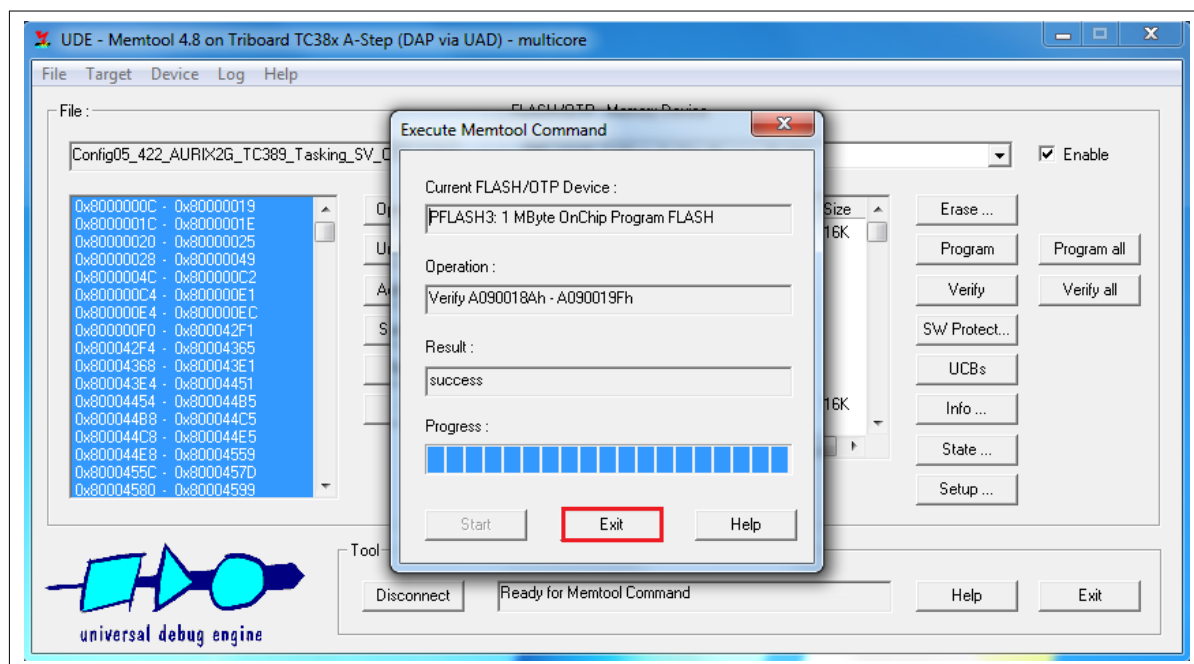


Figure 28 Executing command

10. The selected image is flashed. Click the **Exit** button to finish the process.

Generic information

1.2 MCAL Initialization

In general, a module's initialization function will explicitly configure all SFRs required for achieving the intended functionality. However, If the SFRs are modified by user SW prior to module's initialization, then the following points shall be considered during integration :

- If the SFRs modified by user software can impact the MCAL intended functionality and the modified SFR is not initialized in module's initialization function, then the application software should restore the modified SFRs to the hardware reset values before invoking the module's initialization. Following are some examples for such scenarios.

Example 1: MCAL does not use Dead Time Module(DTM) feature in GTM. If the application software enables the DTM feature, the PWM channels allocated to MCAL PWM driver output may get re-directed to the DTM module for dead-time generation and impact the desired PWM output signal. In this case, user software should restore SFRs to reset value which are modified to enable the DTM feature.

Example 2: Application software has configured GTM Cluster 2 for PWM channels before MCAL initialization. But, MCAL PWM module is configured only to use GTM Cluster 0 and 1, hence MCAL will not configure any SFRs related to GTM Cluster 2. This may impact the PWM driver functionality. In this case, user software should restore SFRs to appropriate values such that the functionality of cluster 0 and cluster 1 is not impacted.

Note: *The list of SFRs initialized by an MCAL driver is provided in the module specific user manual in the API table.*

- If the SFR modification have no impact to the MCAL functionality, there is no action needed before invoking the MCAL module initialization.

Example: If the application has modified the SFRs related to ADC Kernel-1 before the module's initialization and the ADC Kernel-1 is not configured in MCAL ADC driver.

1.3 Multicore support

The MCAL software supports simultaneous invocation of APIs from multiple CPUs. Outcome of this requires partitioning of MCAL configuration and runtime data.

A typical TC3xx device has multiple CPUs. The software stack hosted on a device of the TC3xx family can be partitioned and executed from multiple CPUs simultaneously.

For more information, refer to module-specific multicore section in this document.

Note: *The memory sections marked as global should be relocated to the non-cached LMU region. In devices with no LMU, non-cached DSPR/ DLMU can be used.*

Table 3 **Module details**

Module	Multicore partitioning
ADC	EVADC hardware group
BFX	Supports multicore without resource partition. Invocation from all core in parallel is allowed.
CAN	MCMCAN node
CanTrcv_17_V9251	Not supported
CanTrcv_17_W9255	Not supported

(table continues...)

Generic information
Table 3 (continued) Module details

Module	Multicore partitioning
CRC	DMA based APIs - FCE channel. Non-DMA based APIs - Supports multicore without resource partition. Invocation from all cores in parallel is allowed.
DIO	Supports multicore without resource partition. Invocation from all cores in parallel is allowed.
DMA	DMA channels
DSADC	Not supported
ETH	Ethernet controller
FEE	Not supported
FLS	Not supported
FLSLOADER	Not supported
FR	Not supported
GPT	Logical GPT channels
HSSL	Not supported
I2C	Not supported
ICU	Logical ICU channels
IOM	Not supported
LIN	ASCLIN channels
MCALLIB	Supports multicore without resource partition. Invocation from all cores in parallel is allowed.
MCU	Supports multicore without resource partition. Invocation from all cores in parallel is allowed.
OCU	Logical OCU channels
PORT	Supports multicore without resource partition. Invocation from all cores in parallel is allowed.
PWM	Logical PWM channels
SENT	Sent channel
SMU	Supports multicore without resource partition. Invocation from all cores in parallel is allowed.
SPI	QSPI hardware Kernels
STM	STM timer
UART	Not supported
WDG	WDT of each core

Attention: For detailed multicore capability of each driver, refer to the Integration hints sections.

1.3.1 Multicore initialization

A module is initialized from the context of core, that is, each core must call the initialization function independently.

The following DETs are reported from initialization:

- <MOD>_E_MASTER_CORE_UNINIT: This error is reported if a slave core initialization is invoked prior to initialization of the master core.
- <MOD>_E_CORE_NOT_CONFIGURED: This error is reported if the API is invoked from a core which has no hardware resource allocated.
- <MOD>_E_CORE_MISMATCH: This error is reported if initialization of a module is invoked from slave core instead of master core (applicable to modules where initialization should be invoked by master core only).

Hardware resources are partitioned between cores. Therefore, initialization should be performed by the respective CPU.

For example, if PWM initialization is called from CORE1, all the PWM channels that are allocated to CORE1 in the Resource Manager module will be initialized. If initialization is called from master core, it initializes common resources of the module for example, CLC, global status, global registers and so on along with resources allocated to it.

However, initialization from slave is required only when resources are allocated to any of the slave core.

Note: *Same configuration pointer must be passed from master and slaves cores while invoking initialization.*

The following diagram shows sample invocation of PWM driver initialization from different cores.

Generic information

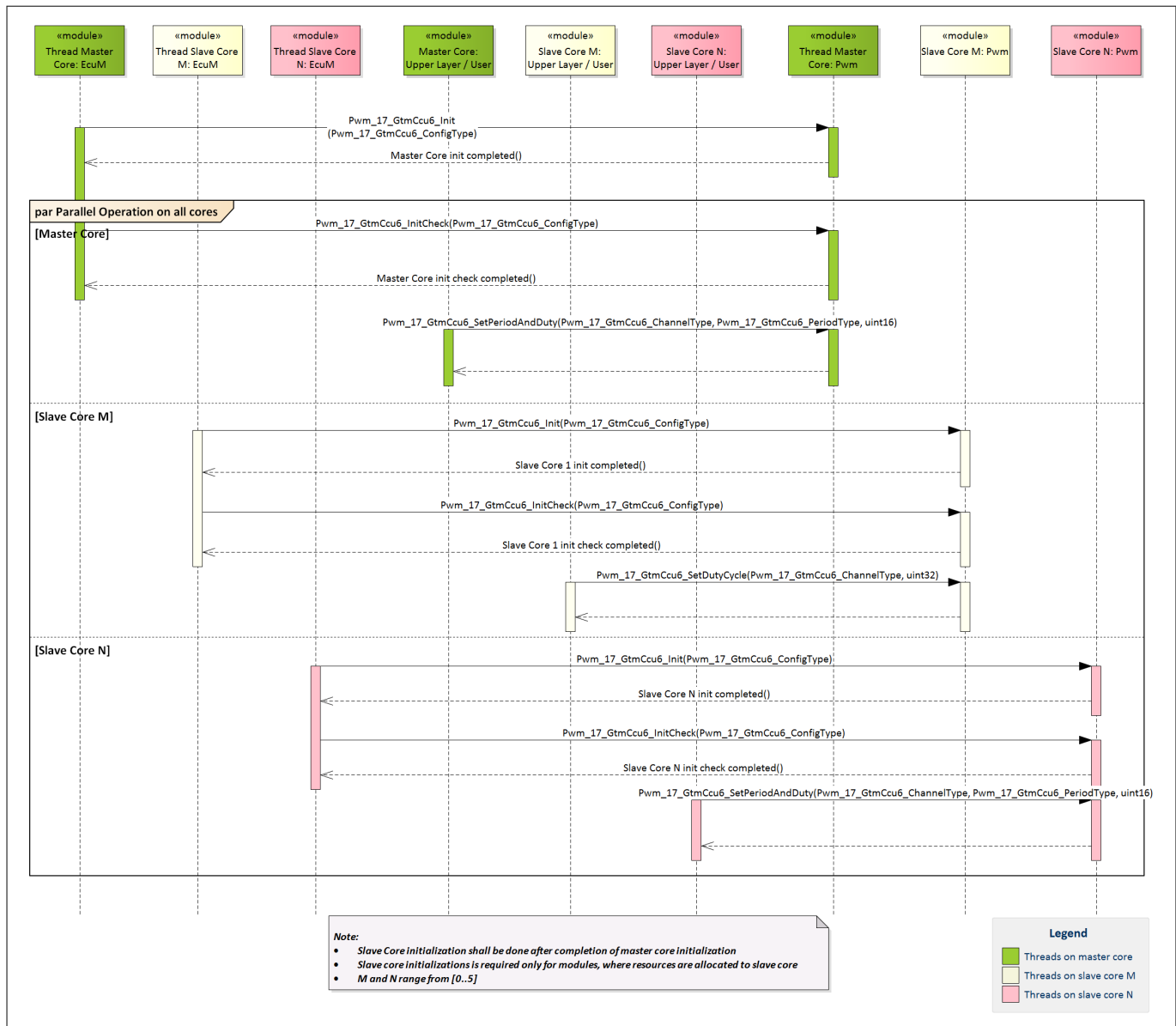


Figure 29 Multicore initialization

1.3.2 Multicore de-initialization

De-initialization should be done from the context of core; each core should call the de-initialization function independently.

The following DETs are reported from de-initialization:

- <MOD>_E_SLAVE_CORE_INIT: This error is reported if de-initialization from master core is invoked while any of the slave core is still in the initialized state.
- <MOD>_E_CORE_MISMATCH: This error is reported if de-initialization of a module is invoked from the slave core instead of the master core (applicable to modules where de-initialization should be invoked by the master core only).

Hardware resources are partitioned between cores. Therefore, de-initialization should be performed by the respective CPU.

For example, if PWM de-init is called from CORE1, all the PWM channels that are allocated to CORE1 in the Resource Manager module will be de-initialized. If de-init is called from the master core, it de-initializes

Generic information

common resources of the module for example, CLC, global RAM status, global registers and so on along with resources allocated to it. Slave cores can be initialized and de-initialized indefinitely as long as the master core is in the initialized state.

However, de-initialization from slave is required only when resources are allocated to any of the slave core.

The following diagram shows sample invocation of the ADC driver de-Initialization from different cores.

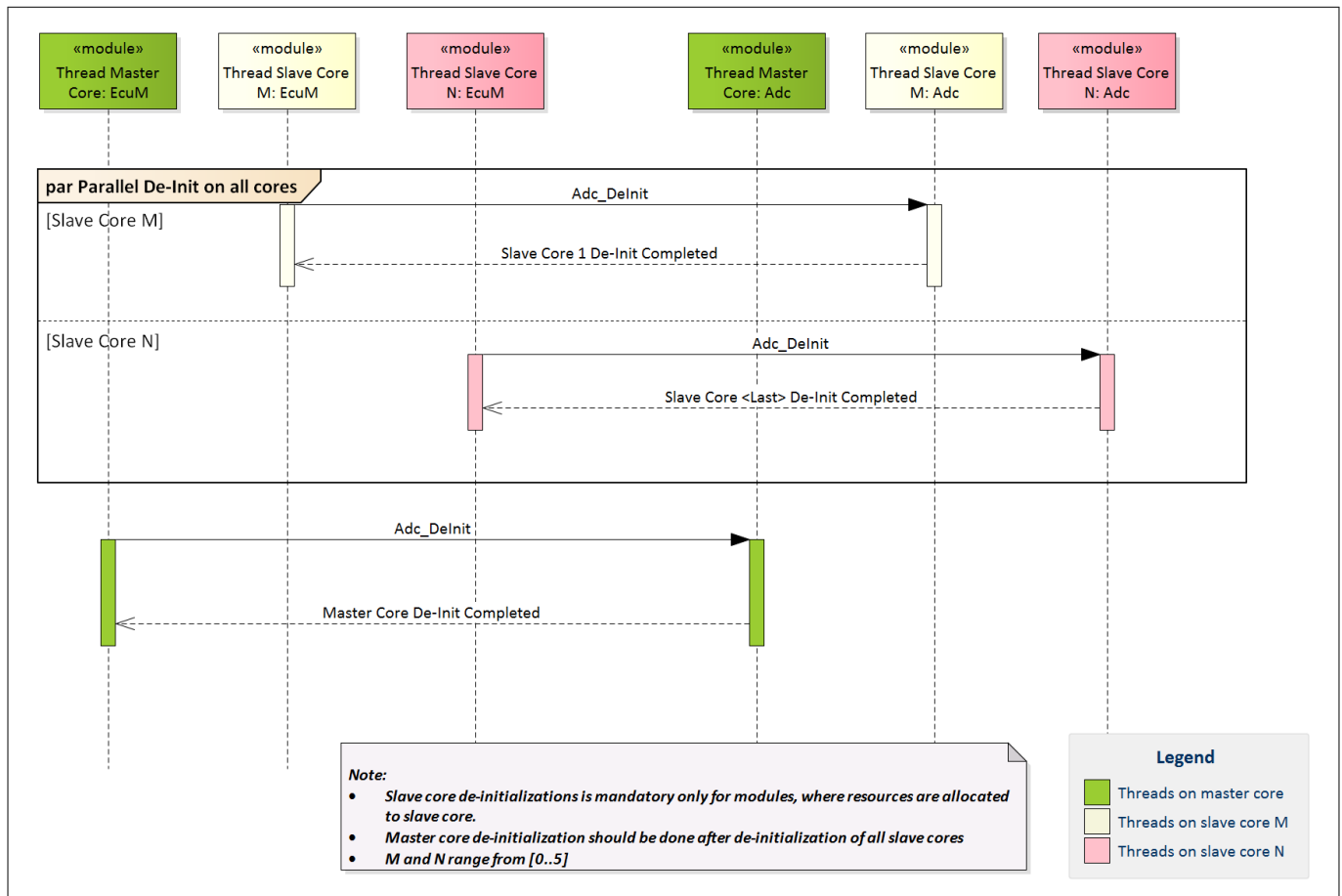


Figure 30 Multicore de-initialization

1.3.3 Multicore runtime

After initialization, runtime APIs can be called for valid channels. Error is reported if the channel passed is not configured for the core. The following diagram shows sample invocation of PWM runtime APIs from different cores.

The following DET is reported from runtime APIs:

- <MOD>_E_CORE_<RESOURCE>_MISMATCH: This error is reported if the API is invoked for an hardware resource that is not configured to be used by the current core.

During the development phase such errors can be identified easily by DETs.

Generic information

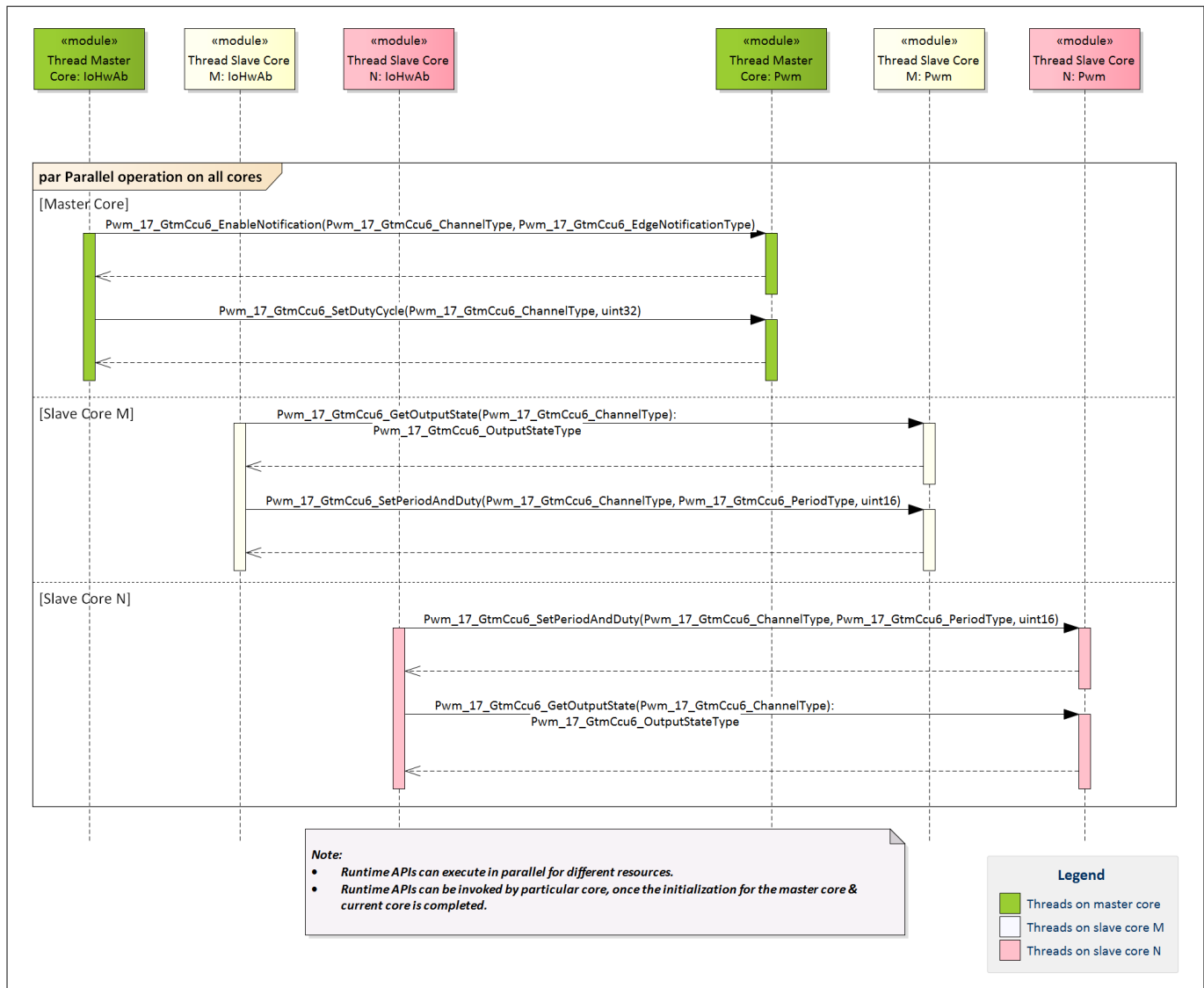


Figure 31 Multicore runtime

1.3.4 Multicore interrupts

Hardware resources assigned to a CPU must have their interrupts routed to the same CPU. This must be ensured by MCAL users.

Interrupts of resources allocated to a core if serviced by different cores will require complex implementation of multiple spin locks and possibly reduce the efficiency of the software.

The interrupt service routines provided by modules are capable to be executed on several CPU cores in parallel.

1.3.5 Resource Manager (ResourceM)

1.3.5.1 Description

The Resource Manager (ResourceM) module is responsible for allocating resources to cores. The unit of resource that is allocated to cores varies across modules. For example, logical PWM channels are allocated to cores for PWM module, QSPI kernels are allocated to cores in case of SPI module.

It is the responsibility of the integrator to restrict allocation of resource to single core.

The master core selection is done in the Resource Manager module. There is one designated master core in which the boot loader starts the master EcuM through EcuM_Init.

Since the Resource Manager is pre-compile, core related resource allocation shall be same across the different configuration variants for all modules in MCAL.

Note: In case a hardware resource is not allocated to any core, it is by default allocated to the master core.

1.3.5.2 Configuration containers and parameters

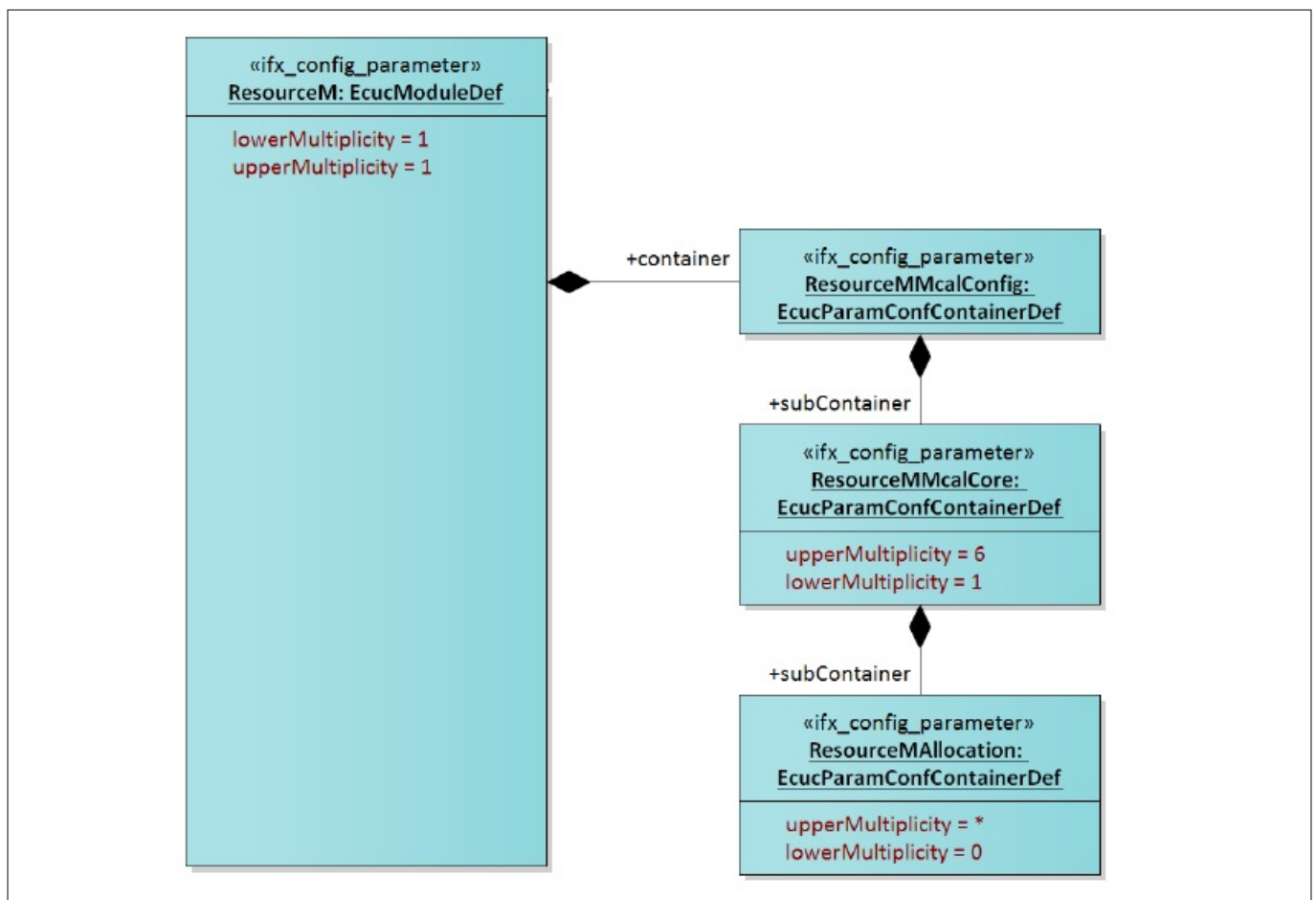


Figure 32 Configuration container hierarchy

1.3.5.2.1 Container: ResourceM

The Resource Manager module is used for allocating hardware resources to the core.

Generic information
1.3.5.2.2 Container: ResourceMAllocation

Container to configure resource allocations of each module assigned in the core.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

ResourceMModuleName
Table 4 Specification for ResourceMModuleName

Name	ResourceMModuleName		
Description	This parameter is used to select module name.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	ADC: ADC module selected CAN: CAN module selected DMA: DMA module selected ETH: ETH module selected GPT: GPT module selected ICU: ICU module selected LIN: LIN module selected PWM: PWM module selected OCU: OCU module selected SPI: SPI module selected STM: STM module selected SENT: SENT module selected Note: Availability of modules is based on the Release Notes.		
Default value	ADC		
Post-Build Variant Value	FALSE	Post-build variant multiplicity	-
Value Configuration Class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

ResourceMResourceRef
Table 5 Specification for ResourceMResourceRef

Name	ResourceMResourceRef		
Description	This parameter contains reference to resource of the selected module.		
Multiplicity	1..1	Type	EcucReferenceDef
Range	Reference to node		

(table continues...)

Generic information
Table 5 (continued) Specification for ResourceMResourceRef

Default value	NULL		
Post-Build Variant Value	FALSE	Post-build variant multiplicity	-
Value Configuration Class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	ResourceMModuleName		

1.3.5.2.3 Container: ResourceMMcalConfig

General container to select the master core and allocate resources.

ResourceMMasterCore

Table 6 Specification for ResourceMMasterCore

Name	ResourceMMasterCore		
Description	This parameter configures the master core.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	CORE0: CORE0 is selected as master core. CORE[x]: CORE[x] is selected as master core, where [x] depends on the device.		
Default value	CORE0		
Post-Build Variant Value	FALSE	Post-build variant multiplicity	-
Value Configuration Class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

ResourceMNumberOfActiveCores

Table 7 Specification for ResourceMNumberOfActiveCores

Name	ResourceMNumberOfActiveCores		
Description	This parameter is to configure the number of active cores in the current device variant. The default value of this parameter is taken from the corresponding device property file. Based on device variant if there is any difference in the number of active cores this parameter shall be configured accordingly		
Multiplicity	1..1	Type	EcucIntegerParamDef

(table continues...)

Generic information
Table 7 (continued) Specification for ResourceMNumberOfActiveCores

Range	1-6		
Default value	Depends on Device		
Post-Build Variant Value	FALSE	Post-build variant multiplicity	-
Value Configuration Class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency			
Autosar Version	Applicable for Autosar versions 4.2.2 and 4.4.0.		

1.3.5.2.4 Container: ResourceMMcalCore

Container to configure the MCAL core.

Post-Build Variant Multiplicity: FALSE

Multiplicity Configuration Class: Pre-Compile

ResourceMCoreID
Table 8 Specification for ResourceMCoreID

Name	ResourceMCoreID		
Description	Establishes a physical core ID mapping for each allocation.		
Multiplicity	1..1	Type	EcucEnumerationParamDef
Range	CORE0: CORE0 is selected as master core. CORE[x]: CORE[x] is selected as master core, where [x] depends on the device.		
Default value	CORE0		
Post-Build Variant Value	FALSE	Post-build variant multiplicity	-
Value Configuration Class	Pre-Compile	Multiplicity configuration class	-
Origin	IFX	Scope	LOCAL
Dependency	-		

1.3.5.3 Code generator plugin files

This section provides details of the code generator plugin files for the Resource Manager.

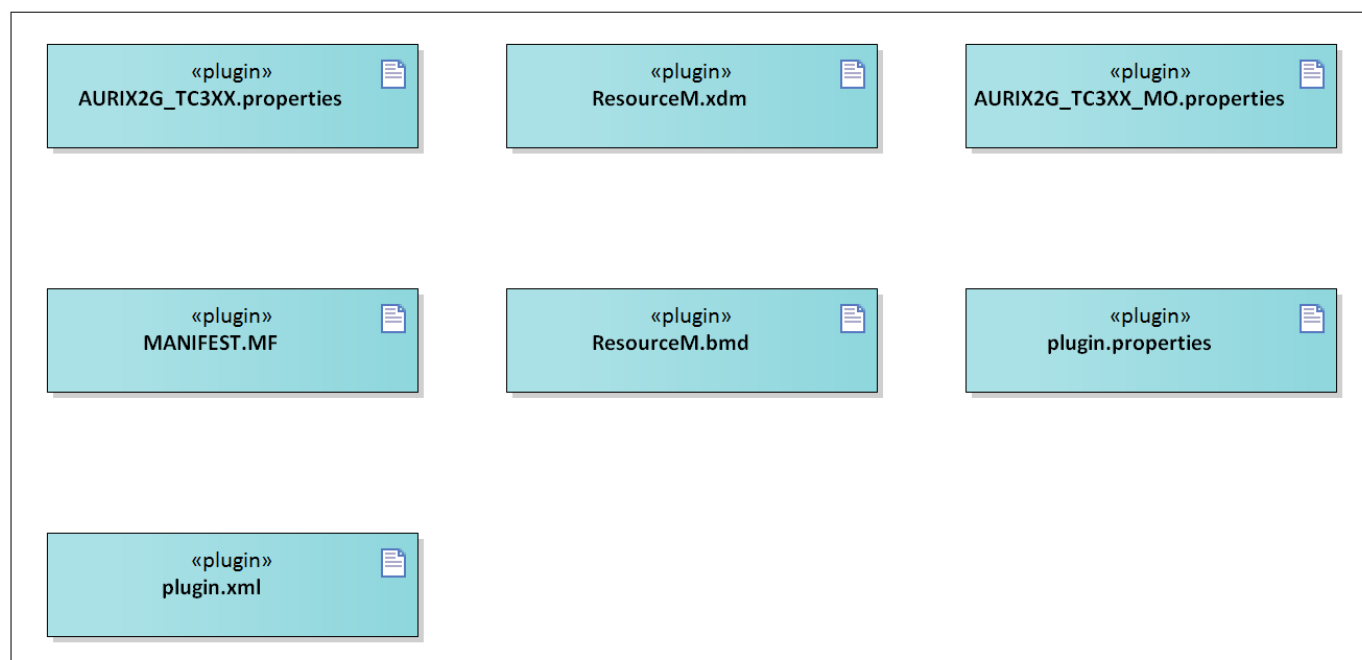
Generic information

Figure 33 Code generator plug-in files

Table 9 Code generator plug-in files

File name	Description
AURIX2G_TC3XX.properties	This file contains the variables defined for handling hardware derivatives. Here, TC3XX = Derivatives supported as per the release notes.
AURIX2G_TC3XX_MO.properties	This file contains the variables defined for handling hardware sub-derivatives. Here, TC3XX = Derivatives supported as per the release notes. Marking Option (MO) = based on defined marking options
MANIFEST.MF	Tresos plugin support file containing the metadata for the Resource Manager
ResourceM.bmd	AUTOSAR format XML data model schema file (for each device)
ResourceM.xdm	Tresos format XML data model schema file
plugin.properties	Tresos plugin support file for the Resource Manager
plugin.xml	Tresos plugin support file for the Resource Manager

1.3.5.4 Example usage

1.3.5.4.1 Configuring master core

CORE0 is selected as master core.

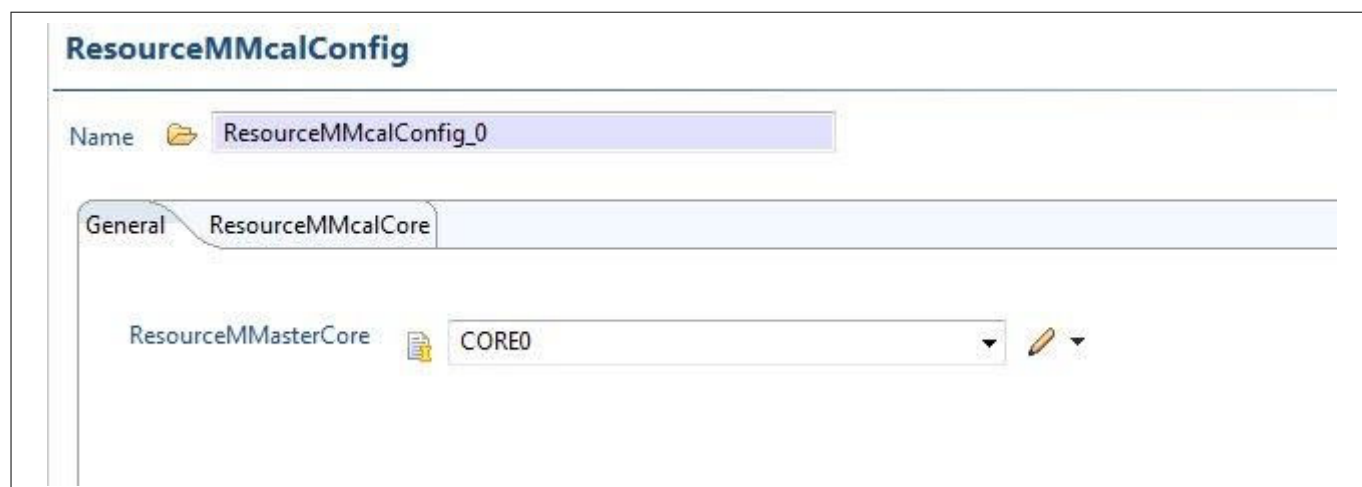


Figure 34 **Selecting master core**

1.3.5.4.2 **Configuring cores**

CORE0 and CORE1 configured for resource allocation.

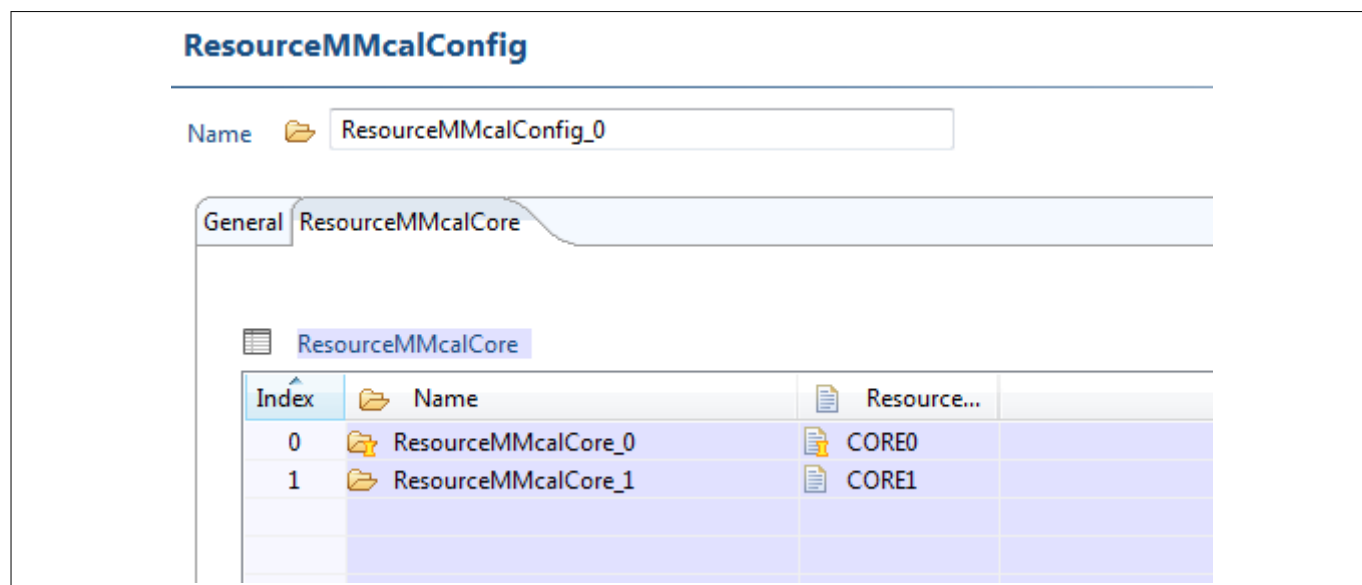
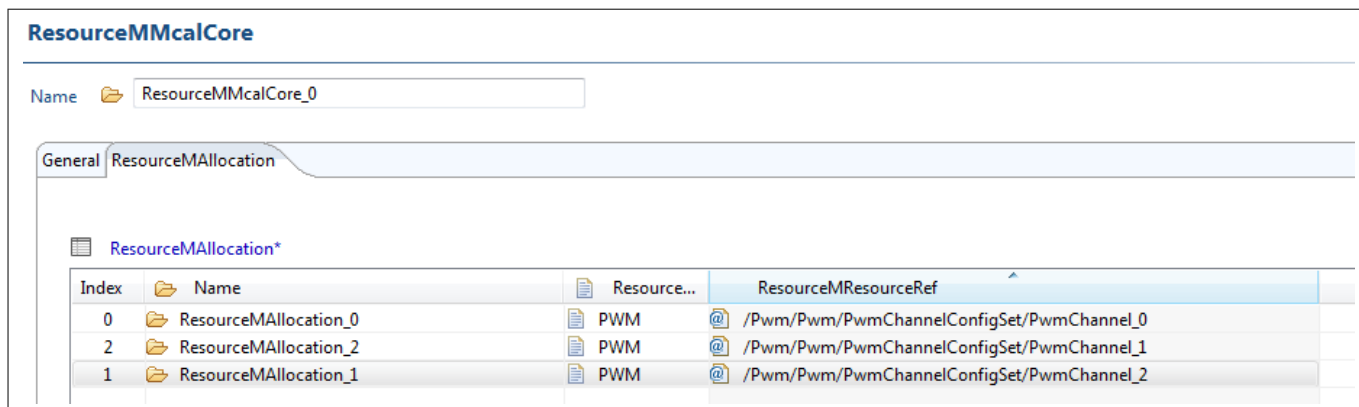


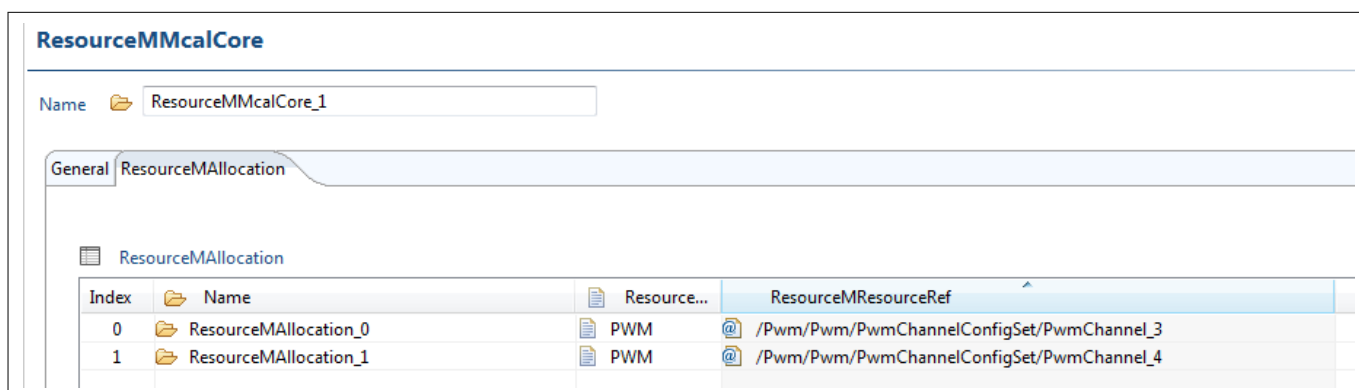
Figure 35 **Resource allocation**

1.3.5.4.3 **Configuring resources to core**

In the following figure Pwm_Channel0, Pwm_Channel1 and Pwm_Channel2 are allocated to CORE0.

Generic information

Figure 36 **Configuring resources**

Similarly Pwm_Channel3 and Pwm_Channel4 are allocated to CORE1.


Figure 37 **PWM allocation**

1.4 User mode execution

The TriCore™ CPU can execute in three privilege modes: User-0, User-1 and Supervisor modes. As SFR access is not possible in the User-0 mode, the MCAL drivers can be executed in the User-1 or Supervisor modes only. Each MCAL driver provides pre-compile configuration parameters to configure the execution mode of the CPU while executing Init/De-Init and runtime APIs independently.

Note: *If a MCAL driver is not accessing any SFR which can be updated in the Supervisor mode only, then the configuration of execution mode for that MCAL driver may not be available*

Examples of updating registers when CPU is in the User-1 and Supervisor modes are depicted in the following sub-sections.

1.4.1 Atomic update of the Supervisor mode register when CPU is in the User-1 mode

The following sequence diagram depicts the use case of updating a Supervisor mode protected register atomically while the CPU is executing in the User-1 mode. The application software or OS is responsible to implement the redirected function call from MCALLIB.

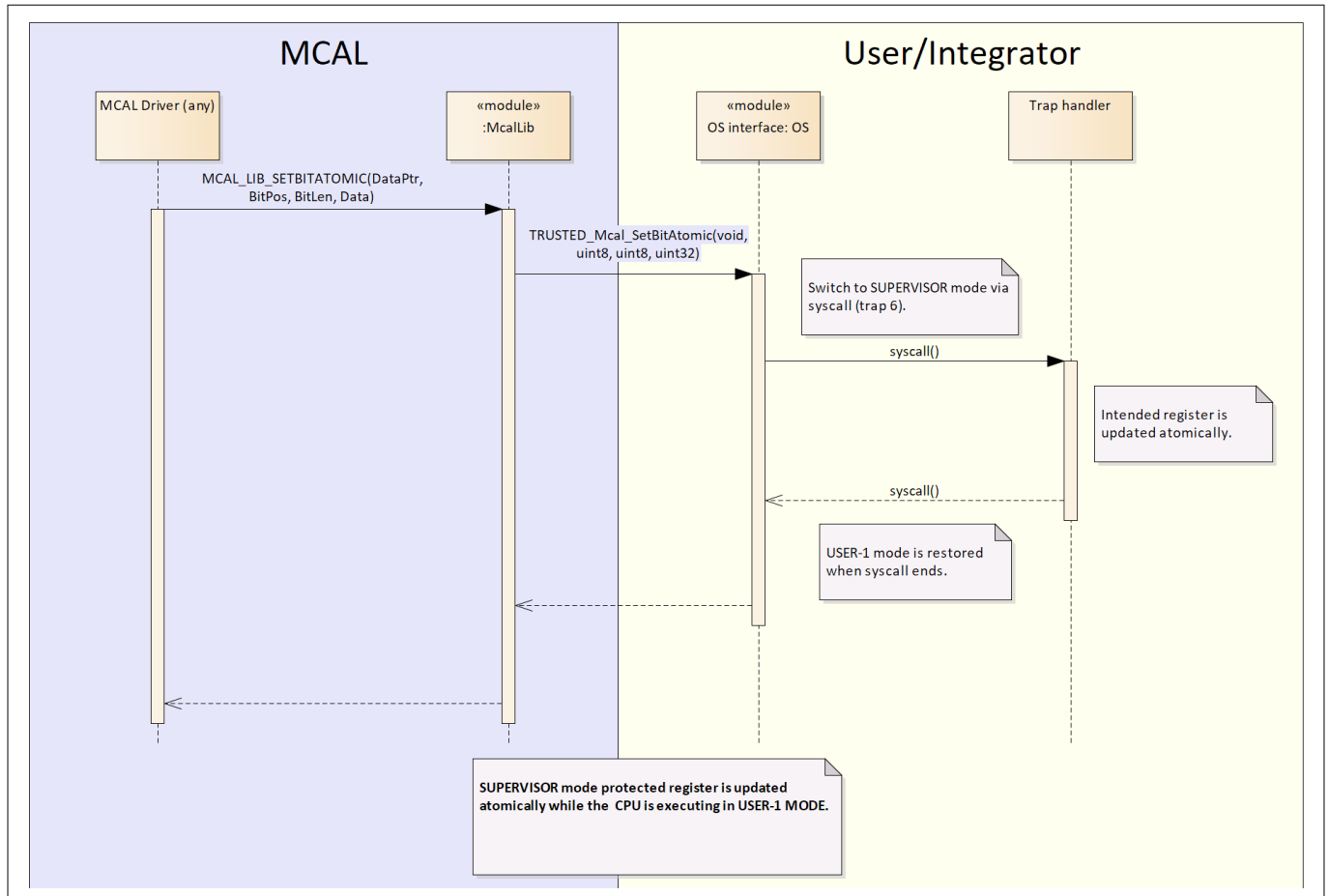


Figure 38 Atomic update of register when CPU is in the User-1 mode

1.4.2 Protected Supervisor mode register update when CPU is in the Supervisor mode

The following sequence diagram depicts the use case of updating an EndInit protected Supervisor mode register while the CPU is executing in the Supervisor mode. In this case, the driver API calls the `Mcal_WritePeripEndInitProtReg()` MCALLIB driver function. This API handles the EndInit protection and updates the protected register as well.

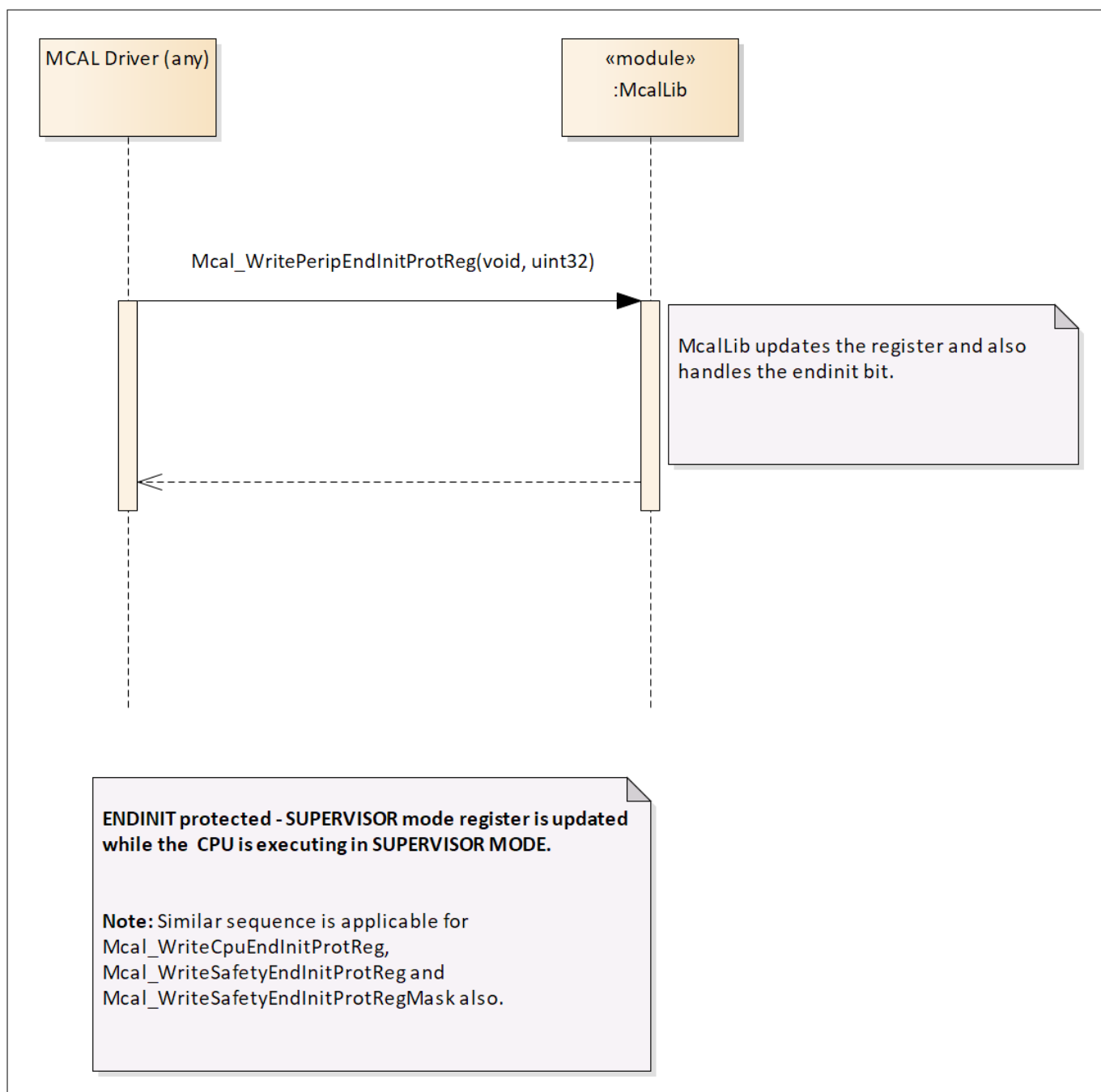


Figure 39 EndInit protected register update when CPU is in the Supervisor mode

1.4.3 Protected Supervisor mode register update when CPU is in the User-1 mode

The following sequence diagram depicts the use case of updating an ENDINIT protected Supervisor mode register while the CPU is executing in the User-1 mode. The application software or OS is responsible to implement the redirected function call from MCALLIB.

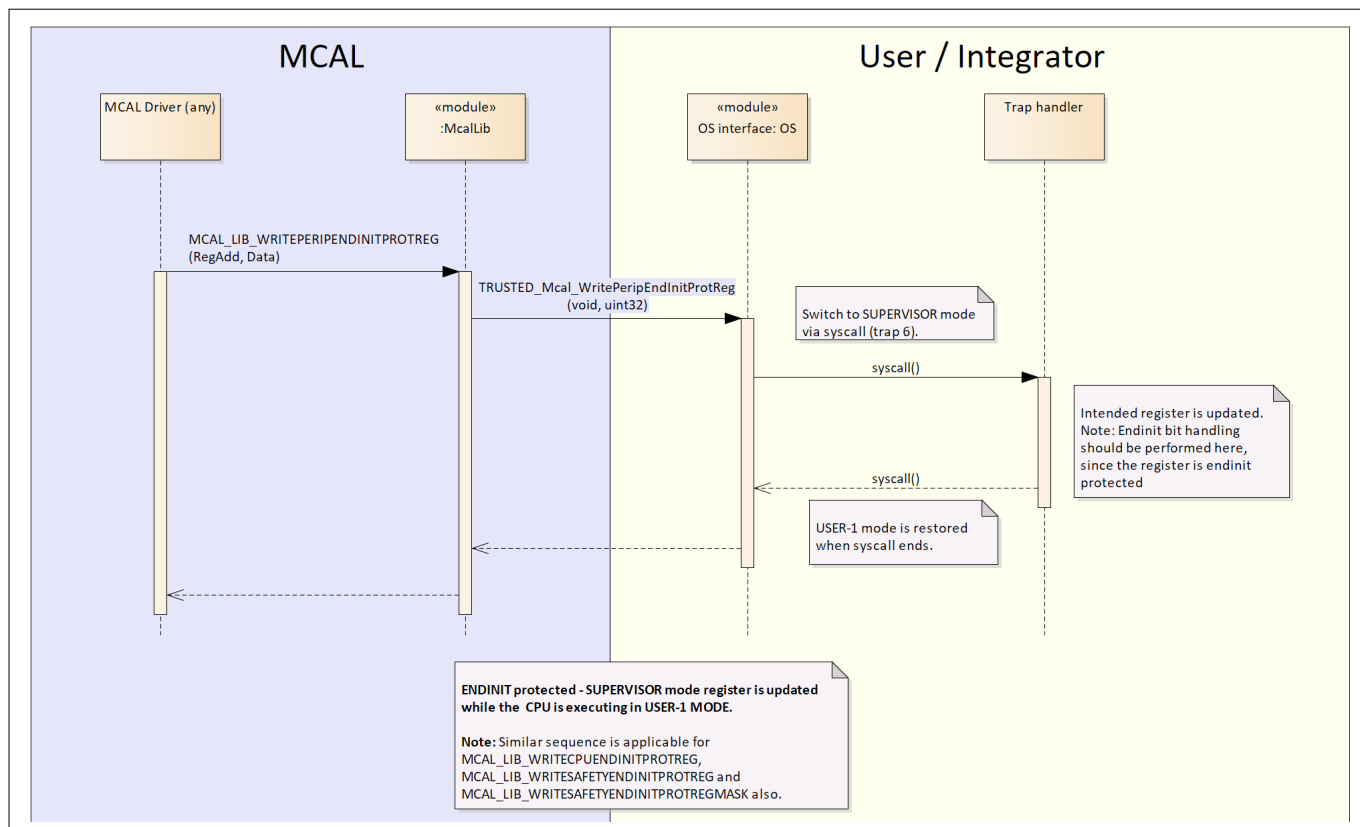
Generic information


Figure 40 EndInit protected register update when CPU is in the User-1 mode

1.4.4 Read Supervisor mode register when CPU is in the User-1 mode

The following sequence diagram depicts the use case of reading a Supervisor mode protected register while the CPU is executing in the User-1 mode. The application software or OS is responsible to implement the redirected function call from MCALLIB.

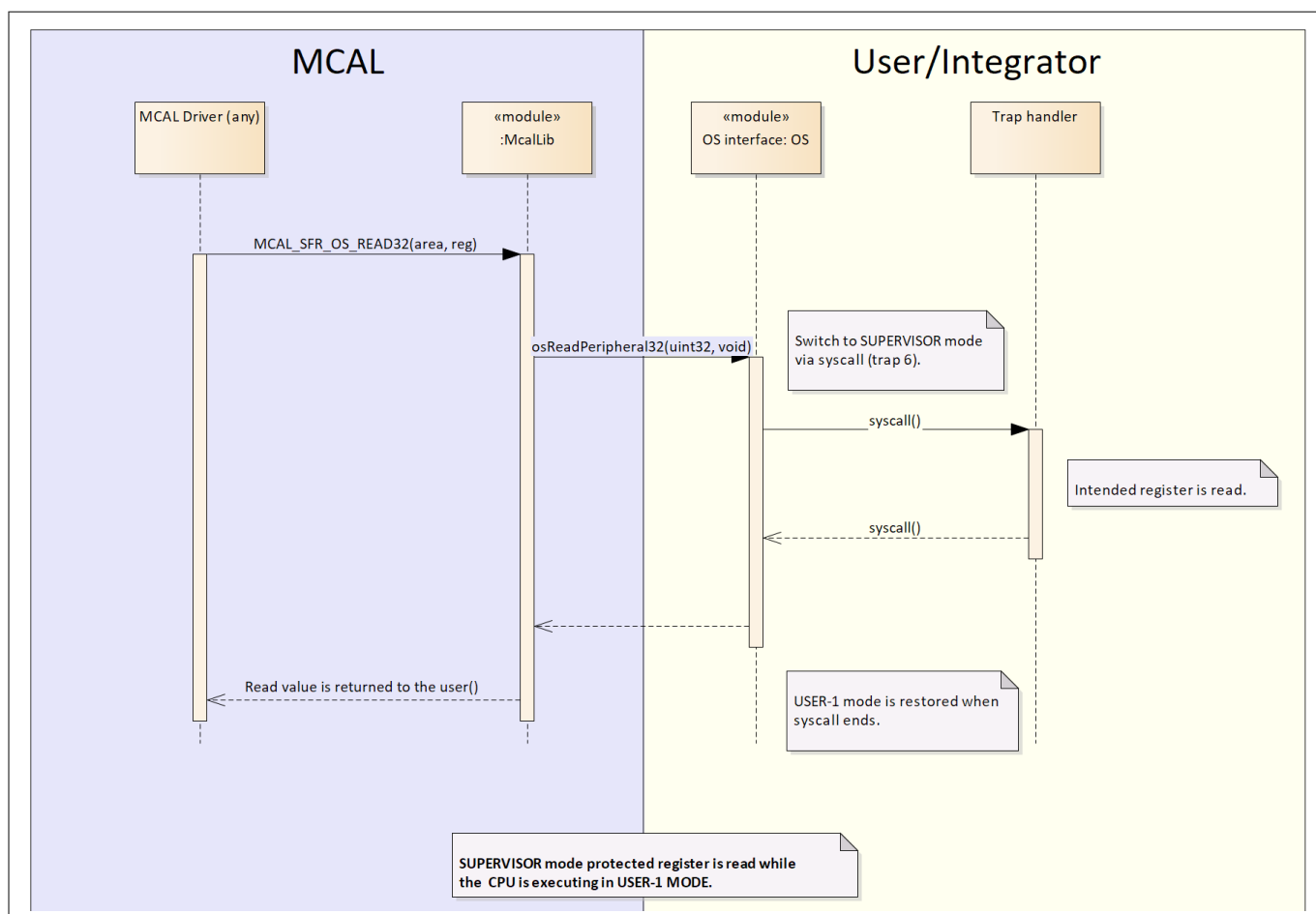


Figure 41 Register read when CPU is in the User-1 mode

1.4.5 Write to Supervisor mode register when CPU is in the User-1 mode

The following sequence diagram depicts the use case of updating a Supervisor mode protected register while the CPU is executing in the User-1 mode. The application software or OS is responsible to implement the redirected function call from MCALLIB.

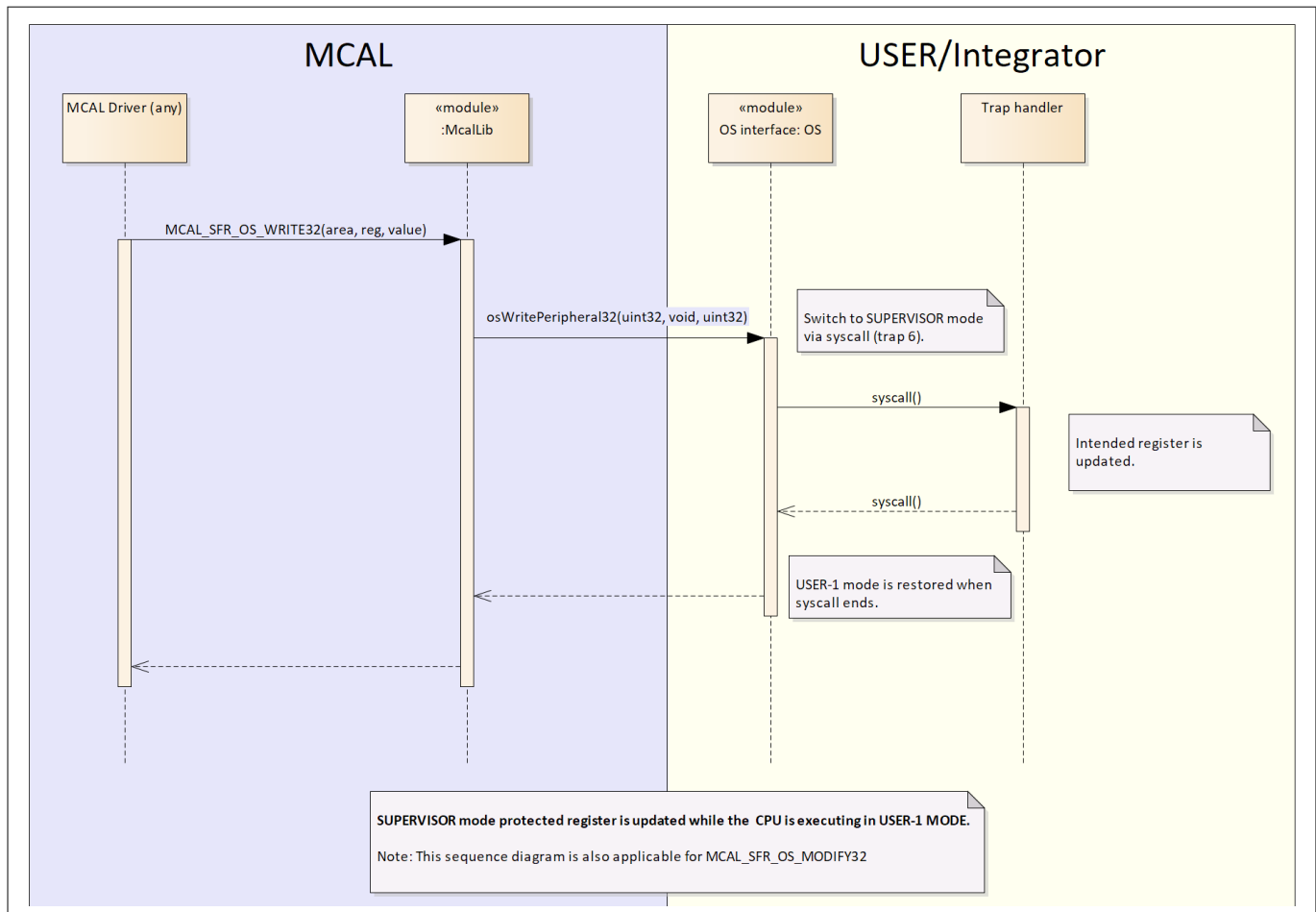


Figure 42 Register write when CPU is in the User-1 mode

1.4.6 Optimization technique for supervisor mode registers access

MCAL drivers operating in USER-1 mode would re-direct read/write access of supervisor mode protected registers to OS through `McalLib_OsStub.h` file.

When the runtime APIs execute in USER-1 mode but interrupt handlers execute as CAT-1, the application may implement the following optimization technique.

Note: CAT-1 operating mode implies that CPU is in supervisor mode.

```

if( CPU_PSW.IO == SV)
{
    Program SFR / Read SFR
}
else
{
    Use OS services to Program SFR / Read SFR in supervisor mode
}

```

The following points need to be considered while implementing above mentioned optimization.

- The implementation of re-directed APIs defined in `McalLib_OsStub.h` stays common between all MCAL drivers. Hence, user needs to evaluate if drivers operate in different modes.

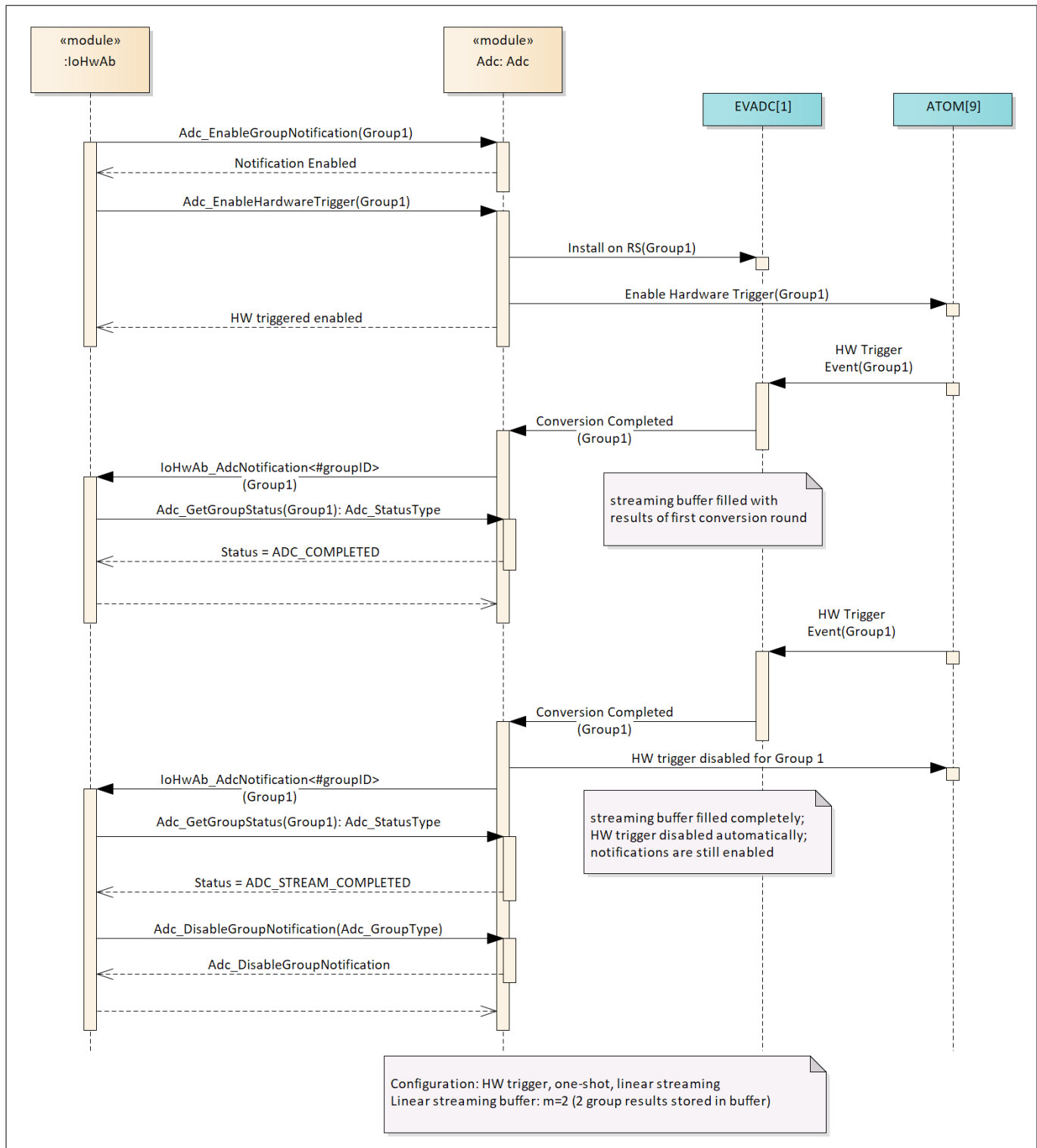
Generic information

- Execution time of CAT-1 ISRs in which supervisor mode registers are programmed would be reduced.
- Since, the re-direction APIs are common between Init/Deinit/Runtime/interrupt, there may be slight increase in execution time of Init/Deinit/Runtime functions.

1.5 Execution context of functions

The MCAL drivers provide both API functions and interrupt handlers. In general, the execution context for APIs shall be task and for interrupt handlers shall be interrupt. However, it is possible to invoke API functions in the context of the notification function invoked by the interrupt handler. An example to depict such a use-case is shown in the following diagram.

Note: *The execution context for each API is available in the Mod_Bswmd.arxml file.*

Generic information

Figure 43 Execution of API in interrupt context

1.6 Memory mapping

All the memory sections belong to the MemMap header file of the respective modules. The memory mapping section name for each module is derived as described in the subsequent sections. The user must use appropriate compiler pragmas in the `Mod_MemMap.h` header file to ensure correct allocation in memory space.

Generic information

Note: The naming convention for all memory sections is as per AUTOSAR, any deviation is captured separately in the module sections. All the memory sections used by the module are also present in the `<Module>_Bswmd.arxml` file under the element `MEMORY-SECTIONS`. User must refer to the EB tresos generated `<Module>_Bswmd.arxml` file for a complete list of memory sections per module.

1.6.1 Memory mapping for code

The memory mapping for code is as follows:

- Start Section: `[SNP]<VI><AI>_START_SEC<_username>_CODE<_mode>[_safety][_coreScope]`
- Stop Section: `[SNP]<VI><AI>_STOP_SEC<_username>_CODE<_mode>[_safety][_coreScope]`

Here,

`[]` = Indicates a mandatory field

`<>` = Indicates an optional field

SNP = Module abbreviation

VI = vendorId of the BSW module (applicable only for modules with upper multiplicity greater than 1). For software provided by Infineon, this value is fixed to 17.

AI = vendorApiInfix of the BSW module (applicable only for modules with upper multiplicity greater than 1)

safety = QM, ASIL_B

coreScope = GLOBAL, CORE0, CORE1, CORE2, CORE3, CORE4, CORE5, LOCAL

Note: 'LOCAL' is used for the modules that does not support multicore.

username = this is an optional field and may be used for functions in code which are required to be placed in specific memory regions

mode = FAST, SLOW

Note: The memory sections of this category should be mapped to `.text` in linker command file.

1.6.2 Memory mapping for callout code

The memory mapping for callout code is as follows:

- Start Section: `[SNP]<VI><AI>_START_SEC[_username]_CALLOUT_CODE<_mode>[_safety][_coreScope]`
- Stop Section: `[SNP]<VI><AI>_STOP_SEC[_username]_CALLOUT_CODE<_mode>[_safety][_coreScope]`

Here,

`[]` = Indicates a mandatory field

`<>` = Indicates an optional field

SNP = Module abbreviation

VI = vendorId of the BSW module (applicable only for modules with upper multiplicity greater than 1). For software provided by Infineon, this value is fixed to 17.

AI = vendorApiInfix of the BSW module (applicable only for modules with upper multiplicity greater than 1)

username = The specific username for each callout function must be specified.

safety = QM, ASIL_B

coreScope = GLOBAL, CORE0, CORE1, CORE2, CORE3, CORE4, CORE5, LOCAL

Note: 'LOCAL' is used for the modules that does not support multicore.

Generic information

mode = FAST, SLOW

Note: *The memory sections of this category should be mapped to .text in linker command file.*

1.6.3 Memory mapping for configuration data

The memory mapping for configuration data is as follows:

- Start Section: [SNP]_<VI>_<AI>_START_SEC_CONFIG_DATA[_safety][_coreScope][_alignment]
- Stop Section: [SNP]_<VI>_<AI>_STOP_SEC_CONFIG_DATA[_safety][_coreScope][_alignment]

Here,

[] = Indicates a mandatory field

<> = Indicates an optional field

SNP = Module abbreviation

VI = vendorId of the BSW module (applicable only for modules with upper multiplicity greater than 1). For software provided by Infineon, this value is fixed to 17.

AI = vendorApiInfix of the BSW module (applicable only for modules with upper multiplicity greater than 1)

safety = QM, ASIL_B

coreScope = GLOBAL, CORE0, CORE1, CORE2, CORE3, CORE4, CORE5, LOCAL

Note: *'LOCAL' is used for the modules that does not support multicore.*

Note: *As per Autosar the [_coreScope] is not recommended for configuration data. Hence this is the deviation from Autosar specification.*

alignment = UNSPECIFIED, 8, 16, 32 (in bits). In cases, where a higher order memory alignment is required it should be specified at the bit level. For example, 256 is to be used for a 32-byte alignment.

Note: *The memory sections of this category should be mapped to .rodata in linker command file.*

1.6.4 Memory mapping for constants

The memory mapping for constant data is as follows:

- Start Section: [SNP]_<VI>_<AI>_START_SEC_CONST[_safety][_coreScope][_alignment]
- Stop Section: [SNP]_<VI>_<AI>_STOP_SEC_CONST[_safety][_coreScope][_alignment]

Here,

[] = Indicates a mandatory field

<> = Indicates an optional field

SNP = Module abbreviation

VI = vendorId of the BSW module (applicable only for modules with upper multiplicity greater than 1). For software provided by Infineon, this value is fixed to 17.

AI = vendorApiInfix of the BSW module (applicable only for modules with upper multiplicity greater than 1)

safety = QM, ASIL_B

coreScope = GLOBAL, CORE0, CORE1, CORE2, CORE3, CORE4, CORE5, LOCAL

Note: *'LOCAL' is used for the modules that does not support multicore.*

Generic information

Note: As per Autosar the `[_coreScope]` is not recommended for constant data. Hence this is the deviation from Autosar specification.

alignment = UNSPECIFIED, 8, 16, 32 (in bits). In cases, where a higher order memory alignment is required it should be specified at the bit level. For example, 256 is to be used for a 32-byte alignment.

Note: The memory sections of this category should be mapped to `.rodata` in linker command file.

1.6.5 Memory mapping for variables

The memory mapping for variable is as follows:

- Start Section: `[SNP]_<VI>_<AI>_START_SEC_VAR<_mode>[_initPolicy][_safety][_coreScope][_alignment]`
- Stop Section: `[SNP]_<VI>_<AI>_STOP_SEC_VAR<_mode>[_initPolicy][_safety][_coreScope][_alignment]`

Here,

`[]` = Indicates a mandatory field

`<>` = Indicates an optional field

SNP = Module abbreviation

VI = vendorId of the BSW module (applicable only for modules with upper multiplicity greater than 1). For software provided by Infineon, this value is fixed to 17.

AI = vendorApiInfix of the BSW module (applicable only for modules with upper multiplicity greater than 1)

initPolicy = CLEARED, INIT

safety = QM, ASIL_B

coreScope = GLOBAL, CORE0, CORE1, CORE2, CORE3, CORE4, CORE5, LOCAL

Note: 'LOCAL' is used for the modules that does not support multicore.

alignment = UNSPECIFIED, 8, 16, 32 (in bits). In cases, where a higher order memory alignment is required it should be specified at the bit level. For example, 256 is to be used for a 32-byte alignment.

mode = FAST, SLOW

Note: The memory sections of this category with `initPolicy = CLEARED` should be mapped to `.bss` in linker command file. If the mode is explicitly set to FAST in the memory section then, it shall be mapped to `.zbss`.

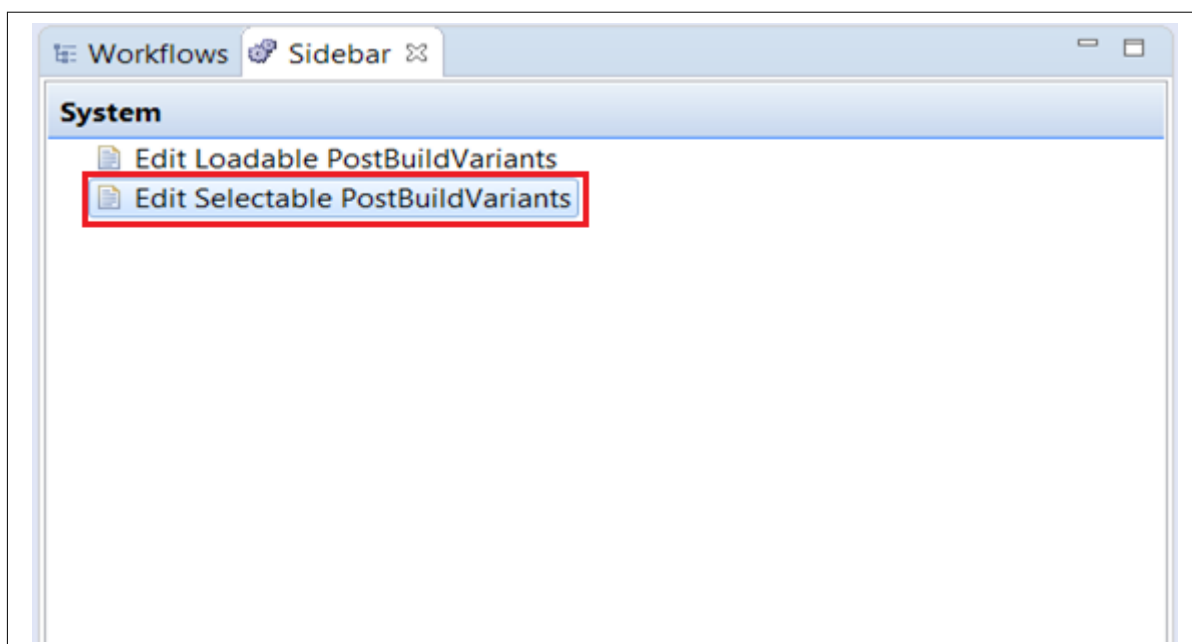
Note: The memory sections of this category with `initPolicy = INIT` should be mapped to `.data` in linker command file. If the mode is explicitly set to FAST in the memory section then, it shall be mapped to `.zdata`.

1.7 Variation point support

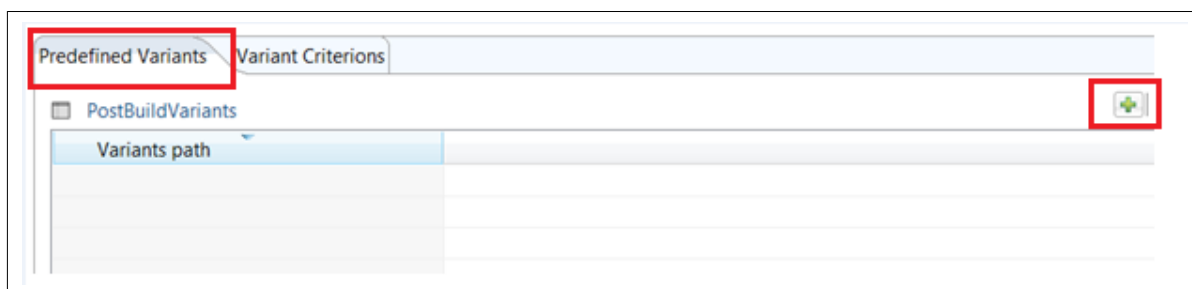
1.7.1 Configuring the Sidebar view

The procedure to prepare variation point supported EB tresos configuration project is as follows:

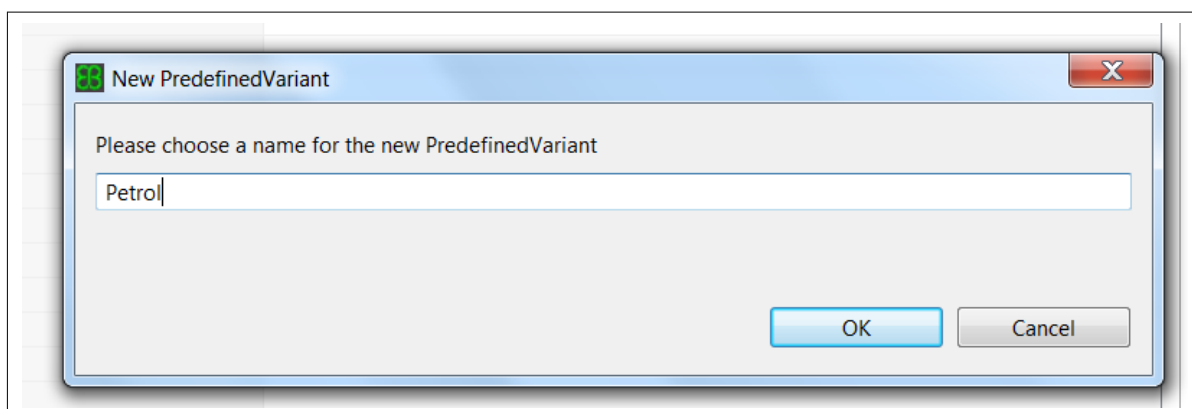
1. Start the EB tresos GUI and create EB tresos configuration project with the required modules along with that add the EcuC module in the project.
2. In the EB tresos GUI menu, select **Window > Show View > Sidebar**.

**Figure 44** PostBuildVariables configuration selection

3. Double-click the **Edit Selectable PostBuildVariables** option.

**Figure 45** Adding predefined variants

4. In the **Predefined Variables** tab, add a new predefined variants by clicking the plus (+) icon.

**Figure 46** New Predefined variant

5. Specify a name for the new PredefinedVariant. For example, Petrol.

Generic information

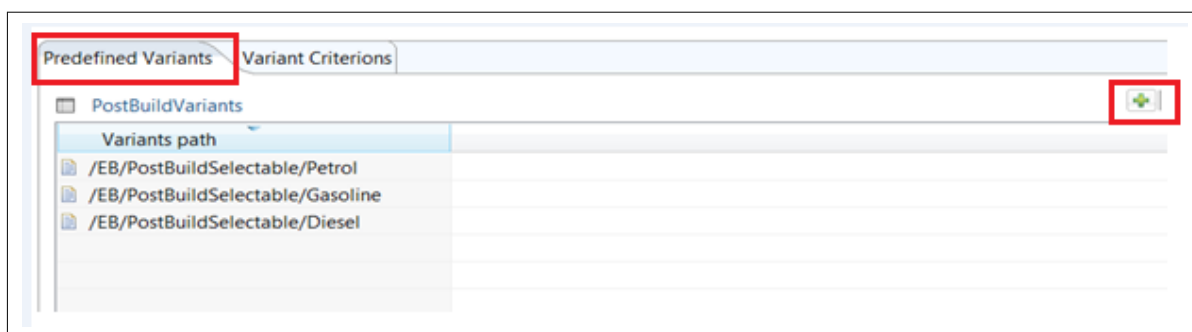


Figure 47 Predefined variables addition

6. Add the required number of variants in the **Predefined Variables** tab. For example, in the above figure, the following predefined variables are added: Petrol, Gasoline and Diesel.

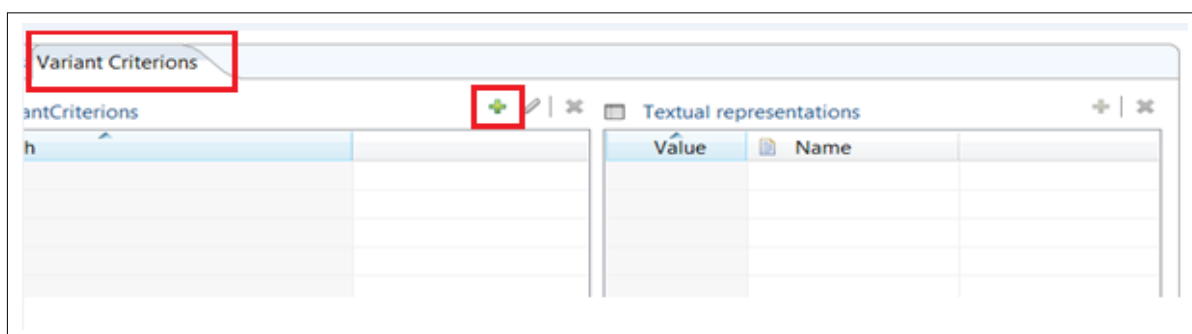


Figure 48 Variant criteria addition

7. In the **Variant Criteria** tab, click the plus (+) icon to add a new criterion.

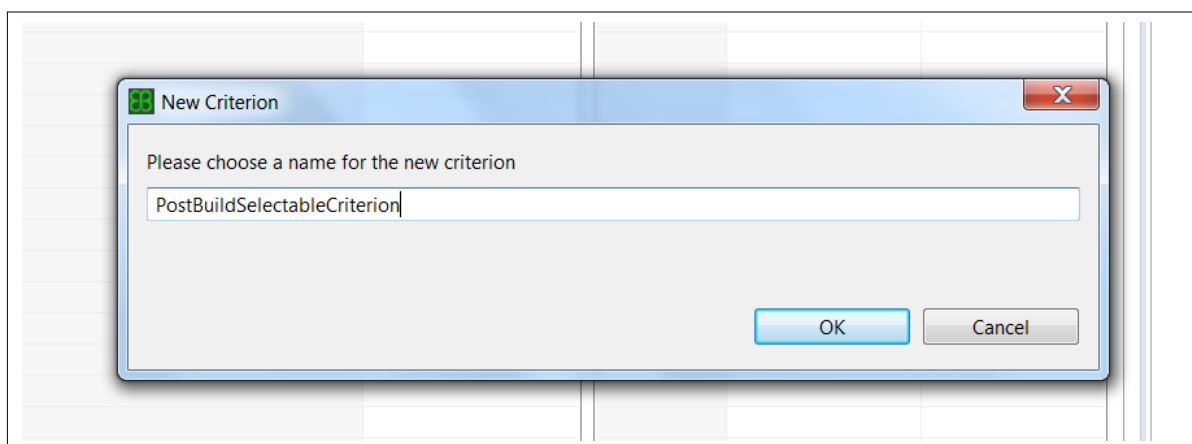


Figure 49 New criterion

8. In the **New Criterion** pop-up, specify a name for the new criterion. For example, PostBuildSelectableCriterion.

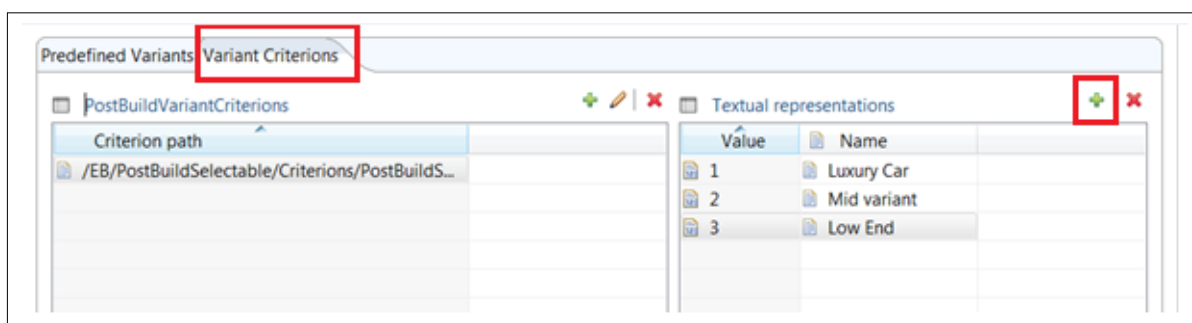


Figure 50 Variant criteria

Generic information

9. In the **Variant Criteria** tab, select the Criterion path. This will enable the **Textual representations** options.
10. In the **Textual representations** pane, click the plus (+) icon to add and edit the *Value* and *Name* fields.
Value: <Userdefined_numericvalue>. Specify a value, for example, 1, 2, 3
Name: <Userdefined_Name>. Specify a name, for example, Luxury Car, Mid variant, Low End

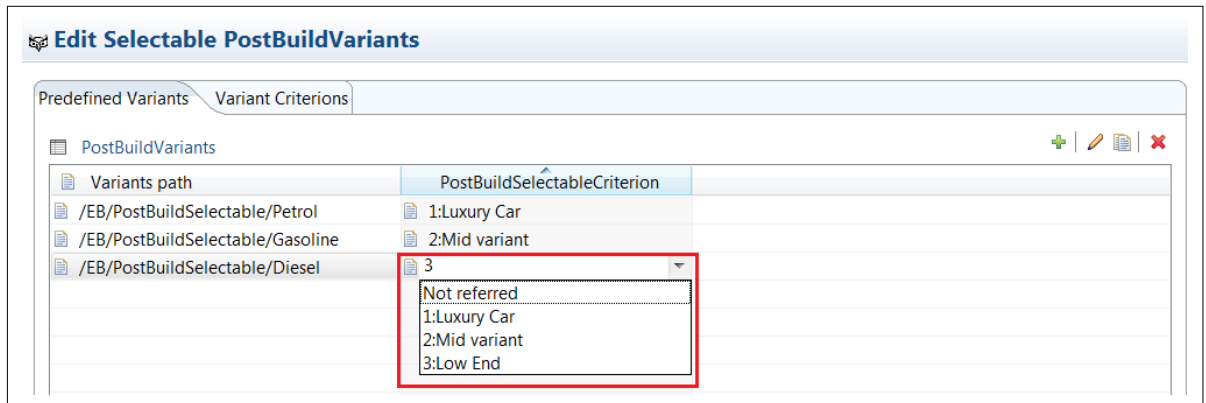


Figure 51 Edit variants

11. In the **Predefined Variables** tab, assign the selectable criterion for each PostBuildVariables by clicking the field and selecting a value from the drop-down list.
12. Close the **Edit Selectable PostBuild Variables** tab. If you keep the **Edit Selectable PostBuild Variables** tab opened, other module configurations cannot be done in the editor window.

1.7.2 Configuring EcuC module

The procedure to set up the EcuC module configuration is as follows:

1. In the **Project Explorer** window, double-click the EcuC module for the selected project.

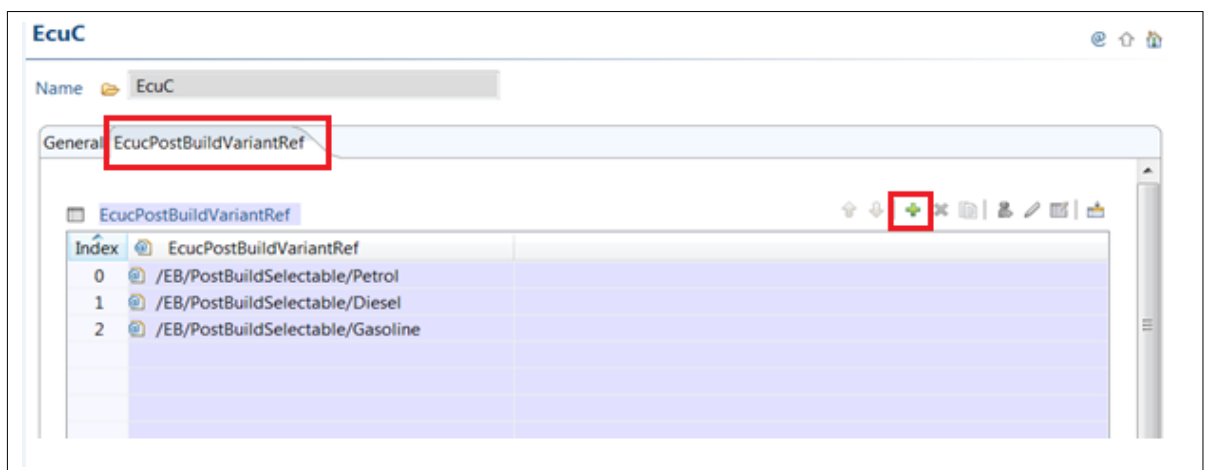
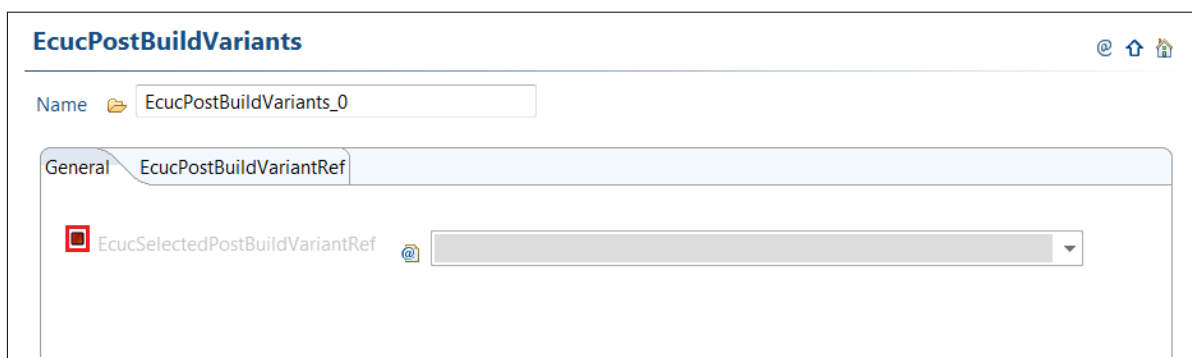
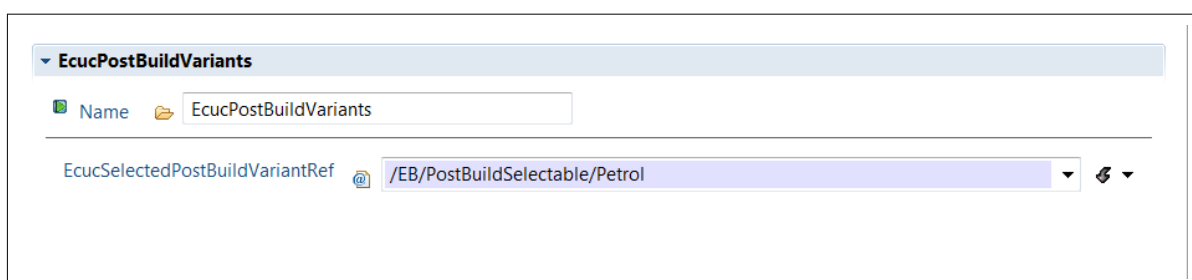


Figure 52 Selecting predefined variants

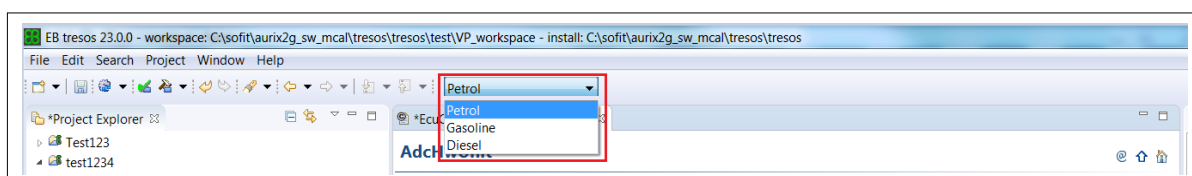
2. Add post-build variant references from the in the EcucPostBuildVariables/**EcucPostBuildVariantsRef** container.


Figure 53 Enable parameter

3. To enable the **EcuCSelectedPostBuildVariantRef** optional parameter, click the Red icon.


Figure 54 Selecting currently activated variant

4. Select a variant, you want to configure, from the drop-down list.
 - *Option (i):* Select the currently working activated variant Petrol from the **EcuCSelectedPostBuildVariantRef** reference parameter.


Figure 55 Selecting variant

- *Option (ii):* Select the currently working activated variant Petrol from the EB tresos menu bar. You can choose either of the options.

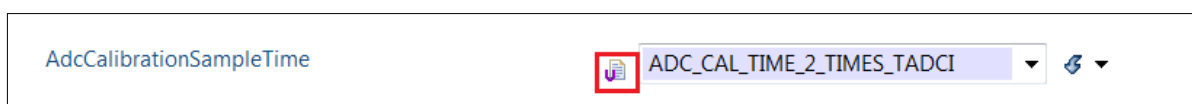
1.7.3 Configuring modules

The following module configuration procedure is for the Adc module, specifically for the AdcCalibrationSampleTime parameter. For other modules, follow the similar steps to configure the variation point supported containers/parameters.

The procedure to configure the variation point supported modules is as follows:

1. In the **Project Explorer** window, double-click a post-build module, for example, Adc module.
2. In the Adc module, select the AdcCalibrationSampleTime parameter (full path: AdcConfigSet/AdcHwUnit/AdcCalibrationSampleTime) and configure values for different variants.

Variant 1: Petrol variant configuration


Figure 56 Parameter configuration

Generic information

3. Select the `AdcCalibrationSampleTime` parameter value from the drop-down list. For example, `ADC_CAL_TIME_2_TIMES_TADCI`.
4. Right-click the `AdcCalibrationSampleTime` parameter by clicking the icon.

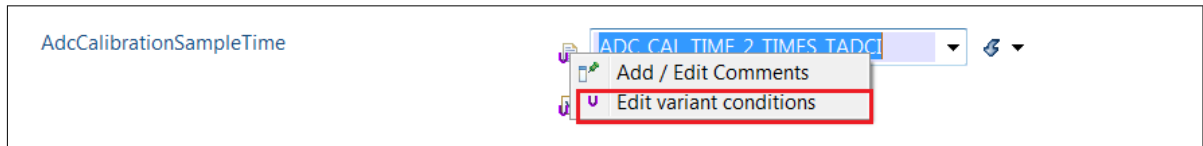


Figure 57 Parameter variant condition settings

5. Select the `Edit variant conditions` option.

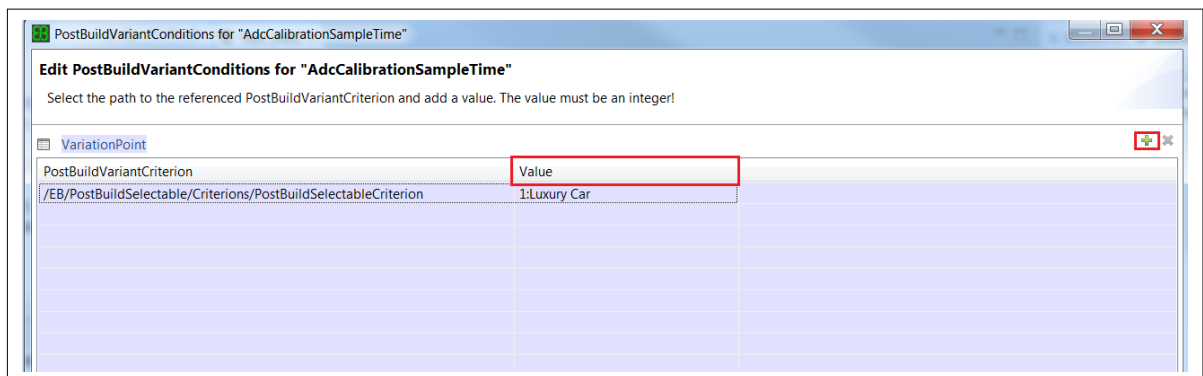


Figure 58 Selecting variant criterions

6. Click the plus (+) icon.
7. Select the `PostBuildVariantCriterion` from the drop-down list.
8. Select a value from the drop-down list for the selected variant (which is already created and assigned, refer to *Section 1.5.1*).

Repeat all the steps 1 through 8 to configure other post-build variant parameters (with `POSTBUILDVARIANTVALUE = true`).

Variant 2: Diesel variant configuration

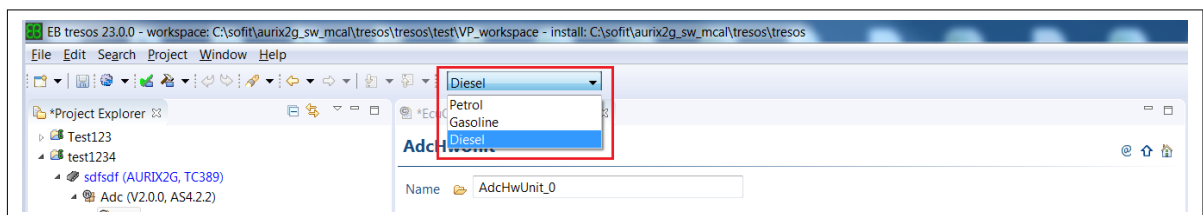


Figure 59 Selecting variants

9. Select the **Diesel** option in the `EcuC/EcucPostBuildVariants/EcucPostBuildVariantRef` parameter or from the EB tresos tool menu.

Note: When you are configuring for the next variant Diesel configuration, the `AdcCalibrationSampleTime` parameter value can be changed to a different value (for example, `ADC_CAL_TIME_4_TIMES_TADCI`).

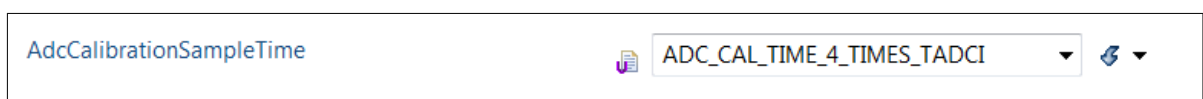


Figure 60 Parameter configuration

10. Right-click the `AdcCalibrationSampleTime` parameter v icon.

Generic information

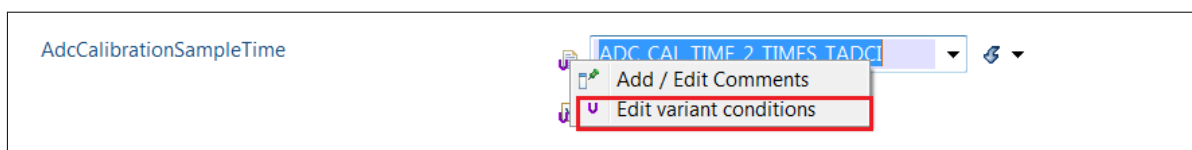


Figure 61 Parameter variant condition settings

11. Select the **Edit variant conditions** from the drop-down list.

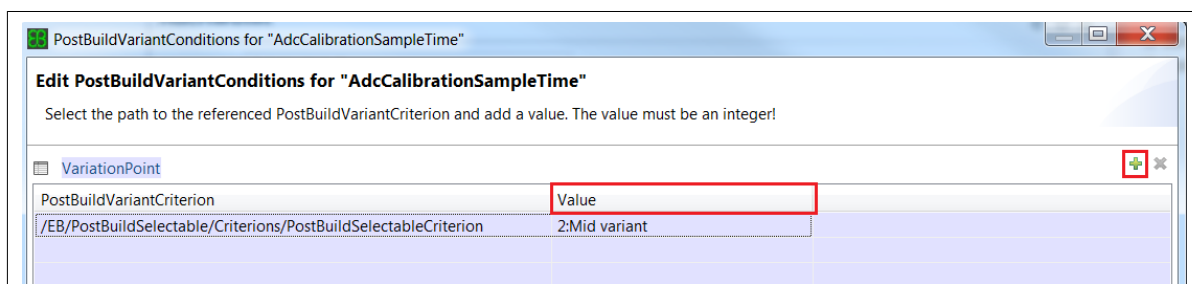


Figure 62 Selecting variant criterions

12. Select a value from the drop-down list for the selected variant (which is already created and assigned, refer to Section 1.5.1).

Similarly, you can configure parameter values for the Gasoline variant.

Note: Ensure to map the correct *PostBuildVariantCriterion* value for the selected variant.

Repeat all the steps 9 through 12 to configure other post-build variant parameters (with POSTBUILDVARIANTVALUE = true).

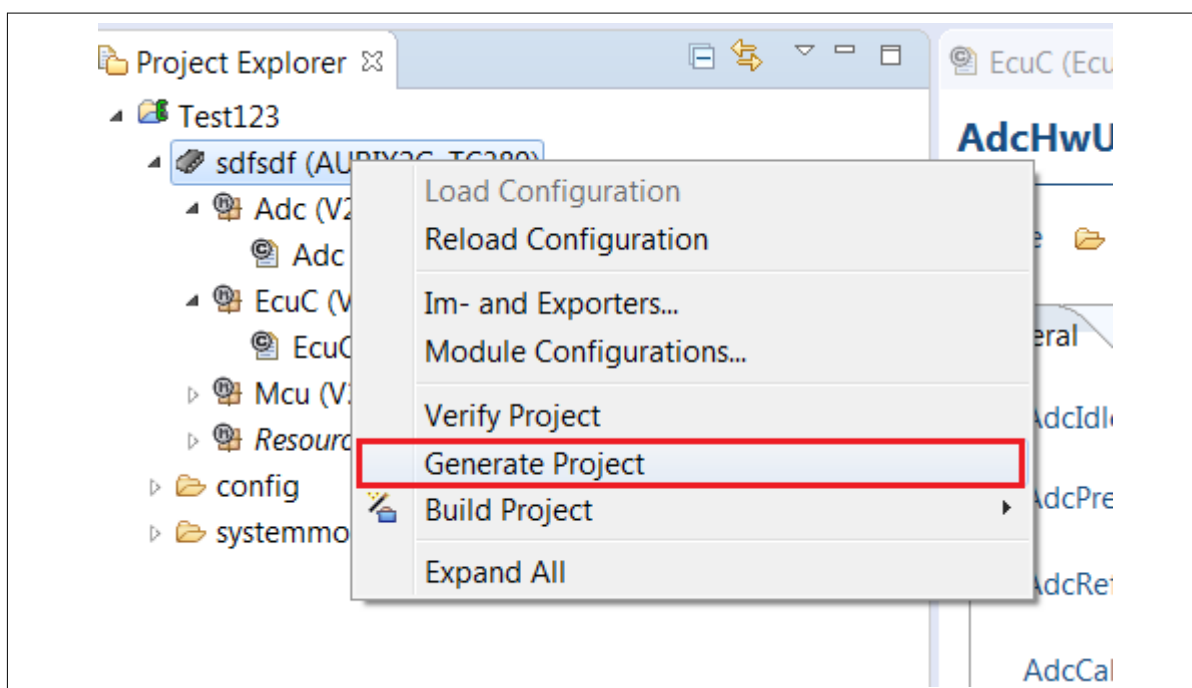
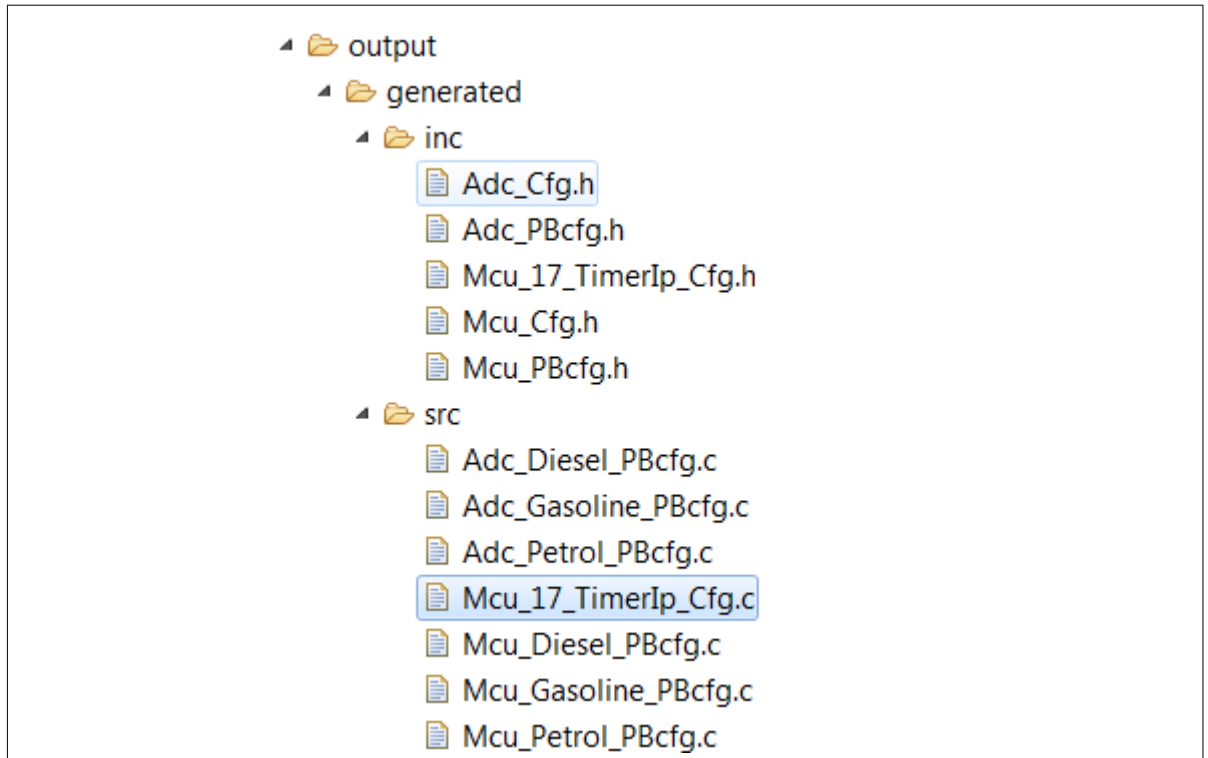


Figure 63 Generate project

13. To generate the output files for all the variants, click the **Generate Project** option. The generated files are displayed in the **Project Explorer** window.

**Figure 64** **Generated output**

1.7.4 **Post-build variant multiplicity and Multiplicity configuration class**

In general, the drivers provide configuration for multiple channels, groups, blocks, and so on (for example, ADC driver provides AdcGroup container, SPI driver provides SpiJob container). While configuring such containers the user must take into account the post-build variant multiplicity and multiplicity configuration class of such containers.

The user must consider the following points:

- If the post-build variant multiplicity is set to FALSE for a container, then the number of instances of such container cannot change across the post-build variants.
- If the multiplicity configuration class is set to Pre-compile, then the number of instances of such container cannot change across post-build configurations.

For details on the post-build variant multiplicity and multiplicity configuration class of such containers, refer to the *Configuration interfaces* section for each driver in this User Manual.

1.7.5 **Variation point command line code generation**

1.7.5.1 **Importing and exporting configuration data**

The procedure to import or export variation point configuration data is as follows:

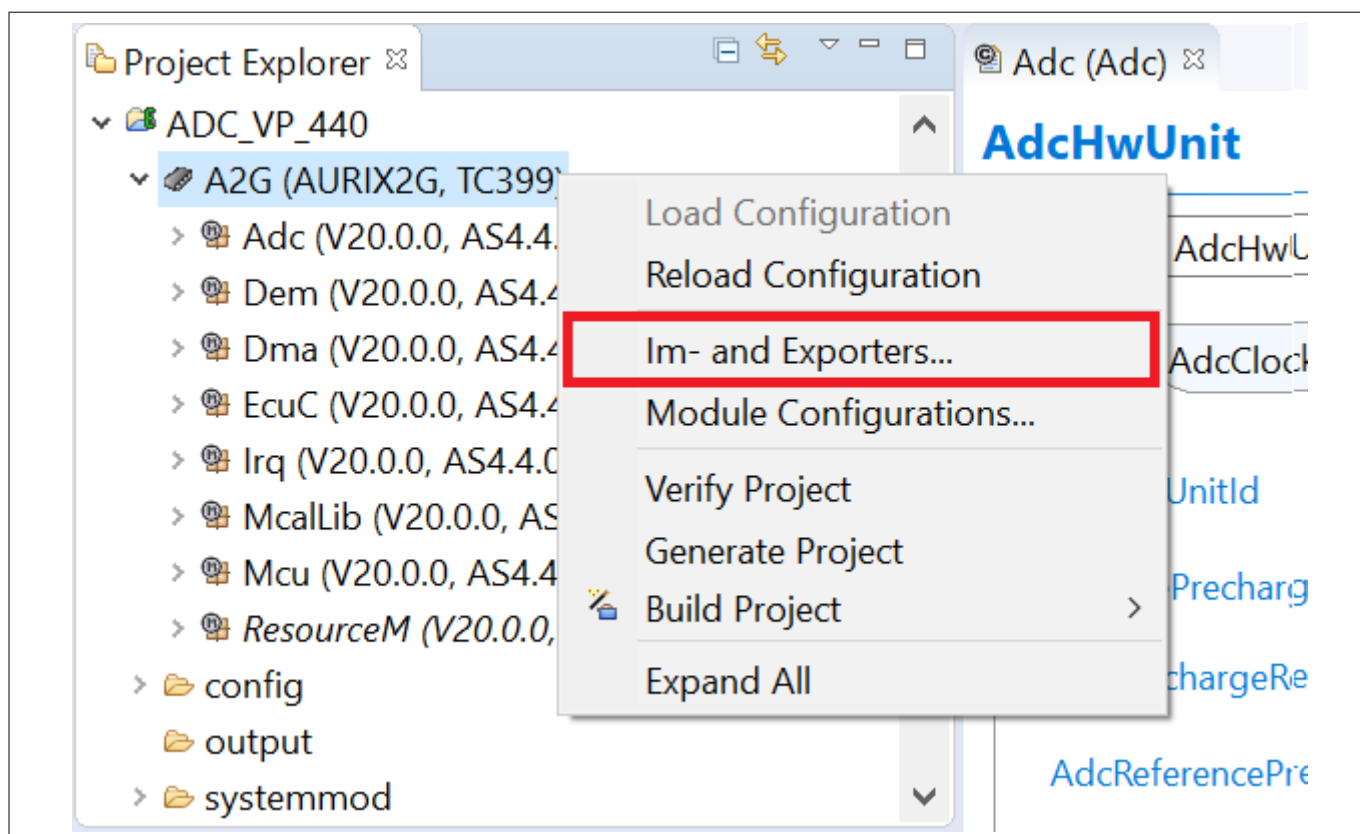


Figure 65 Creating AUTOSAR Im- and Exporter

1. To create new import or export configuration, right click on the ECU configuration node of the Project and select the **Im- and Exporters...** option.

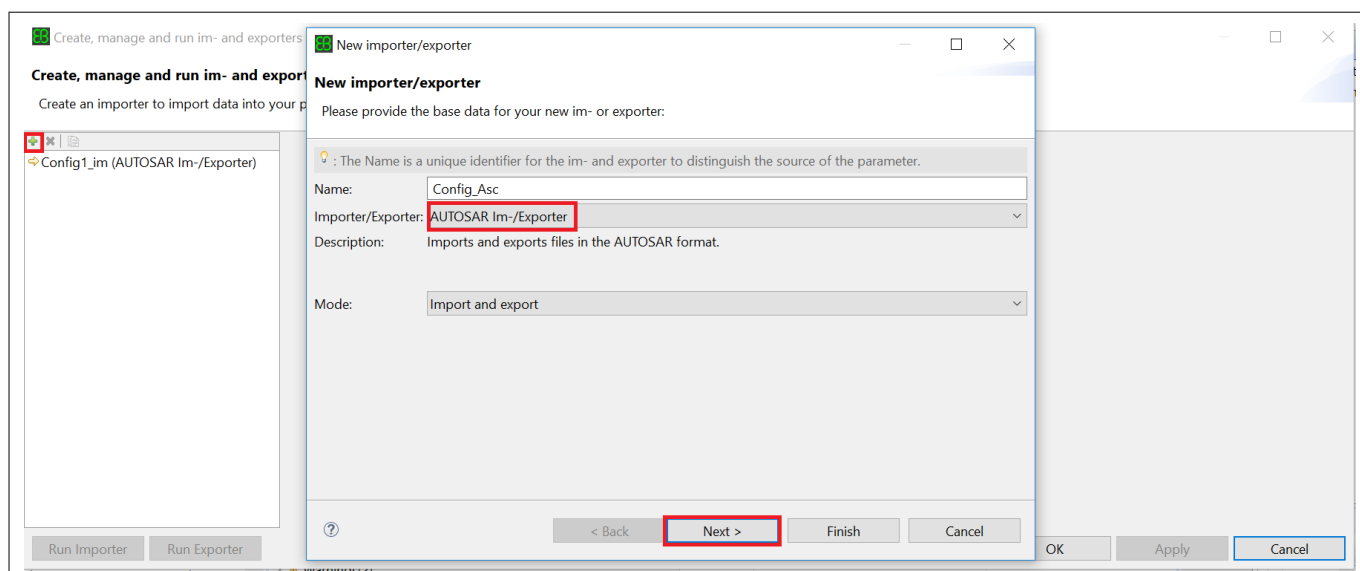


Figure 66 Selecting Importer/Exporter type

2. In the **New importer/exporter** section, select **AUTOSAR Im-/Exporter** option from the drop down menu and then click **Next**.

Generic information

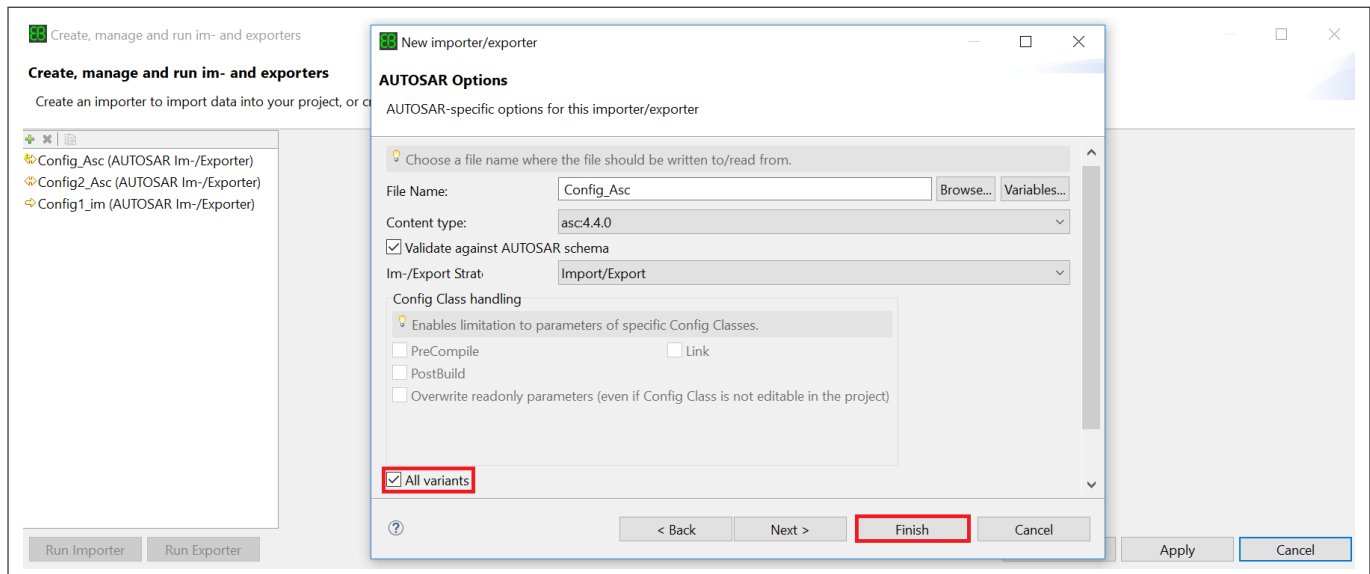


Figure 67 AUTOSAR options

3. In the **AUTOSAR Options** section, select the **All variants** checkbox and then click **Finish** to create the **AUTOSAR Im-/Exporter**.

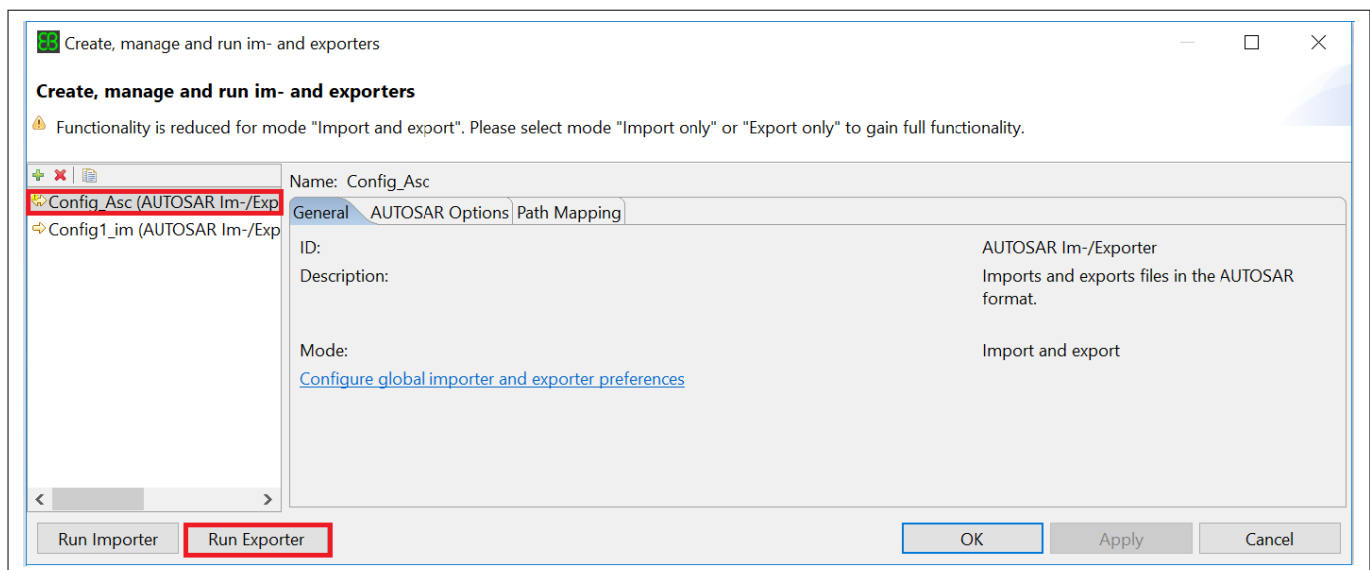
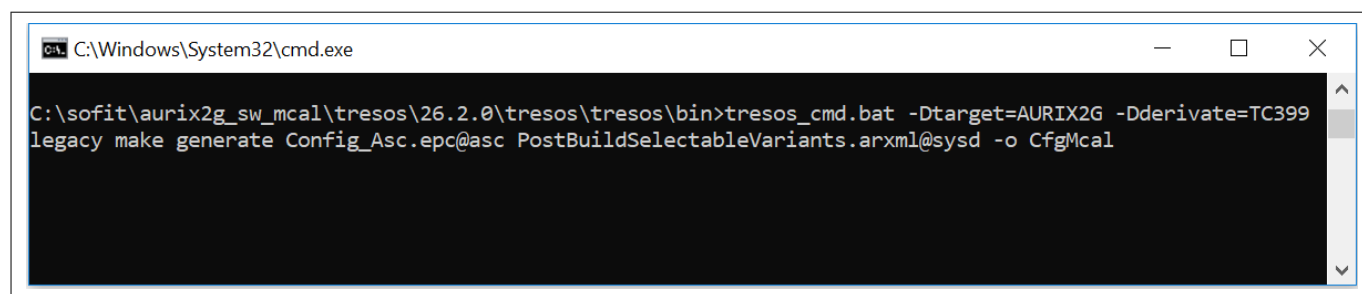


Figure 68 Exporting configuration data

4. Select the newly created **AUTOSAR Im-/Exporter** and click on **Run Exporter** to generate the **AUTOSAR module description** file and the **AUTOSAR system description** file.

1.7.5.2 Generating project via command line

Use the following command to generate the output files for all the variants via command line: `tresos_cmd.bat -Dtarget=AURIX2G -Dderivate=<DERIVATIVE> legacy generate <MODULE_DESCRIPTION_FILE>@asc <SYSTEM_DESCRIPTION_FILE>@sysd -o <OUTPUT_DIRECTORY>`



```
C:\Windows\System32\cmd.exe

C:\sofit\aurix2g_sw_mcal\tresos\26.2.0\tresos\tresos\bin>tresos_cmd.bat -Dtarget=AURIX2G -Dderivate=TC399
legacy make generate Config_Asc.epc@asc PostBuildSelectableVariants.arxml@sysd -o CfgMcal
```

Figure 69 **Command line code generation**

2 Safety view

2.1 Safety objectives

The 2nd generation of AURIX™ MCAL software is developed as a software safety element out of context (MCAL SEooC). For information on safety objectives realized by the MCAL SW SEooC and further information relevant for the integrator, refer to the respective release Safety Case Report.

2.2 Common Assumptions of Use (AoU)

The AoU that are common to all the productive MCAL drivers are described in this section.

- **Activate hardware mechanisms for safe SFR access**

The integrated system shall activate all hardware safety mechanisms which are related to register access.

Note: *All hardware safety mechanisms are documented in the Hardware Safety Manual (example for Bus Access Protection: SM[HW]:PORT:CFG_AS_AP).*

[cover parentID={1A65938D-1EC8-443e-9C36-43B5A30E0895}]

- **AoU described in the hardware safety manual**

The integrated system shall implement the hardware AoU as described in the Hardware Safety Manual.

[cover parentID={AB0ABD33-91E4-449c-9D99-84F8E5D36520}]

- **Safety Level partitioning**

Safety level partitioning is not supported in MCAL. The applications using a particular MCAL driver shall have the same safety levels, that is, applications having different safety levels shall not access the same MCAL driver.

Note: *Safety Level of the individual MCAL modules are captured in the release notes.*

[cover parentID={2E77E158-5321-4e20-A0D9-2AD517502D93}]

- **Ensure proper flash and initialization of memory**

The integrated system shall ensure proper flash and initialization of memory (code, callout code, configuration data, constants and variable initialization(e.g. start-up clear copy)) for successful execution of MCAL and memory initialization shall be developed and tested according to the highest ASIL of MCAL modules.

Note: *Linker option related to user initialized code/memory can be used by ensuring the proper working of the linker option and above AoU.*

[cover parentID={61898D6F-85B1-45d2-AFC6-81F863C669B8}]

- **Detect missing and unintended interrupts**

The integrated system shall implement mechanisms to detect the missing the CPU interrupts. The system shall also implement mechanisms to detect the missing and unexpected interrupts serviced by the DMA.

Note: *Recommendations for implementations are given in the hardware safety manual under ESM[SW]:IR:ISR_MONITOR.*

Note: *The MCAL drivers monitor the interrupt service requests routed to the CPU. The drivers report any unintended service requests to the application though safety error.*

[cover parentID={7F55BEAD-491E-49af-9CCB-7BE5D4791ED0}]

Safety view

- **DMA: Address CRC**

The integrated system shall compare the expected and the calculated DMA address checksum.

Note: *The DMA driver passes the checksum to the integrated system.*

Note: *MCAL drivers using the DMA driver do not support the address CRC feature.*

[cover parentID={8E5A971C-8D93-465a-BC13-90A16649F906}]

- **DMA: Data CRC**

The integrated system shall compare the expected and the calculated DMA data checksum.

Note: *The DMA driver provides the checksum to the integrated system.*

Note: *MCAL drivers using the DMA driver do not use the data CRC feature.*

[cover parentID={46118DC3-2851-4faa-B82D-580F87AA8DDD}]

- **DMA: Error handling**

The integrated system shall define appropriate actions in case of safety-related DMA errors.

Note: *The DMA driver detects the DMA error interrupt for transactions initiated through the DMA driver and notifies the user through the callback function.*

Note: *Safety-related DMA errors are the ones which have the potential to impact the safety goal.*

Note: *Refer to the hardware safety manual for recommendations on appropriate actions.*

[cover parentID={D0623C34-EB76-4dfd-BB86-C995FFE538C8}]

- **DMA: Timestamp**

The integrated system shall compare the expected and the reported DMA timestamp.

Note: *The DMA driver passes the timestamp to the integrated system.*

Note: *MCAL drivers using the DMA do not use the timestamp feature.*

[cover parentID={B270B32C-B467-4c91-80E0-6CB1A7BBB00F}]

- **E2E protection for communication**

The integrated system shall provide an end-to-end protection for all types of safe communication with the expected safety level of the data to be transferred.

Note: *This is affecting communication through CAN, CAN_TRCV, FR, LIN and ETH.*

Note: *Safety Level of the individual MCAL modules are captured in the release notes.*

[cover parentID={F6201C3E-653E-468c-AB27-C788BEA6CF6D}]

- **Enable hardware safety mechanisms for memory (Flash, RAM)**

The integrated system shall enable all hardware safety mechanisms for RAM and Flash.

Note: *All hardware safety mechanisms are documented in the Hardware Safety Manual (example for NVM: SM[HW]:NVM).*

[cover parentID={3E831224-4DEF-4b04-A977-A7FBB2B7EDB6}]

Safety view

- **Ensure safe data storage by mechanism outside SEooC**

The integrated system shall implement safety mechanisms for safe data storage in software layers outside the SEooC MCAL memory drivers with the expected safety level. Refer to Memory drivers.

Note: *CRC check for example, handled by NVRAM manager or SW-C is a measure to fulfill the safe data storage.*

Note: *MCAL memory drivers contain the following drivers: Internal Flash (FLS) driver, Flash EEPROM Emulation (FEE) driver. Note: Safety Level of the individual MCAL modules are captured in the release notes.*

[cover parentID={8437EB86-EF8E-47ca-B4D1-955BACD8EBAF}]

- **FFI in memory space for external services**

External services invoked by MCAL SEooC shall implement safety measures to ensure that there is no interference to memory space outside the specified one.

Note: *The external services accessed by MCAL software modules are documented in the module-specific sections of this document.*

[cover parentID={78C81004-95B3-49db-916B-A41008BE66BA}]

- **Generation of configuration data**

The integrator shall check and ensure the correctness of generated configuration data.

Note: *Code generation in the EB tresos tool is based on code templates.*

Note: *The correctness of the generated configuration data might be verified through system tests.*

[cover parentID={07C2AA6D-8779-4886-953A-12277B5BA2AE}]

- **Hardware resource access**

The integrated system shall not access the hardware resources, which are controlled and configured by the MCAL SEooC.

Note: *SFRs and memories are typical hardware resources.*

Note: *SMU SFRs are locked after initialization. Therefore, they cannot be modified during the runtime phase.*

[cover parentID={1DB1F6DA-4297-44c3-A023-B76087886A3B}]

- **Interference from software outside MCAL SEooC**

The system integrator shall implement measures to ensure no interference from the environment to MCAL SEooC data (software variables) and hardware resources (for example, SFRs) controlled by MCAL SEooC.

Note: *The measure could be that the environment is developed according to the same safety level as MCAL SEooC.*

Note: *This means that for example, MCAL SEooC will NOT implement mechanisms like redundant and diverse storage for safety-related global variables.*

Safety view

Note: *Potential integrator measures in this context are to define MPU partitions to protect RAM variables or SFR areas.*

Note: *Safety Level of the individual MCAL modules are captured in the release notes.*

[cover parentID={A46A519F-08F8-44f4-B4DD-C7B6844C16DF}]

- **Interrupt mapping**

The user shall ensure that hardware resources assigned to a core must have their interrupt service request routed to the same core.

[cover parentID={999CEDF3-9C31-483c-9AB9-603A917F7710}]

- **Invoked external services**

The integrated system shall ensure that all external services invoked by MCAL SEooC (for example, OS change user/supervisor mode) shall be compliant with at least the highest safety level supported by MCAL modules as mentioned in the release notes.

Note: *The external services accessed by MCAL software modules are documented in the module-specific sections of this document.*

Note: *Safety Level of the individual MCAL modules are captured in the release notes.*

[cover parentID={BC00B7DC-D8F9-4d4f-A555-0E631D607FE9}]

- **Program flow monitoring at system level**

The integrated system shall implement program flow monitoring at system level.

[cover parentID={93AA34FE-6F4A-40d4-A192-DC1A206EEDE3}]

- **Safety Switch and Init Check**

MCAL modules containing the safety switch (e.g. McuSafetyEnable) and the InitcheckAPI switch (e.g. McuInitCheckApi), need to have both switches enabled, in order to reach the specified safety claim. The invocation of the Init check function needs to be executed before any API other than the initialization function is used.

In case these aforementioned conditions are not met, the integrator is responsible to ensure that the conditions that trigger safety errors, production errors and runtime errors, as listed in the respective MCAL module User Manual, are not bringing the system to an unsafe state.

Note: *List of errors to be verified can be referred from Table under section Errors handling in the module specific user manual.*

[cover parentID={94ED57B0-5EE7-4bf6-A5B2-03FC141C017E}]

- **SMU alarms oscillator and clock monitor**

The integrated system shall deactivate the SMU alarm for oscillator and clock monitoring before calling the APIs for clock setup and activate the SMU alarm back afterwards.

Note: *The API sequence for clock setup is Mcu_InitClock(), Mcu_GetPllStatus() and Mcu_DistributePllClock().*

[cover parentID={9DFCACF4-75F2-4edd-B906-D132178A7F18}]

- **SMU alive alarm test**

The integrated system shall execute the "SMU Alive Test" provided by the SMU driver at least once per driving cycle and take the appropriate actions depending on the result.

Safety view

Note: Please refer to the hardware safety manual for recommendations on "appropriate actions".

[cover parentID={9D039BB8-CA5B-4537-9301-37BE53DF4E65}]

- **SMU in core domain and stand-by domain**

The integrated system shall enable the SMU Standby domain so that it can assert the configured reaction in response of incoming alarm signals.

[cover parentID={1DEF1963-B9E2-4493-BC9A-AAA614FE1AD1}]

- **SMU register monitoring test**

The integrated system shall take appropriate actions depending on the result of the register monitor test service provided by the SMU driver.

Note: Refer to the hardware safety manual for recommendations on appropriate actions.

[cover parentID={99A96D8A-F392-4c84-A20A-1B8005F35E32}]

2.3 Mapping of MCAL to external safety mechanism (ESM)

The following table lists the mapping of ESMs as provided in the *AURIX TC3xx Safety Manual* to the MCAL software.

ESM ID	Comments/implementation hints from MCAL
ESM[SW]:DMA:ERROR_HANDLING	<p>The DMA driver detects the DMA error interrupt for transactions initiated via the DMA driver and notifies the user via call back function.</p> <p>Note: MCAL module that uses DMA channel handles this error in respective module. Refer module specific User Manual for the more details.</p>
SMC[SW]:SMU:CONFIG	The SMU driver configures the SMU peripheral in core domain and standby domain.
ESM[SW]:SMU:APPLICATION_SW_ALARM	The SMU driver provides the Smu_SetAlarmStatus and Smu_ClearAlarmStatus APIs to set and clear alarms.
ESM[SW]:SMU:REG_MONITOR_TEST	The SMU driver provides the Smu_RegisterMonitor API to execute the register monitor test.
ESM[SW]:SMU:ALIVE_ALARM_TEST	The SMU driver provides the Smu_CoreAliveTest API to execute the core alive test.
SMC[SW]:FCE:CRC_CFG	The CRC driver configures the FCE channel and calculates the CRC for the data provided by the user. The comparison between calculated CRC value and the expected CRC value is not provisioned by MCAL.

Safety view

ESM ID	Comments/implementation hints from MCAL
ESM[SW]:IR:ISR_MONITOR	<p>The MCAL drivers monitor the interrupt service requests routed to CPU. The drivers report any unintended service requests to the application though safety error.</p> <p>Refer to Reporting of unintended service requests.</p> <p>Note: <i>Unintended service requests routed to DMA is not monitored by MCAL drivers.</i></p>
ESM[SW]:DMA:TIMESTAMP	<p>The time stamp feature of DMA is supported in the DMA driver. The application software shall use this feature while using the DMA driver.</p> <p>Note: <i>MCAL drivers using the DMA do not use the timestamp feature.</i></p>
ESM[SW]:DMA:DATA_CRC	<p>The data CRC feature of DMA is supported in the DMA driver. The application software shall use this feature while using the DMA driver.</p> <p>Note: <i>MCAL drivers using the DMA driver do not use the data CRC feature.</i></p>
ESM[SW]:DMA:ADDRESS_CRC	<p>The address CRC feature of DMA is supported in the DMA driver. The application software shall use this feature while using the DMA driver.</p> <p>Note: <i>MCAL drivers using the DMA driver do not support the address CRC feature.</i></p>
ESM[SW]:CONVCTRL:CONFIG_CHECK	<p>The MCU driver performs a read back of CONVCTRL_PHSCFG register after programming it in the API Mcu_InitClock(). If the read value differs from the expected value then a safety error is reported. The reported safety error shall be addressed by the user software.</p>
ESM[SW]:EVADC:CONFIG_CHECK	<p>The ADC driver provides initialization and initialization check APIs. The initialization API programs the GxANCFG and GxSYNCTR SFRs. The initialization check reads back and verifies the content of GxANCFG and GxSYNCTR for correctness. User shall use these services to detect a failure in programming of GxANCFG or GxSYNCTR. A return value of E_NOT_OK from initialization check API indicates an error and shall be addressed by the user software.</p>

Note: *AoU from AURIX TC3xx Safety Manual, other than the ones mentioned in the above table are not provisioned by MCAL and hence, requires evaluation of the integrator.*

2.3.1 Reporting of unintended service requests

Scenario for the ADC, DSADC and MCU drivers

The ADC, DSADC and MCU drivers may report an unintended service request safety error when a second hardware interrupt occurs in between the start of an ISR and clearing of the interrupt flag. **The safety error reported in such a scenario must be ignored by the user.** The following figure describes the scenario in detail.

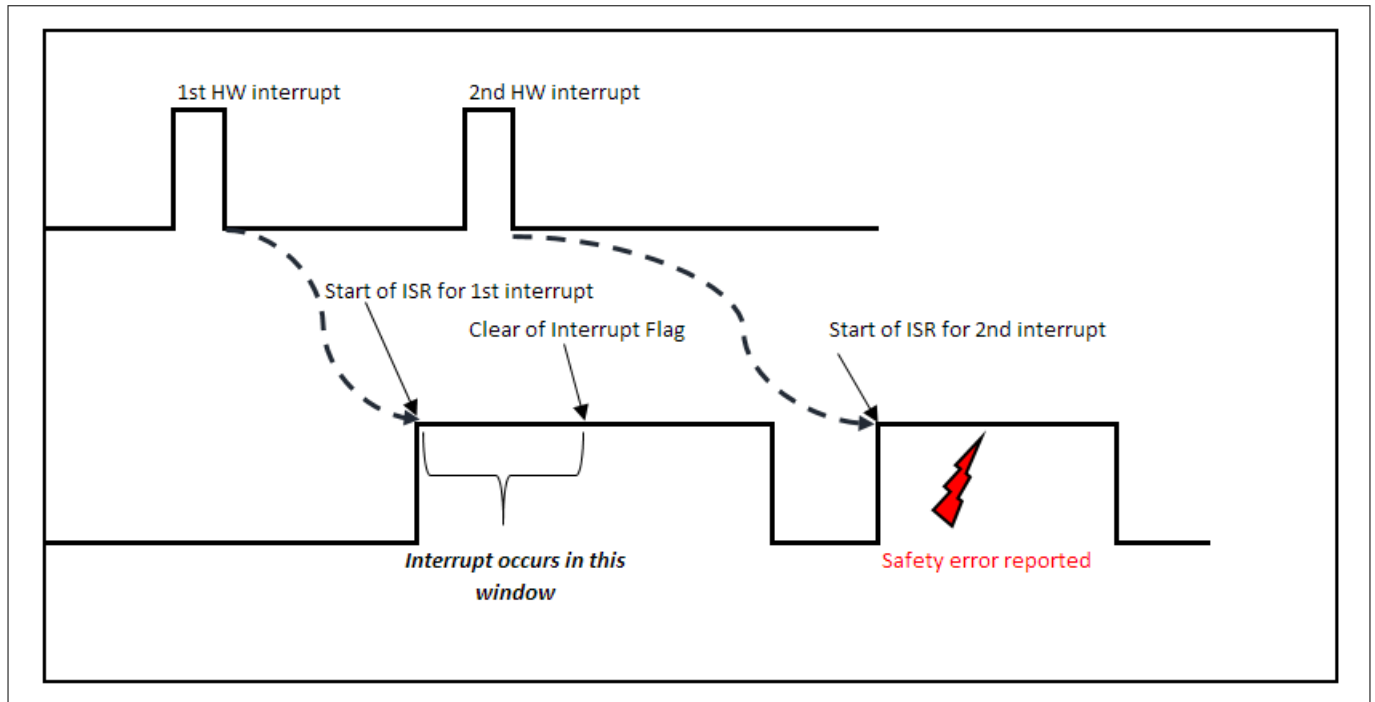


Figure 70 Incorrect reporting of unintended service request - 1

Note: This scenario typically happens when the interrupts occur at a rate higher than the service rate.

Scenario for the PWM and MCU drivers

The MCU driver may report an unintended service request safety error when the PWM APIs are invoked in any of the following scenarios and an interrupt for the same channel occurs in a small window as shown in the following figure:

- Pwm_17_GtmCcu6_SetDutyCycle() is invoked to set the duty cycle as 0% or 100% for a channel
- Pwm_17_GtmCcu6_SetPeriodAndDuty() is invoked to set period as 0 or duty as 0% or duty as 100% for a channel
- Pwm_17_GtmCcu6_EnableNotification() is invoked for a variable period channel for which the period is currently set to 0
- Pwm_17_GtmCcu6_DisableNotification() is invoked

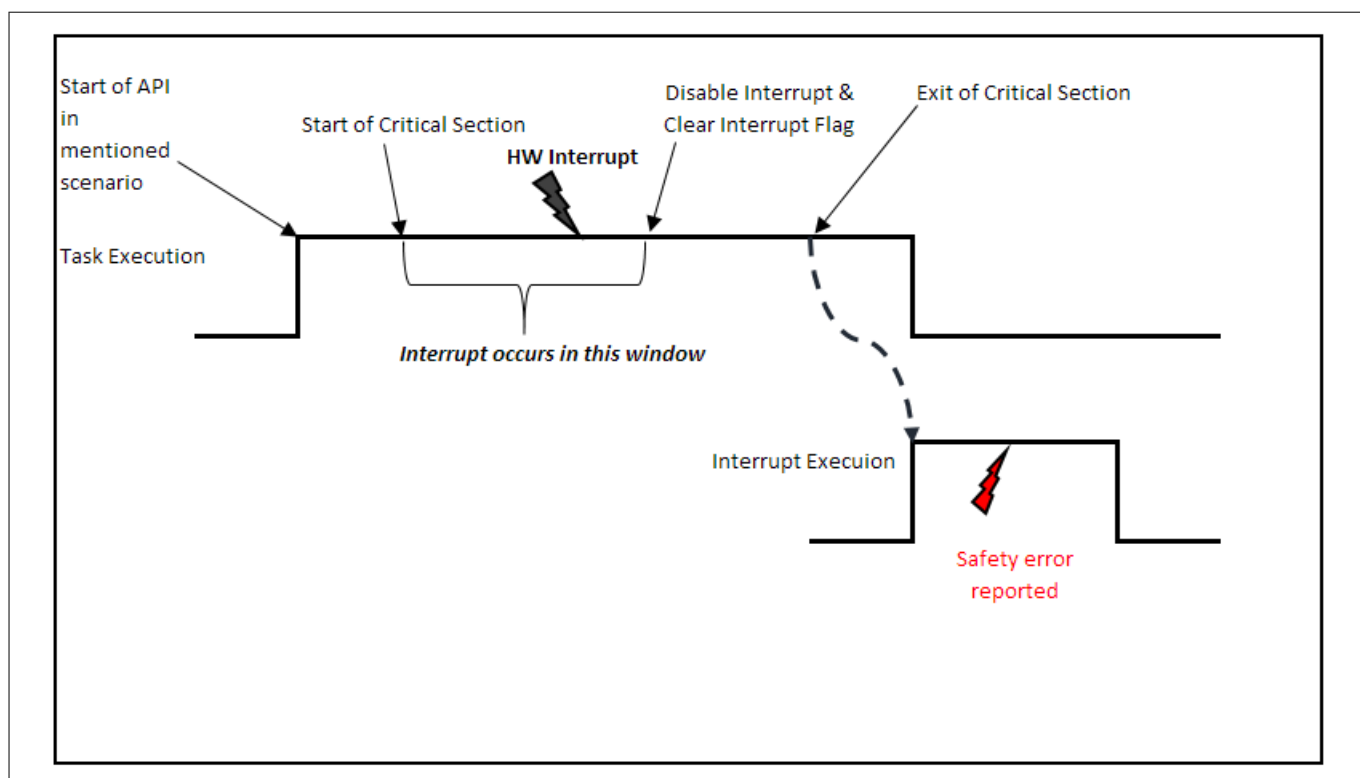


Figure 71 **Incorrect reporting of unintended service request - 2**

Scenario for the DSADC driver

The DSADC driver may report an unintended service request safety error when the DSADC channel is configured as DSADC_SINGLE_READ and the Dsadc_ReadResult API is invoked and an interrupt for the same channel occurs in a small window as shown in the following figure:

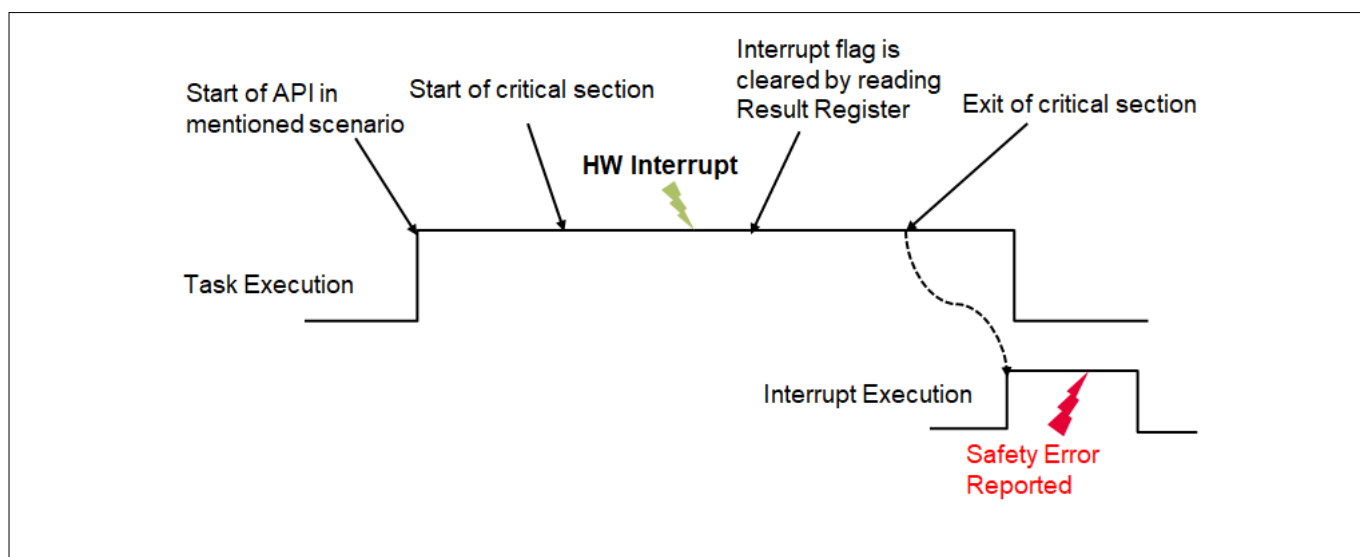


Figure 72 **Incorrect reporting of unintended service request - 3**

The safety error reported in the above scenarios must be ignored by the user.

Deviations and limitations

3 Deviations and limitations

This section describes the deviations and limitations which are common to all the MCAL modules.

3.1 Software specification deviations

Table 10 Known deviations

Reference	Deviation
SWS_COMPILER_00010, AUTOSAR_SWS_CompilerAbstraction.	The MCAL has defined the value behind the compiler symbols provided in Compiler.h file.

3.2 Limitations

This section describes the limitations of Commonly applicable to MCAL Modules.

Table 11 Known Limitations

Reference	Limitation
Port P21_LPCR2.PS LVDS functionality support for TC35x, TC33x Ext Devices.	<p>The PORT bit P21_LPCR2.PS controls functionalities of enabling LVDS and impacts XTAL gain for devices that have P21 LVDS support. For radar devices that do not have P21 LVDS support (TC35x, TC33x EXT), this bit impacts only the XTAL gain.</p> <p>Setting P21_LPCR2.PS to 0 selects for a 3.3V Pad Supply while setting P21_LPCR2.PS to 1 selects for a 5V Pad Supply.</p> <p>Note: Overall XTAL gain should be adjusted to the needs of the connected external crystal / resonator in conjunction with the used capacitor configuration and OSCCON.GAINSEL settings. Settings must be confirmed with Crystal supplier.</p> <p>Access to PORT P21_LPCR2.PS bit shall be done by the integrator before the MCAL Mcu_Init API is called.</p> <p>The MCAL Port configuration setting for P21 should be in compliance with the P21_LPCR2.PS bit set by the customer as Port_Init API accesses the P21_LPCR2 register.</p> <p>Unexpected behaviours may occur if conflicting values are written to the register by the integrator before Mcu_Init is invoked and when Port_Init is called.</p> <p>e.g. P21_LPCR2.PS bit set to "1" to indicate a Pad Supply of 5V through register access and LVDS is disabled in Tresos PORT configuration leading Port_Init to write "0" to P21_LPCR2 in Port_Init API.</p>

Revision history

Revision history

Major changes since the last version

Date	Version	Description
2024-08-01	18.0	Document is released.
2024-07-23	17.1	<ul style="list-style-type: none"> Updated AoU 'Code and configuration data correctly flashed' to 'Ensure proper flash and initialization of memory' under 2.2 Common Assumptions of Use (AoU) section. Updated Note for AOu 'Generation of configuration data' under 2.2 Common Assumptions of Use (AoU) section. Added Limitation 'Port P21_LPCR2.PS LVDS functionality support for TC35x, TC33x Ext Devices' under section 3.2 Limitations. Updated latest version of "MC-ISAR_TC3xx_HWErrataAnalysis.xlsx" in Reference Documents.
2023-07-24	17.0	Document is released.
2023-07-24	16.1	Section 2.1 Safety Objectives updated to refer information relevant for the integrator from the Safety Case Report.
2023-07-21	16.0	Document is released.
2023-07-19	15.1	<ul style="list-style-type: none"> 2 Safety view <ul style="list-style-type: none"> Section 2.1 Safety objectives is updated to refer Safety Case Report document for Safety objectives. Removed sections 2.1.1 Safety objective 1 and 2.1.2 Safety objective 2.
2023-07-06	15.0	Document is released.
2023-07-05	14.1	<ul style="list-style-type: none"> 2.2 Common Assumptions of Use(AoU) section <ul style="list-style-type: none"> Added Safety Switch and Init check AoU added.
2023-07-04	14.0	Document is released.
2023-07-04	13.3	Chapter <u>1.3.5.2</u> , Configuration containers and parameters section added container ResourceMNumberOfActiveCores.
2023-06-30	13.2	<ul style="list-style-type: none"> 2.2 Common Assumptions of Use(AoU) section <ul style="list-style-type: none"> Description updated for E2E protection for communication, ensure safe data and storage by mechanism outside SEooC, Invoked external services and Interference from software outside MCAL SEooC AoUs. ASILpartitioning changed to Safety Level partitioning and updated the description.
2023-05-25	13.1	<ul style="list-style-type: none"> Chapter <u>2.1.1.1.1</u>, description updated and added Wdg, Spi, and Dma to the modules list having ASIL D initialization at system level. 'Hardware resource access' under section 2.2 is updated to remove "documented in the module-specific sections of this document" since the module specific sections are moved to module specific UMs.
2021-11-18	13.0	Document is released.
2021-11-17	12.1	<ul style="list-style-type: none"> Note enhanced for the AoU 'Generation of configuration data'.
2021-11-02	12.0	Document is released.

Revision history

Date	Version	Description
2021-10-28	11.1	<ul style="list-style-type: none"> Note added in the section Creating an EB tresos project. New section named MCAL Initialization added. New section named Deviations and limitations added. Note updated in the section Creating an EB tresos project. Updated the section Resource Manager.
2021-03-26	11.0	Document is released.
2021-03-23	10.1	<ul style="list-style-type: none"> Note added on usage of platform files. Memory mapping section updated.
2020-12-08	10.0	Document is released.
2020-12-08	9.1	<ul style="list-style-type: none"> Optimization technique for supervisor mode registers access section added Creating an EB tresos project section updated to describe selection of Autosar Release version "Mixed " Grammatical and textual corrections done (Non-functional changes) Module details table of multicore support section updated Safety realization section updated Building and linking to generate final application binary section updated
2020-08-14	9.0	Document is released.
2020-08-07	8.1	<ul style="list-style-type: none"> The document is renamed from "MCAL User Manual for BASIC package" to "MCAL User Manual General". Document aligned with MCAL User Manual for BASIC package, Version 1.40.0_13.0. The driver module chapters are moved to separate module specific user manuals document. Generic information and Safety view section updated
2019-08-05	8.0	Update the images and device information in the chapter Generic information.
2019-07-26	7.0	<ul style="list-style-type: none"> Can_17_McmCan <ul style="list-style-type: none"> Description for the Can_17_McmCan_Main_Function_BusOff API is updated Description for the CanControllerId configuration parameter is updated

Revision history

Date	Version	Description
2019-07-23	6.0	<ul style="list-style-type: none"> Hardware-software mapping and Integration hints are updated for all drivers. Adc <ul style="list-style-type: none"> Configuration parameters for BWD support is added. Configuration parameter for alias support is updated. Can_17_McmCan <ul style="list-style-type: none"> Description for the Can_17_McmCan_Main_Function_Read and Can_17_McmCan_IsrRxFIFOHandler APIs is updated. Description for the CanControllerTrcvDelayCompensationOffset configuration parameter is updated. Limitations and deviations section is updated. Key architectural considerations section is updated. CanTrcv_17_W9255 <ul style="list-style-type: none"> Integration hints section is updated. Critical section WakeFlagUpdate is removed. Integration hints section for the MCU support is updated. Dio <ul style="list-style-type: none"> AoU for Dio_FlipChannel API is added. Fee <ul style="list-style-type: none"> Example usage section is updated. Fls_17_Dmu <ul style="list-style-type: none"> Safety error names are updated. Limitations and deviations section is updated. Gpt <ul style="list-style-type: none"> Integration hints section is updated for SchM. Icu_17_Timerlp <ul style="list-style-type: none"> The IcuIncrementalInterfaceApi configuration parameter is made editable to enable the user to use incremental interface API. Example usage for incremental interface mode is added. Configuration parameter IcuMaxChannel,GPT12DirPortSelection,GPT12DirPortSelection, GPT12InputPortSelection, TimChannelPortPinSelect, ErulInputPin,GPT12CounterType,GPT12BlockReference ,TimChannelInputSelect and CCChannelInputSelection are updated. Description for the APIs Icu_17_Timerlp_StartIncInterface, Icu_17_Timerlp_StopIncInterface, Icu_17_Timerlp_ReadEncCount and Icu_17_Timerlp_ReadEncCountDir is updated Description for the data type Icu_17_Timerlp_ChannelType is updated. Mcallib <ul style="list-style-type: none"> AoU for the usage of the Mcal_WriteSafetyEndInitProtReg API is added. Mcu <ul style="list-style-type: none"> User hints in the Mcu_InitRamSection and Mcu_SetMode APIs are updated. AoUs for SMU alarm and sequence of Timerlp APIs is added. The McuExtClock1Div configuration parameter is renamed to McuFoutClockDiv. Description for the McuClockReferencePointFrequency2 and McuAdasFrequency configuration parameters is updated. Key architectural considerations section is updated. Ocu <ul style="list-style-type: none"> Key architectural considerations section is updated.

Revision history

Date	Version	Description
		<ul style="list-style-type: none"> - AoU section updated. - Limitations and deviations section updated. • Port <ul style="list-style-type: none"> - Limitations and deviations section updated. • Pwm_17_GtmCcu6 <ul style="list-style-type: none"> - AoU section is updated. - Datatype Pwm_17_GtmCcu6_ChannelType is updated. - DSADC notification-related information is added. - Error table is updated. • Spi <ul style="list-style-type: none"> - Limitations and deviations section is updated.
2019-04-22	5.0	Added support for the TC37xA and TC37xA_ED devices.
2019-04-11	4.0	<ul style="list-style-type: none"> • Added support for the TC35xA device. • Added the CanTrcv_17_V9251, CanTrcv_17_W9255 and OCU drivers.
2019-02-06	3.0	<p>In the <i>Generic information</i> chapter, added the <i>Post-build variant multiplicity and Multiplicity configuration class</i> section.</p> <p>Updated the <i>Configuration interfaces</i> and <i>Deviations</i> sections for the SPI driver.</p>
2019-02-04	2.0	<ul style="list-style-type: none"> • <i>Integration hints</i> and <i>Reference information</i> for all modules updated. • Module-specific AoUs updated. • Added the EB tesos: license handling section. • Added AoUs in the Common Assumptions of Use (AoUs) section. • Updated the ESM mapping in the Mapping of MCAL to external safety mechanism section.
2018-10-12	1.0	Initial version.

Trademarks

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

Edition 2024-08-01

Published by

Infineon Technologies AG
81726 Munich, Germany

© 2024 Infineon Technologies AG
All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-vjw1559807504045

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenhheitsgarantie").

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

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

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