MCU 3.0 driver user guide

TRAVEO™ T2G family

About this document

Scope and purpose

This guide describes the architecture, configuration, and usage of the MCU driver. This guide also explains the functionality of the driver and provides a reference to the driver's API.

The installation, build process, and general information about the use of the EB tresos Studio are not within the scope of this document. See the *EB tresos Studio for ACG8 user's guide* [8] for detailed information on these topics.

Intended audience

This document is intended for anyone who uses the MCU driver of the TRAVEO™ T2G family.

Document structure

Chapter 1 General overview gives a brief introduction to the MCU driver, explains the embedding of the driver in the AUTOSAR environment and describes the supported hardware and development environment.

Chapter 2 Using the MCU driver provides detailed steps required to use the MCU driver in the application.

Chapter 3 Structure and dependencies describes the file structure and the dependencies for the MCU driver.

Chapter 4 EB tresos Studio configuration interface describes the driver's configuration with the EB tresos Studio software.

Chapter 5 Functional description gives a functional description of all services offered by the MCU driver.

Chapter 6 Hardware resources describes the hardware resources used by the driver.

The Appendix A and Appendix B provides the complete API reference and access register table.

Abbreviations and definitions

Table 1 Abbreviation

Abbreviation	Description
AHB	Advanced High-performance Bus
ALTHF	Alternate High-frequency clock
ALTLF	Alternate Low-frequency clock
ASIL	Automotive Safety Integrity Level
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software. Standardized part of software which does not fulfill a vehicle functional job.
BOD	Brown-out Detection

Abbreviation	Description
CCO	Current Controlled Oscillator
CM0+	Cortex®-M0+ processor
CM4	Cortex®-M4 processor
CM7	Cortex®-M7 processor
DEM	Diagnostic Event Manager
DET	Default Error Tracer
DMA	Direct Memory Access
DSI	Digital System Interconnect
EB tresos ECU AUTOSAR Suite	A collection of AUTOSAR Basic Software modules and a Runtime Environment integrated in a common configuration and build environment.
EB tresos Studio	Elektrobit Automotive configuration framework
ECO	External Crystal Oscillator
FLL	Frequency Locked Loop
HF clock	High-frequency clock
HVLVD	High Voltage / Low Voltage Detector
ILO	Internal Low-speed Oscillator
ISR	Interrupt Service Routine
LF clock	Low-frequency clock
LPECO	Low-power External Crystal Oscillator
LVD	Low Voltage Detector
IMO	Internal Main Oscillator
MCAL	Microcontroller Abstraction Layer
MCU	Microcontroller Unit
OCD	Over-current Detection
OS	Operating System
OVD	Over-voltage Detection
PCLK	Programmable Clock
PFD	Phase Frequency Detector
PLL	Phase Locked Loop
PMIC	Power Management Integrated Circuit
RAM	Random Access Memory
REGHC	High-current Regulator
ROM	Read-Only Memory
RTC	Real-Time Clock
SSCG	Spread Spectrum Clock Generator
VADJ	Voltage Adjustment
WCO	Watch Crystal Oscillator
WDT	Watchdog Timer

Abbreviation	Description
WFI	Arm® Wait For Interrupt instruction
µC	Microcontroller

Related documents

AUTOSAR requirements and specifications

Bibliography

[1] General specification of basic software modules, AUTOSAR release 4.2.2

[2] AUTOSAR specification of MCU driver, release 4.2.2.

[3] AUTOSAR specification of standard types, release 4.2.2.

[4] Specification of ECU configuration parameters, AUTOSAR release 4.2.2.

[5] AUTOSAR specification of default error tracer, release 4.2.2.

[6] AUTOSAR specification of diagnostics event manager, release 4.2.2.

[7] Specification of memory mapping, AUTOSAR release 4.2.2.

Elektrobit automotive documentation

Bibliography

[8] EB tresos Studio for ACG8 user's guide.

Hardware documentation

The hardware documents are listed in the delivery notes.

Related standards and norms

Bibliography

[9] AUTOSAR layered software architecture, release 4.2 revision 2.

Table of contents

1 General overview

1.1 Introduction to the MCU driver

The MCU driver is a set of software routines to initialize the MCU, and provides configuration options for the following:

- Clock settings
- Low-power modes
- RAM section initialization

The driver is compliant with the AUTOSAR standard and is implemented according to *AUTOSAR specification of MCU driver* [2].

In addition, the MCU driver is delivered with a plugin for the EB tresos Studio software, which allows you to statically configure the driver options. The driver also provides an interface to define symbolic names and the functionality of all configuration options.

1.2 User profile

This guide is intended for users with a basic knowledge of the following domains:

- Embedded systems
- C programming language
- The AUTOSAR standard
- The target hardware architecture

1.3 Embedding in the AUTOSAR environment

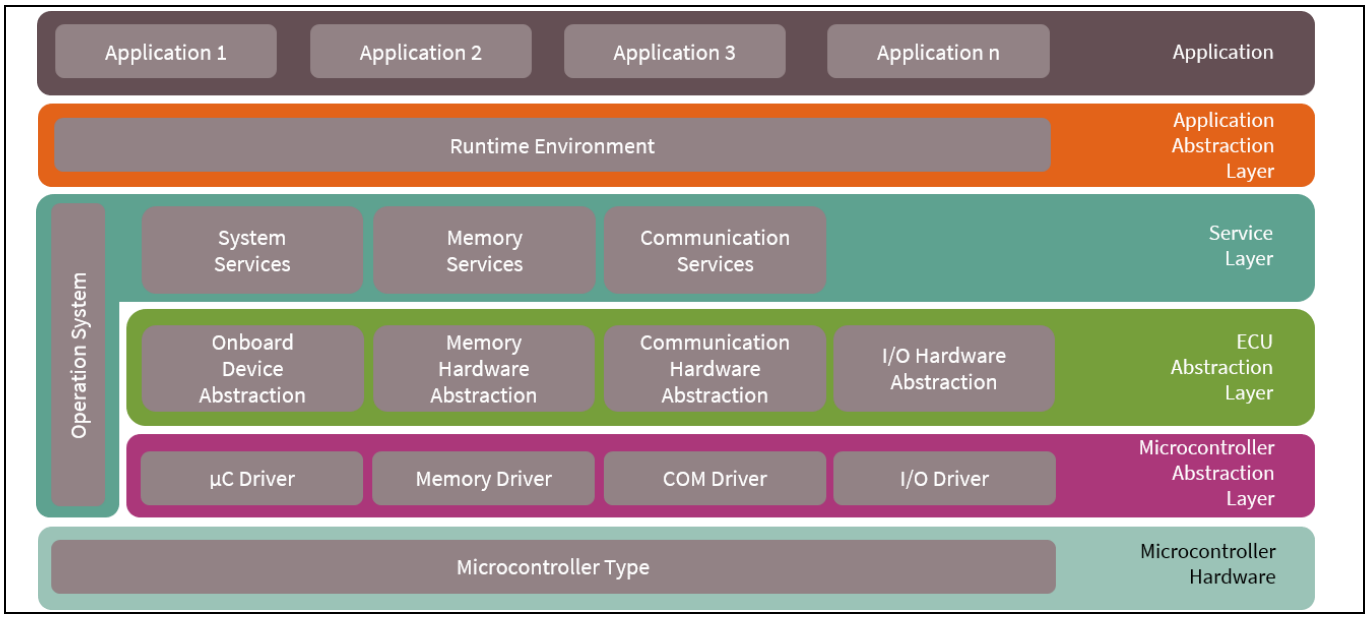

Figure 1 Overview of AUTOSAR software layers

Figure 1 shows the layered AUTOSAR software architecture. The MCU driver (Figure 2) is a part of the microcontroller abstraction layer (MCAL), the lowest layer of basic software in the AUTOSAR environment.

As an internal I/O driver, the MCU driver provides a standardized and µC-independent interface to higher software layers for accessing clocks and CPU modes of the ECU hardware.

For an overview of the AUTOSAR layered software architecture, see *AUTOSAR layered software architecture* [9].

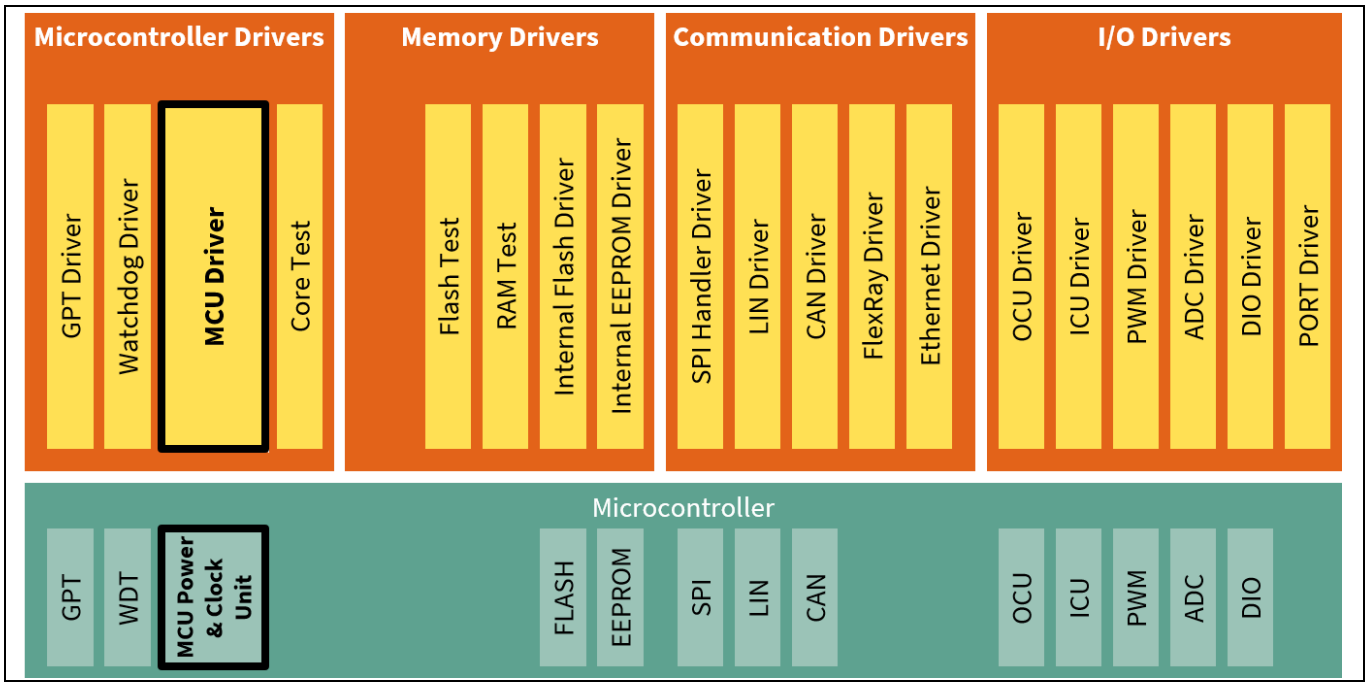

Figure 2 MCU driver in MCAL layer

1.4 Supported hardware

This version of the MCU driver supports the TRAVEO™ T2G microcontroller family. The supported derivatives are listed in the release notes.

Additional derivatives that contain only a subset of the capabilities of one derivative mentioned above can be implemented and supported by providing a resource file with its properties.

1.5 Development environment

The development environment corresponds to AUTOSAR release 4.2.2. The Base, Make, and Resource modules are required for the proper functionality of the MCU driver.

1.6 Character set and encoding

All source code files of the MCU driver are restricted to the ASCII character set. The files are encoded in UTF-8 format, with only the 7-bit subset (values 0x00 … 0x7F) being used.

1.7 Multicore support

The MCU driver supports multicore type II. The multicore type III can also be supported for some APIs (for example, read-only API or atomic-write API). For each multicore type, see the following sections:

Note: If multicore type III is required, the section including the data related to the read-only API or atomic write API must be allocated to the memory, and can be read from any cores.

1.7.1 Multicore type

In this section, type I, type II, and type III are defined as multicore characteristics.

1.7.1.1 Single core only (multicore type I)

For this multicore type, the driver is available only on a single core. This type is referred as “Multicore Type I”.

Multicore type I has the following characteristics:

- The peripheral channels are accessed by only one core.

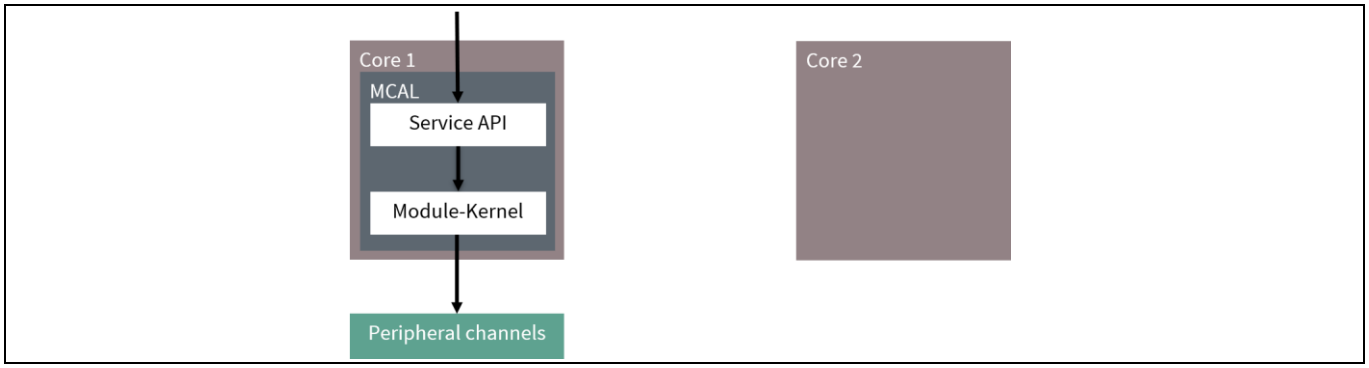

Figure 3 Overview of the multicore type I

1.7.1.2 Core-dependent instances (multicore type II)

For this multicore type, the driver has core-dependent instances with individually allocable hardware. This type is referred as “Multicore Type II”.

Multicore type II has the following characteristics:

- The driver code is shared among all cores.
  - A common binary is used for all cores.
  - A configuration is common for all cores.
- Each core runs an instance of the driver.
- Peripheral channels and their data can be individually allocated to cores but cannot be shared among cores.
- One core will be the master; the master core must be initialized first.
  - Cores other than the master core are called satellite cores.

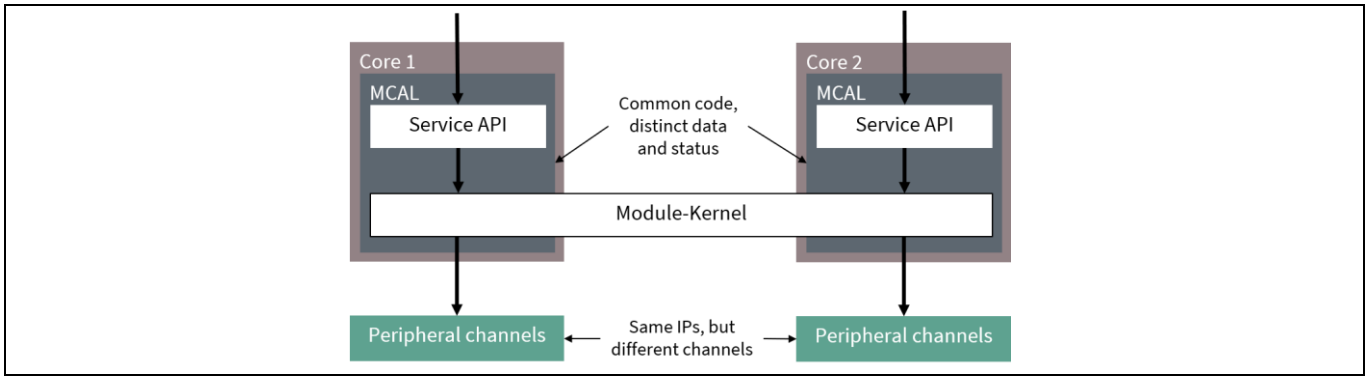

Figure 4 Overview of the multicore type II

1.7.1.3 Core-independent instances (multicore type III)

For this multicore type, the driver has core-independent instances with globally available hardware. This type is referred as “Multicore Type III”.

Multicore type III has the following characteristics:

- The code of the driver is shared among all cores.
  - A common binary is used for all cores.
  - A configuration is common for all cores.
- Each core runs an instance of the driver.
- Peripheral channels are globally available for all cores.

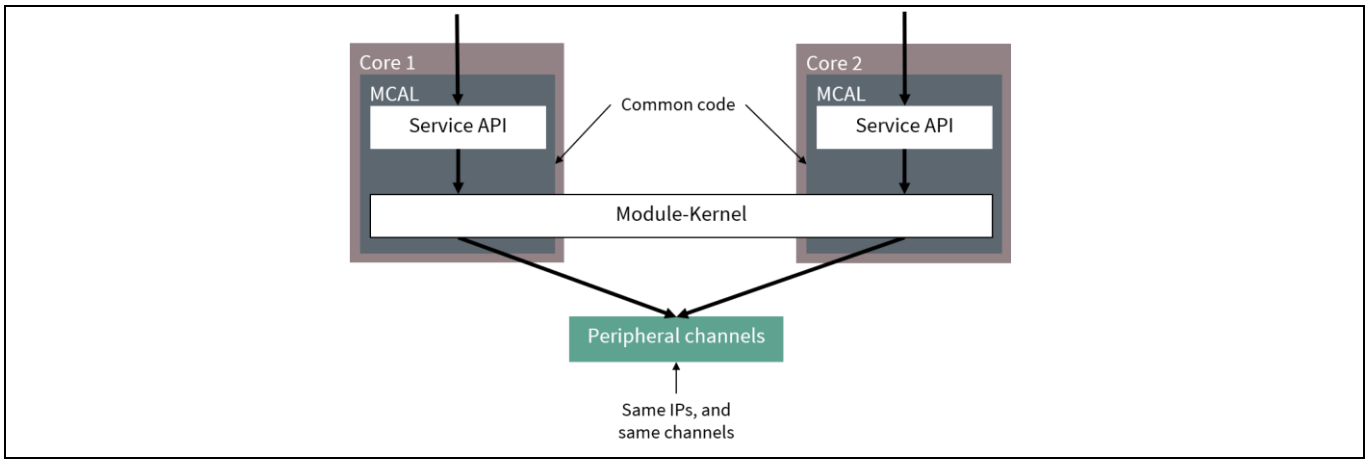

Figure 5 Overview of the multicore type III

1.7.2 Virtual core support

The MCU driver supports any number of cores. The configured cores need not be equal to the physical cores.

The MCU driver calls a configurable callout function (`McuGetCoreIdFunction`) to identify the core that is currently executing the code. This function can be implemented in the integration scope. The function can be written such that it does not return the physical core, but instead returns the SW partition ID, OS application ID, or any attribute/parameter. By interpreting these as the core, the MCU driver can support multiple SW partitions on a single physical core.

2 Using the MCU driver

This chapter describes all necessary steps to incorporate the MCU driver into your application.

2.1 Installation and prerequisites

Note: Before you start, see the EB tresos Studio for ACG8 user's guide [8] for the following information:

1. *The installation procedure of EB tresos ECU AUTOSAR components.*
2. *The usage of the EB tresos Studio.*
3. *The usage of the EB tresos ECU AUTOSAR build environment (It includes an explanation of how to set up and integrate the own application within the EB tresos ECU AUTOSAR build environment).*

The installation of the MCU driver complies with the general installation procedure for EB tresos ECU AUTOSAR components given in the documents mentioned above. If the driver has been successfully installed, the driver will appear in the module list of the EB tresos Studio (see *EB tresos Studio for ACG8 user's guide* [8]).

This document assumes that the project is properly set up and is using the application template as described in the *EB tresos Studio for ACG8 user's guide* [8]. This template provides the necessary folder structure, project, and makefiles needed to configure and compile an application within the build environment. You need to be familiar with the usage of the command line shell.

2.2 Configuring the MCU driver

This section provides a brief overview of the configuration structure defined by AUTOSAR to use the MCU driver.

Use the following containers to configure the common behavior:

- `McuGeneralConfiguration`: Restrict/extend the API of the MCU module and enable/disable default error trace (DET).
- `McuModuleConfiguration`: Configure the clock, RAM initialization, and low-power modes of the MCU module.
- `McuPublishedInformation`: Holds the value of the cause of reset supported in the MCU.

See chapter 4 EB tresos Studio configuration interface.

Note: Ensure that the application also includes an AUTOSAR-compliant DET when default error detection is enabled; otherwise, the application will not compile.

You must set up the following characteristics for each MCU configuration:

- Clock configuration
- Number of RAM sectors
- RAM sector configuration
- Number of low-power modes
- Low-power mode configuration

The `McuGeneralConfiguration` container describes the individual MCU setup information.

2.2.1 Architecture specifics

- `McuSafetyFunctionApi`: Adds or removes the `Mcu_CheckClockStatus()` and `Mcu_CheckModeStatus()` services from the code.
- `McuErrorCalloutFunction`: Specifies the error callout handler that is called when errors are detected during runtime.
- `McuEnableGetCoreIDApi`: Adds or removes the `Mcu_GetCoreID()` service from the code.
- `McuEnableSetModeApiOnly`: Enables only the `Mcu_Init()`, `Mcu_SetMode()`, `Mcu_CheckClockStatus()`, and `Mcu_CheckModeStatus()` services from the code.
- `McuIncludeFile`: Specifies the file name to include definitions such as declaration for error callout handler.
- `McuModuleConfiguration`: Contains architecture-specific parameters. See section 4.2 MCU module configuration.
- `McuMulticore`: Contains multicore-specific parameters. See section 4.8 MCU multicore.

2.3 Adapting an application

To use the MCU driver in an application, do the following:

Step 1: Include the MCU driver header file by adding the following line of code to the source file:

```
#include "Mcu.h" /* MCU Driver */
```

This publishes all needed function, data prototypes, and symbolic names of the configuration to the application.

Step 2: Implement the error callout function for ASIL safety extension.

To do this, declare the error callout function in the file specified by `McuIncludeFile` and implement it in your application (see section 7.4 Required callback functions, Error callout API).

The error callout function name can be configured by the `McuErrorCalloutFunction` parameter.

Step 3: Initialize and configure the MCU.

See chapter 4 EB tresos Studio configuration interface. The MCU module will automatically be enabled if an appropriate parameter configuration of the MCU module is available in the application.

The MCU initialization can be done with the following function call and parameter. This API must be called on the all cores.

```
Mcu_Init(&Mcu_Config[0]);
```

The master core must be initialized prior to the satellite core. All cores must be initialized with the same configuration.

As part of the initialization process, call the `Mcu_InitClock` API.

The following is a short example for a clock “MY_CLOCK” configured as a clock setting and for a mode “MY_MODE” configured as a mode setting:

```
Mcu_InitClock(McuConf_McuClockSettingConfig_MY_CLOCK);
```

An additional call of the next API function might be needed (depending on the underlying hardware) to set up the clock properly:

```
Mcu_DistributePllClock();
```

`Mcu_InitClock` and `Mcu_DistributePllClock` APIs must be called on the core that MY_CLOCK is allocated.

If RAM sectors are configured, they need to be initialized with a separate call of the API function, stated below, for each RAM sector configuration set:

```
Mcu_InitRamSection(RamSectorConfigurationID);
```

This API must be called on the core that `McuRamSectorSettingConf` is allocated.

All other APIs (except for `Mcu_CheckClockStatus`, `Mcu_CheckModeStatus`) calls might be used after successful initialization of the MCU whenever necessary. These functions are:

```
Mcu_GetPllStatus();
Mcu_GetResetRawValue();
Mcu_GetResetReason();
Mcu_PerformReset();
Mcu_GetVersionInfo(&versioninfo);
Mcu_SetMode(McuConf_McuModeSettingConf_MY_MODE);
```

Note: Because power mode must be controlled on each CPU core when entering system sleep or Deep Sleep mode, ensure that the MCU driver can run on each CPU core.

The `Mcu_SetMode` API must be called on the core that `McuModeSettingConf` is allocated.

Your application must provide the notification functions and their declarations that you configured. The file containing the declarations must be included using `McuGeneralConfiguration/McuIncludeFile`. The notification functions take no parameters and have void return type:

```
void MyNotificationFunction(void)
{
/* Insert your code here */
}
```

The notification function is called from an interrupt context.

2.4 Starting the build process

Do the following to build your application:

Note: For a clean build, use the build command with target `clean_all`. before (`make clean_all`).

1. On the command shell, type the following command to generate the necessary configuration-dependent files. See 3.3 Generated files.

   ```
   > make generate
   ```

2. Type the following command to resolve the required file dependencies:

   ```
   > make depend
   ```

3. Type the following command to compile and link the application:

   ```
   > make (optional target: all)
   ```

The application is now built. All files are compiled and linked to a binary file, which can be downloaded to the target CPU cores.

Note: *The MCU driver must be located on all CPU cores to enter low-power mode. In this case, the MCU driver must be built for all CPU cores.*

2.5 Measuring stack consumption

Do the following to measure stack consumption. It requires the Base module for proper measurement.

Note: *All files (including library files) should be rebuilt with the dedicated compiler option. The executable file built by this step must be used only to measure stack consumption.*

1. Add the following compiler option to the Makefile to enable stack consumption measurement:

   ```
   -DSTACK_ANALYSIS_ENABLE
   ```

2. Type the following command to clean library files:

   ```
   > make clean_lib
   ```

3. Follow the build process described in section 2.4 Starting the build process.
4. Follow the instructions in the release notes and measure the stack consumption.

2.6 Memory mapping

The *Mcu_MemMap.h* file in the *$(TRESOS_BASE)/plugins/MemMap_TS_T40D13M0I0R0/include* directory is a sample. This file is replaced by the file generated by MEMMAP module. Input to the MEMMAP module is generated *as Mcu_Bswmd.arxml* in the *$(PROJECT_ROOT)/output/generated/swcd* directory of your project folder.

2.6.1 Memory allocation keyword

- `MCU_START_SEC_CODE_ASIL_B` / `MCU_STOP_SEC_CODE_ASIL_B`

  The memory section type is CODE. All executable code is allocated in this section.

- `MCU_START_SEC_CONST_ASIL_B_UNSPECIFIED` / `MCU_STOP_SEC_CONST_ASIL_B_UNSPECIFIED`

  The memory section type is CONST. The following constants are allocated in this section:

  - All configuration data except reset.
  - Hardware register base address data.
  - Pointer to the driver status.

- `MCU_START_SEC_CONST_ASIL_B_32` / `MCU_STOP_SEC_CONST_ASIL_B_32`

  The memory section type is CONST. The following constants are allocated in this section:

  - Reset configuration data.

- `MCU_CORE[MasterCoreId]_START_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED` / `MCU_CORE[MasterCoreId]_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED`

  `MasterCoreId` means the `McuCoreConfigurationId` specified in `McuMasterCoreReference`.

  The memory section type is VAR. The following variables are allocated in this section:

  - Pointer to the configuration data.

- `MCU_CORE[ClockCoreId]_START_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED / MCU_CORE[ClockCoreId]_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_UNSPECIFIED`

  `ClockCoreId` means the `McuCoreConfigurationId` specified in `McuClockCoreAssignment`.

  The memory section type is VAR. The following variables are allocated in this section:

  - Pointer to the current clock configuration data.

- `MCU_CORE[ClockCoreId]_START_SEC_VAR_INIT_ASIL_B_GLOBAL_32 / MCU_CORE[ClockCoreId]_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_32`

  `ClockCoreId` means the `McuCoreConfigurationId` specified in `McuClockCoreAssignment`.

  The memory section type is VAR. The following variable is allocated in this section:

  - Wait cycle for disabling the FLL.
  - Wait cycle for disabling the PLL.
  - Wait cycle for disabling the SSCG.

- `MCU_CORE[McuCoreConfigurationId]_START_SEC_VAR_INIT_ASIL_B_GLOBAL_8 / MCU_CORE[McuCoreConfigurationId]_STOP_SEC_VAR_INIT_ASIL_B_GLOBAL_8`

  The memory section type is VAR. The following variables are allocated in this section:

  - Driver status.
  - Core ID for current mode.

- `MCU_CORE[MasterCoreId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED / MCU_CORE[MasterCoreId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_UNSPECIFIED`

  `MasterCoreId` means the `McuCoreConfigurationId` specified in `McuMasterCoreReference`.

  The memory section type is VAR. The following variables are allocated in this section:

  - Reset reason.
  - Reset raw value.

- `MCU_CORE[ClockCoreId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32 / MCU_CORE[ClockCoreId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32`

  `ClockCoreId` means the `McuCoreConfigurationId` specified in `McuClockCoreAssignment`.

  The memory section type is VAR. The following variable is allocated in this section:

  - Waiting time for setting the PWR_LVD_CTL register.

2.6.2 Memory allocation and constraints

All memory sections that store init or uninit status must be zero-initialized before any driver function is executed on any core. If core consistency checks are disabled, inconsistent parameters would be detected and reported by PPU and SMPU.

- `MCU_CORE[McuCoreConfigurationId]_START_VAR_[INIT_POLICY]_ASIL_B_LOCAL_[ALIGNMENT] / MCU_CORE[McuCoreConfigurationId]_STOP_VAR_[INIT_POLICY]_ASIL_B_LOCAL_[ALIGNMENT]`

  This section is read/write accessed only from the core represented by McuCoreConfigurationId. Therefore, this section can be allocated to any RAM region. It is recommended to allocate the section to cache-able SRAM, not TCRAM.

- `MCU_CORE[McuCoreConfigurationId]_START_VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]`
  /
  `MCU_CORE[McuCoreConfigurationId]_STOP_VAR_[INIT_POLICY]_ASIL_B_GLOBAL_[ALIGNMENT]`

  This section is read/write accessed from the core represented by `McuCoreConfigurationId` and read accessed from the other cores. Therefore, this section must not be allocated to TCRAM. For the core represented by `McuCoreConfigurationId`, this section must be allocated to either non-cache-able or write-through cache-able SRAM area. For performance, it is recommended to allocate the section to write-through cache-able SRAM. For other cores, this section must be allocated to non-cache-able SRAM area.

- STACK section

  TCRAM has dedicated memory for each core at the same address, and because of its performance it is recommended to allocate STACK to TCRAM.

For the details of `INIT_POLICY` and `ALIGNMENT`, see the *Specification of memory mapping* [7].

3 Structure and dependencies

The MCU driver consists of static, configuration, and generated files.

3.1 Static files

- $(PLUGIN_PATH)=$(TRESOS_BASE)/plugins/MCU_TS_* path to the MCU module plugin.
- *$(PLUGIN_PATH)/lib_src* contains all static source files of the MCU driver. These files represent the functionality of the driver therefore, the files are independent of any configuration sets.
- *$(PLUGIN_PATH)/src* contains configuration-dependent source files or special derivative files. Each file is rebuilt when the configuration set is changed.

All necessary source files will be automatically compiled and linked during the build process and all include paths will be set if the MCU driver is enabled.

- *$(PLUGIN_PATH)/include* is the basic public include directory that you need to include in *Mcu.h*.
- *$(PLUGIN_PATH)/autosar* directory contains the AUTOSAR ECU parameter definition with vendor, architecture, and derivative-specific adaptations to create a correct matching parameter configuration for the MCU module.

3.2 Configuration files

The configuration of the MCU driver is done with the EB tresos Studio software. When saving a project, the configuration description is written to the *Mcu.xdm* file. It is located under *$(PROJECT_ROOT)/config* in your project folder. This file serves as the input to generate the configuration-dependent source and header files during the build process.

3.3 Generated files

During the build process, the following files are generated based on the current configuration description, and are in the *output/generated* subfolder of your *project* folder:

- *include/Mcu_Cfg.h* provides settings of configurations with pre-compile attribute; for example, all symbolic names required by the API for clock, RAM sector, and low-power mode configurations. In addition, this file defines a `DemEventId` parameter of the DEM module, which is referred in the configuration. The DEM module is included by *Mcu.h*.
- *include/Mcu_Cfg_Arch.h* provides architecture-specific settings of configurations with pre-compile attribute; for example, each hardware IP register base address.
- *include/Mcu_PBcfg.h* provides settings of configurations with post-build attribute; for example, symbolic names of module configurations. In addition, it defines the number of modules, clock, RAM sector, and low-power mode configurations.
- *include/Mcu_PBcfg_Arch.h* provides architecture-specific settings of configurations with post-build attribute.
- include/Mcu_ExternalInclude.h includes the header files specified by McuIncludeFile.
- *src/Mcu_PBcfg.c* contains the constants for the MCU configuration.
- *src/Mcu_Irq.c* contains the interrupt service routine.

Note: *Generated source files need not to be added to your application make file. They are compiled and linked automatically during the build process. Check the consistency of the configuration and generated files.*

- *src/Mcu_CalloutWrapper.c* contains the wrapper functions for callouts.
- *swcd/Mcu_Bswmd.arxml* contains BSW module description.

Note: *Additional steps are required for the generation of the BSW module description.*
*In EB tresos Studio, follow the menu path **Project** > **Build Project** and select **generate_swcd**.*

3.4 Dependencies

3.4.1 DET

If default error detection is enabled in the MCU driver module configuration, DET must be installed, configured, and integrated into the application.

3.4.2 DEM

If clock failure notification or reset failure notification is enabled in the MCU driver module configuration, DEM must be installed, configured, and integrated into the application.

3.4.3 AUTOSAR OS

The OS must be used to configure and to create the ISR vector table entries for the MCU driver. See section 6.2 Interrupts for more information. `GetCoreID` can optionally be set to the configuration parameter `McuGetCoreIdFunction`.

3.4.4 BSW scheduler

The MCU driver uses the following services of the BSW scheduler (originally named *SchM*, now *BswM*) to enter and leave critical sections:

- `SchM_Enter_Mcu_MCU_EXCLUSIVE_AREA_[McuCoreConfigurationId](void)`
- `SchM_Exit_Mcu_MCU_EXCLUSIVE_AREA_[McuCoreConfigurationId](void)`

Make sure that the BSW scheduler is properly configured and initialized before using the MCU driver.

3.4.5 Error callout handler

The error callout handler is called on every error that is detected regardless of whether default error detection is enabled or disabled. The error callout handler is an ASIL safety extension that is not specified by AUTOSAR. It is configured via the configuration parameter `McuErrorCalloutFunction`.

4 EB tresos Studio configuration interface

The GUI is not part of this delivery. For further information, see *EB tresos Studio for ACG8 user's guide* [8].

4.1 General configuration

The `McuGeneralConfiguration` container has the following parameters to configure the general functions of the MCU driver:

- `McuDevErrorDetect` enables or disables the development error notification feature for the MCU driver module.

  Setting this parameter to FALSE disables the notification of development errors via DET. However, in contrast to the AUTOSAR specification, detection of development errors is still enabled as a safety mechanism (fault detection).

- `McuGetRamStateApi` is not used and is not being evaluated.
- `McuInitClock` enables or disables the clock initialization functionality.
- `McuNoPll` enables or disables the functionality of the PLL clock.
- `McuPerformResetApi` enables or disables the reset functionality.
- `McuVersionInfoApi` enables or disables the functionality to read the module version information.
- `McuSafetyFunctionApi` adds or removes the `Mcu_CheckClockStatus()` and `Mcu_CheckModeStatus()` services from the code.
- `McuErrorCalloutFunction` is used to specify the error callout function name. The function is called on every error. The ASIL level of this function limits the ASIL level of the MCU driver.

Note: *`McuErrorCalloutFunction` must be a valid C function name; otherwise an error would occur in the configuration phase.*

- `McuEnableGetCoreIDApi` adds or removes the `Mcu_GetCoreID()` service from the code.
- `McuEnableSetModeApiOnly` enables only the `Mcu_Init()`, `Mcu_SetMode()`, `Mcu_CheckClockStatus()`, and `Mcu_CheckModeStatus()` services from the code. This option can be enabled when the configuration is for the core other than main core (for example, Arm® Cortex® M0+ for sleep mode).

Note: *If this option is enabled, the configuration parameters `McuInitClock`, `McuNoPll`, `McuPerformResetApi`, and `McuVersionInfoApi` have no effect.*

- `McuIncludeFile` lists the file names that will be included within the driver. Any application-specific symbol that is used by the MCU driver module configuration (such as error callout function) should be included by configuring this parameter.

Note: *`McuIncludeFile` must be a filename with the .h extension and a unique name; otherwise some errors would occur in the configuration phase.*

4.2 MCU module configuration

The `McuModuleConfiguration` container has the following the parameters to configure the microcontroller-specific functions:

- `McuClockSrcFailureNotification` enables or disables clock failure notification to DEM.
  - `DISABLED`: Disables clock failure notification to DEM.
  - `ENABLED`: Enables clock failure notification to DEM.
- `McuResetFailureNotification` enables or disables reset failure notification to DEM.
- `McuNumberOfMcuModes` specifies the number of modes configured.
- `McuRamSectors` specifies the number of RAM sectors configured.
- `McuResetSetting` is not used; instead, the architecture-specific parameter `McuResetSelect` is used.
- `McuResetSelect` specifies the reset type:
  - `MCU_SW_RESET`: Software reset: This parameter relates to the MCU-specific reset configuration. It applies to the `Mcu_PerformReset()` function, which performs a microcontroller reset using the hardware function of the microcontroller.
- `McuEnableCacheFlushBeforeReset` enables or disables flushing cache before performing reset.

Note: *`McuEnableCacheFlushBeforeReset` is available only if `McuPerformResetApi` is TRUE, `McuResetSelect` is activated.*

Note: *If this parameter is TRUE, the stack and static data of the MCU driver must be allocated to a non-cached memory area.*

- `McuRamWriteBufferTimeoutBeforeReset` specifies the timeout count value used when checking whether the RAM write buffer is empty.
  - 1 - 4294967295: Timeout count value used when verifying that the RAM write buffer is empty.
- `McuForcedResetEnable` enables or disables performing reset even if an error occurs in `Mcu_PerformReset()`.

Note: *`McuForcedResetEnable` is available only if `McuPerformResetApi` is TRUE and `McuResetSelect` is activated.*

This parameter is also applied when `Mcu_PerformReset()` is called on the core that the system resource is not assigned (The system resource is assigned by `McuModeSettingConf` contains `McuUpdateSystemResource` set to TRUE).

- `McuRam0Macro<n>RetainBeforeReset` (<n> = 0 … 15) specifies whether to retain RAM0 Macro <n> during reset.

Note: *`McuRam0Macro<n>RetainBeforeReset` is available only if `McuPerformResetApi` is TRUE, `McuResetSelect` is activated, and RAM0 Macro <n> is supported by the derivative.*

Note: *If this parameter is TRUE, the stack and static data of the MCU driver must not be allocated to the SRAM0 area corresponding to the RAM0 macro <n>.*

This parameter is not applied the case that `Mcu_PerformReset()` is called on the core that system resource is not assigned (The system resource is assigned by `McuModeSettingConf` contains `McuUpdateSystemResource` set to TRUE.) In this case, RAM0 Macro <n> is not retained.

- `McuRam1RetainBeforeReset` specifies whether to retain RAM1 during reset.

Note: *`McuRam1RetainBeforeReset` is available only if `McuPerformResetApi` is TRUE, `McuResetSelect` is activated and RAM1 is supported by the derivative.*

Note: *If this parameter is TRUE, the stack and static data of the MCU driver must not be allocated to the SRAM1 area.*

This parameter is not applied the case that `Mcu_PerformReset()` is called on the core that system resource is not assigned by `McuModeSettingConf` with `McuUpdateSystemResource` whose value is set to TRUE. In this case, RAM1 is not retained.

- `McuRam2RetainBeforeReset` specifies whether to retain RAM2 during reset.

Note: *`McuRam2RetainBeforeReset` is available only if `McuPerformResetApi` is TRUE, `McuResetSelect` is activated, and RAM2 is supported by the derivative.*

Note: *If this parameter is TRUE, the stack and static data of the MCU driver must not be allocated to the SRAM2 area.*

This parameter is not applied the case that `Mcu_PerformReset()` is called on the core that system resource is not assigned by `McuModeSettingConf` with `McuUpdateSystemResource` whose value is set to TRUE. In this case, RAM2 is not retained.

- `McuClearResetReasonRegister` enables or disables clearing the reset reason registers during `Mcu_Init()`.

Note: *If `McuClearResetReasonRegister` is FALSE, you should initialize the following reset reason registers; otherwise the `Mcu_GetResetReason()` API would not be able to read the reset reason correctly.*

  - `RES_CAUSE`
  - `RES_CAUSE2`

- `McuEnableDefaultClock` initializes the clock during `Mcu_Init()` when checked. This has the advantage that subsequent calls to `Mcu_InitClock()` can be omitted and all subsequent initialization or startup operations benefit from the higher speed of the clock.

Note: *If `McuInitClock` is FALSE, this parameter should also be disabled.*

- `McuDefaultClockSetting` selects the default clock setting configuration from `McuClockSettingConfig`, which is used to initialize the clock automatically when the MCU is initialized (`Mcu_Init()`).

Note: *This parameter is available only if `McuEnableDefaultClock` is TRUE.*

MCU module configuration contains the following containers:

- `McuLowVoltageDetectionCallbackFunctions` (see section 4.3 MCU low-voltage-detection callback functions)
- `McuDemEventParameterRefs` (see section 4.4 MCU DEM event parameter references)
- `McuClockSettingConfig` (see section 4.5 MCU clock setting configuration)
- `McuModeSettingConf` (see section 4.6 MCU mode settings configuration)
- `McuRamSectorSettingConf` (see section 4.7 MCU RAM section configuration)

4.3 MCU low-voltage-detection callback functions

The `McuLowVoltageDetectionCallbackFunctions` container has the following parameters to configure the callback functions for notifying an error from low-voltage detection:

- `McuHvLvd1Notification` specifies the function for notifying the error from HVLVD1.
- `McuHvLvd2Notification` specifies the function for notifying the error from HVLVD2.

Note: *Notifications must be declared and defined outside the MCU module. The file containing the declarations must be included using `McuIncludeFile`.*

4.4 MCU DEM event parameter references

The `McuDemEventParameterRefs` container has the following parameters to configure DEM event notification:

- `MCU_E_CLOCK_FAILURE` refers to the configured DEM event to report "Clock source failure".
- `MCU_E_RESET_FAILURE` refers to the configured DEM event to report "Reset failure".

4.5 MCU clock setting configuration

The `McuClockSettingConfig` container has the following parameters to configure the clock:

- `McuClockSettingId` is a logical ID of the clock setting. This value will be assigned to the following symbolic names:
  - The symbolic name derived from the `McuClockSettingConfig` container short name is prefixed with "`McuConf_McuClockSettingConfig_`".
  - **Example:** `McuConf_McuClockSettingConfig_McuClockSettingConfig_0`.

Note: *In the same `McuModuleConfiguration` container, `McuClockSettingId` must be unique and consecutive.*

- `McuUnlockWatchdogEnable` enables unlocking the watchdog once before setting the LF clock and ILO0 clock.

Note: *If `McuUnlockWatchdogEnable` is FALSE, setting of the LF clock and ILO0 clock will be skipped when watchdog is locked.*

- `McuClockCoreAssignment` specifies the reference to the `McuCoreConfiguration` for assigning the core to `McuClockSettingConfig`.

Note: *McuClockCoreAssignment must have a valid reference to McuCoreConfiguration. The value of McuClockCoreAssignment must be same for all McuClockSettingConfig.*

The MCU clock configuration holds the following containers.

- McuClocksIn (see section 4.5.1 MCU clock input)
- McuClockSettings (see section 4.5.2 MCU clock settings)
- McuClockReferencePoint (see section 4.5.3 MCU clock reference point)

Note: *The acceptable frequency range of each clock shown in the following sections depends on the subderivative. For more information about the derivative-dependent clock frequency, see the hardware technical reference manual or datasheet.*

Note: *If each clock is disabled, its frequency will be set to 0.0 (in Hz).*

Note: *As the number of clock configuration increases, the duration of critical section in Mcu_Init(), Mcu_InitClock(), and Mcu_SetMode() will be longer.*

4.5.1 MCU clock input

The McuClocksIn container has the following parameters to configure input clocks:

- McuImoEnable enables or disables the IMO clock.

Note: *This parameter must be set to TRUE at all times for all functions to work properly.*

- McuImoFrequency specifies the frequency of the IMO clock (in Hz).

Note: *If McuImoEnable is FALSE, this parameter must be set to 0.0 (in Hz).*

- McuExtFrequency specifies the frequency of the external clock (in Hz).
- McuAlthfFrequency specifies the frequency of the ALTHF clock (in Hz).

Note: *If ALTHF clock is not supported by the derivative, this parameter must be set to 0.0 (in Hz).*

- McuAltlfFrequency specifies the frequency of the ALTLF clock (in Hz).

Note: *If ALTLF clock is not supported by the derivative, this parameter must be set to 0.0 (in Hz).*

- McuDsiOut<n>Frequency (<n> = 0 … 15) specifies the frequency of the DSI output <n> clock (in Hz).

Note: *McuDsiOut<n>Frequency is available only if each DSI mux is supported by the derivative.*

The MCU clock input configuration holds the following containers:

- McuEcoSettings (see section 4.5.1.1 MCU ECO clock settings)
- McuLpEcoSettings (see section 4.5.1.4 MCU LPECO clock settings)
- McuIloSettings (see section 4.5.1.6 MCU ILO clock settings)
- McuWcoSettings (see section 4.5.1.8 MCU WCO clock settings)

4.5.1.1 MCU ECO clock settings

The `McuEcoSettings` container has the following parameters to configure the ECO clock:

- `McuEcoEnable` enables or disables the ECO clock.
- `McuAgcEnable` enables or disables automatic gain control.
- `McuEcoAmpStabilizationTimeout` specifies the timeout count value used when verifying whether the ECO clock has stabilized.
  - 1 - 4294967295: Timeout count value used when verifying whether the ECO clock has stabilized.

Note: *Even if `McuEcoAmpStabilizationTimeout` is deactivated, the ECO clock status will be checked once.*

- `McuEcoFrequency` specifies the frequency of the ECO clock oscillator (in Hz).

Note: *If `McuEcoEnable` is FALSE, this parameter must be set to 0.0 (in Hz).*

The MCU ECO clock settings configuration holds the following containers:

- `McuEcoPrescalerSettings` (see section 4.5.1.2 MCU ECO prescaler settings)
- `McuEcoTrimSettings` (see section 4.5.1.3 MCU ECO trim settings)

4.5.1.2 MCU ECO prescaler settings

Use the following parameters to configure the ECO prescaler:

- `McuEcoPrescalerEnable` enables or disables the ECO prescaler.
- `McuEcoPrescalerValue` specifies the ECO prescaler value.
  - 1 - 1024.99609375: ECO prescaler value.
- `McuEcoPrescalerEnableTimeout` specifies the timeout count value used when verifying whether the ECO prescaler is enabled.
  - 1 - 4294967295: Timeout count value used when verifying whether the ECO prescaler is enabled.

Note: *Even if `McuEcoPrescalerEnableTimeout` is deactivated, the ECO prescaler status will be checked once.*

- `McuEcoPrescaledFrequency` specifies the frequency of the prescaled ECO clock (in Hz).

Note: *`McuEcoPrescaledFrequency` automatically displays the resulting frequency calculated by the following formula: `McuEcoPrescaledFrequency` = `McuEcoFrequency` / `McuEcoPrescalerValue`*

4.5.1.3 MCU ECO trim settings

The `McuEcoTrimSettings` container has the following parameters to configure the ECO clock trim setting:

- `McuEcoAmplitudeTrimValue` specifies the ECO amplitude trim value to set the crystal drive level.
  - `MCU_ECO_AMPLITUDE_TRIM_VP_LESS_THAN_0_35V`: ECO amplitude trim when Vp < 0.35 [V].
  - `MCU_ECO_AMPLITUDE_TRIM_VP_LESS_THAN_0_40V`: ECO amplitude trim when Vp < 0.40 [V].
  - …
- `McuEcoFeedbackResistorTrimValue` specifies the ECO feedback resistor trim value.
  - 0 - 3: ECO feedback resistor trim value.
- `McuEcoFilterTrimValue` specifies the ECO low-pass filter frequency trim value.
  - 0 - 3: ECO low-pass filter frequency trim value.
- `McuEcoGainTrimValue` specifies the ECO amplifier gain trim value.
  - 0 - 7: ECO amplifier gain trim value.
- `McuEcoWatchdogTrimValue` specifies the ECO watchdog trim value.
  - `MCU_ECO_WATCHDOG_TRIM_VP_GREATER_THAN_0_05V`: ECO watchdog trim when Vp > 0.05 [V].
  - `MCU_ECO_WATCHDOG_TRIM_VP_GREATER_THAN_0_10V`: ECO watchdog trim when Vp > 0.10 [V].

4.5.1.4 MCU LPECO clock settings

The `McuLpEcoSettings` container has the following parameters to configure the LPECO clock:

- `McuLpEcoEnable` enables or disables the LPECO clock.
- `McuLpEcoStopForUpdate` enables or disables stopping the LPECO clock once before setting.

Note: *If `McuLpEcoStopForUpdate` is FALSE, setting of the LPECO clock will be skipped when it is running.*

- `McuLpEcoAmplitudeDetectorEnable` enables or disables the minimum amplitude detector for the LPECO clock.

Note: *If the minimum amplitude detector is enabled, it is also checked that amplitude is sufficient for LPECO stabilization.*

- `McuLpEcoMaximumAmplitude` specifies the LPECO maximum oscillation amplitude.
  - `MCU_LPECO_AMPLITUDE_1_35V`: LPECO maximum oscillation amplitude 1.35 [V].
  - `MCU_LPECO_AMPLITUDE_1_80V`: LPECO maximum oscillation amplitude 1.80 [V].
- `McuLpEcoLoadCapacitanceRange` specifies the LPECO load capacitance range of the crystal.
  - `MCU_LPECO_LOAD_CAPACITANCE_TO_10PF`: LPECO load capacitance range [5 pF – 10 pF].
  - `MCU_LPECO_LOAD_CAPACITANCE_TO_15PF`: LPECO load capacitance range (10 pF – 15 pF].
  - `MCU_LPECO_LOAD_CAPACITANCE_TO_20PF`: LPECO load capacitance range (15 pF – 20 pF].
  - `MCU_LPECO_LOAD_CAPACITANCE_TO_25PF`: LPECO load capacitance range (20 pF – 25 pF].
- `McuLpEcoAmpStabilizationTimeout` specifies the timeout count value used when verifying that the LPECO clock has stabilized.
  - 1 - 4294967295: Timeout count value used when verifying that the LPECO clock has stabilized.

Note: *Even if `McuLpEcoAmpStabilizationTimeout` is deactivated, the LPECO clock status will be checked once.*

- `McuLpEcoFrequency` specifies the frequency of the LPECO clock oscillator (in Hz).

Note: *If `McuLpEcoEnable` is FALSE, this parameter must be set to 0.0 (in Hz).*

The MCU LPECO clock settings configuration holds the following container:

- `McuLpEcoPrescalerSettings` (see section 4.5.1.5 MCU LPECO prescaler settings)

4.5.1.5 MCU LPECO prescaler settings

Use the following parameters to configure the LPECO prescaler:

- `McuLpEcoPrescalerEnable` enables or disables the LPECO prescaler.
- `McuLpEcoPrescalerValue` specifies the LPECO prescaler value.
  - 1 - 1024.99609375: LPECO prescaler value.
- `McuLpEcoPrescalerEnableTimeout` specifies the timeout count value used when verifying that the LPECO prescaler is enabled.
  - 1 - 4294967295: Timeout count value used when verifying that the LPECO prescaler is enabled.

Note: *Even if `McuLpEcoPrescalerEnableTimeout` is deactivated, the LPECO prescaler status will be checked once.*

- `McuLpEcoPrescaledFrequency` specifies the frequency of the prescaled LPECO clock (in Hz).

Note: *`McuLpEcoPrescaledFrequency` automatically displays the resulting frequency calculated by the following formula:*
`McuLpEcoPrescaledFrequency` = `McuLpEcoFrequency` / `McuLpEcoPrescalerValue`

4.5.1.6 MCU ILO clock settings

The `McuIloSettings` container has the following parameters to configure the ILO clocks:

- `McuIlo0Enable` enables or disables the ILO0 clock.
- `McuIlo0OnBackupEnable` enables or disables the ILO0 remaining on if the backup domain is supported by the derivative.
- `McuIlo0MonitorEnable` enables or disables the internal ILO0 clock monitoring circuit.

Note: *This parameter must be set to FALSE as the ILO0 clock monitoring feature is no longer supported.*

- `McuIlo0Frequency` specifies the frequency of the ILO0 clock oscillator (in Hz).

Note: *If `McuIlo0Enable` is FALSE, this parameter must be set to 0.0 (in Hz).*

The MCU ILO clock settings configuration holds the following container:

- `McuIlo1Settings` (see section 4.5.1.7 MCU ILO1 clock settings)

4.5.1.7 MCU ILO1 clock settings

The `McuIlo1Settings` container has the following parameters to configure the ILO1 clock:

- `McuIlo1Enable` enables or disables the ILO1 clock.
- `McuIlo1MonitorEnable` enables or disables the internal ILO1 clock monitoring circuit.

Note: *This parameter must be set to FALSE as the ILO1 clock monitoring feature is no longer supported.*

- `McuIlo1Frequency` specifies the frequency of the ILO1 clock oscillator (in Hz).

Note: *If `McuIlo1Enable` is FALSE, this parameter must be set to 0.0 (in Hz).*

4.5.1.8 MCU WCO clock settings

The `McuWcoSettings` container has the following parameters to configure the WCO clock:

- `McuWcoEnable` enables or disables the WCO clock.
- `McuWcoStopForUpdate` enables or disables stopping the WCO clock once before setting.

Note: *If `McuWcoStopForUpdate` is FALSE, setting of the WCO clock will be skipped when it is running.*

- `McuWcoType` specifies the type of board-level connections to the WCO pins.
  - `MCU_WCO_WATCH_CRYSTAL`: Watch crystal
  - `MCU_WCO_CLOCK_SIGNAL`: Clock signal
- `McuWcoPrescaler` specifies the prescaler for real-time clock. This parameter can be set when `McuWcoEnable` is TRUE and `McuWcoType` is `MCU_WCO_CLOCK_SIGNAL`.
  - `MCU_WCO_SQUAREWAVE_32768HZ`: 32768-Hz square wave.
  - `MCU_WCO_SINEWAVE_60HZ`: 60-Hz sine wave.
  - `MCU_WCO_SINEWAVE_50HZ`: 50-Hz sine wave.

Note: *The valid range of `McuWcoPrescaler` is device-specific. See the hardware register technical reference manual for details.*

- `McuWcoStabilizationTimeout` specifies the timeout count value used when verifying that the WCO clock has stabilized.
  - 1 - 4294967295: Timeout count value used when verifying that the WCO clock has stabilized.

Note: *Even if `McuWcoStabilizationTimeout` is deactivated, the WCO clock status will be checked once.*

- `McuWcoFrequency` specifies the frequency of the WCO clock oscillator (in Hz).

Note: *If `McuWcoEnable` is FALSE, this parameter must be set to 0.0 (in Hz).*

4.5.2 MCU clock settings

The `McuClockSettings` container holds the configurations for clock common settings:

- `McuPclkEnableTimeout` specifies the timeout count value used when verifying that the PCLK has enabled. This parameter is not used.
  - 1 - 4294967295: Timeout count value used when verifying that the PCLK has enabled.
- `McuPeriGroupBusTransferTimeout` specifies the AHB-Lite bus transfer timeout value in the peripheral group clock cycle.
  - 0 - 65534: AHB-Lite bus transfer timeout value.
- `McuBackupClockSource` specifies the source clock of the backup clock.
  - `MCU_CLOCK_WCO`: WCO clock.
  - `MCU_CLOCK_ALTBAK`: Alternate backup domain clock (LF clock).
  - `MCU_CLOCK_ILO0`: ILO0 clock.
  - `MCU_CLOCK_LPECO_PRESCALE`: Prescaled LPECO.
- `McuBackupClockFrequency` is the frequency of the backup clock (in Hz).

Note: *`McuBackupClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
McuBackupClockFrequency = The frequency of the clock specified by McuBackupClockSource

- `McuFast0ClockFrequency` is the frequency of the fast 0 clock (in Hz).

Note: *`McuFast0ClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
If the device supports Arm® Cortex®-M4 CPU, `McuFast0ClockFrequency` = (The value of `McuClockRootFrequency` for which the corresponding `McuClockRoot` is set to `MCU_CLOCK_ROOT0)` / `McuFast0ClockDivision`
If the device supports Arm® Cortex®-M7 CPU, `McuFast0ClockFrequency` = (The value of `McuClockRootFrequency` for which the corresponding `McuClockRoot` is set to `MCU_CLOCK_ROOT1)` / `McuFast0ClockDivision`

- `McuFast0ClockDivision` specifies the division value of the fast 0 clock.
  - 1.0 - 256.96875: Fast 0 clock division value.

Note: *Fractional value cannot be configured on some subderivatives.*

- `McuFast1ClockFrequency` is the frequency of the fast 1 clock (in Hz).

Note: *`McuFast1ClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
`McuFast1ClockFrequency` = (The value of `McuClockRootFrequency` for which the corresponding `McuClockRoot` is set to `MCU_CLOCK_ROOT1)` / `McuFast1ClockDivision`

- `McuFast1ClockDivision` specifies the division value of the fast 1 clock.
  - 1.0 – 256.96875: Fast 1 clock division value.
- `McuFast2ClockFrequency` is the frequency of the fast 2 clock (in Hz).

Note: *McuFast2ClockFrequency automatically displays the resulting frequency calculated by following formula:*

McuFast2ClockFrequency = (The value of McuClockRootFrequency for which the corresponding McuClockRoot is set to MCU_CLOCK_ROOT1) / McuFast2ClockDivision

- McuFast2ClockDivision specifies the division value of the fast 2 clock.
  - 1.0 – 256.96875: Fast 2 clock division value.
- McuFast3ClockFrequency is the frequency of the fast 3 clock (in Hz).

Note: *McuFast3ClockFrequency automatically displays the resulting frequency calculated by following formula:*

McuFast3ClockFrequency = (The value of McuClockRootFrequency for which the corresponding McuClockRoot is set to MCU_CLOCK_ROOT1) / McuFast3ClockDivision

- McuFast3ClockDivision specifies the division value of the fast 3 clock.
  - 1.0 – 256.96875: Fast 3 clock division value.
- McuSlowClockFrequency is the frequency of the slow clock (in Hz).

Note: *McuSlowClockFrequency automatically displays the resulting frequency calculated by the following formula:*
If the device supports Arm® Cortex®-M4 CPU, McuSlowClockFrequency = McuPeriClockFrequency / McuSlowClockDivision
If the device supports Arm® Cortex®-M7 CPU, McuSlowClockFrequency = McuMemClockFrequency / McuSlowClockDivision

- McuSlowClockDivision specifies the division value of the slow clock.
  - 1 - 256: Slow clock division value.
- McuPeriClockFrequency is the frequency of the peripheral clock (in Hz).

Note: *McuPeriClockFrequency automatically displays the resulting frequency calculated by the following formula:*
McuPeriClockFrequency = (The value of McuClockRootFrequency for which the corresponding McuClockRoot is set to MCU_CLOCK_ROOT0) / McuPeriClockDivision

- McuPeriClockDivision specifies the division value of the peripheral clock.
  - 1 - 256: Peripheral clock division value.
- McuMemClockFrequency is the frequency of the memory clock (in Hz).

Note: *McuMemClockFrequency automatically displays the resulting frequency calculated by the following formula:*
McuMemClockFrequency = (The value of McuClockRootFrequency for which the corresponding McuClockRoot is set to MCU_CLOCK_ROOT0) / McuMemClockDivision

- McuMemClockDivision specifies the division value of the memory clock.
  - 1 - 256: Memory clock division value.
- McuTrcDbgClockFrequency is the frequency of the trace and debug clock (in Hz).

Note: *`McuTrcDbgClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
`McuTrcDbgClockFrequency` = (The value of `McuClockRootFrequency` for which the corresponding `McuClockRoot` is set to `MCU_CLOCK_ROOT0`) / `McuTrcDbgClockDivision`

- `McuTrcDbgClockDivision` specifies the division value of the trace and debug clock.
  - 1 - 256: Trace and debug clock division value.
- `McuFlashWaitCycle` specifies the wait cycle for accessing the flash memory.
  - 0 - 15: Wait cycle for accessing the flash memory.
- `McuFlash1WaitCycle` specifies the wait cycle for accessing the FLASH1 memory.
  - 0 - 15: Wait cycle for accessing the FLASH1 memory.
- `McuFastRomWaitCycle` specifies the wait cycle for accessing the ROM on the fast clock domain.
  - 0 - 3: Wait cycle for accessing the ROM on the fast clock domain.
- `McuSlowRomWaitCycle` specifies the wait cycle for accessing the ROM on the slow clock domain.
  - 0 - 3: Wait cycle for accessing the ROM on the slow clock domain.
- `McuFastRam0WaitCycle` specifies the wait cycle for accessing the RAM0 on the fast clock domain.
  - 0 - 3: Wait cycle for accessing the RAM0 on the fast clock domain.
- `McuSlowRam0WaitCycle` specifies the wait cycle for accessing the RAM0 on the slow clock domain.
  - 0 - 3: Wait cycle for accessing the RAM0 on the slow clock domain.
- `McuFastRam1WaitCycle` specifies the wait cycle for accessing the RAM1 on the fast clock domain.
  - 0 - 3: Wait cycle for accessing the RAM1 on the fast clock domain.
- `McuSlowRam1WaitCycle` specifies the wait cycle for accessing the RAM1 on the slow clock domain.
  - 0 - 3: Wait cycle for accessing the RAM1 on the slow clock domain.
- `McuFastRam2WaitCycle` specifies the wait cycle for accessing the RAM2 on the fast clock domain.
  - 0 - 3: Wait cycle for accessing the RAM2 on the fast clock domain.
- `McuSlowRam2WaitCycle` specifies the wait cycle for accessing the RAM2 on the slow clock domain.
  - 0 - 3: Wait cycle for accessing the RAM2 on the slow clock domain.
- `McuCsvReferenceClock` specifies the reference clock of the clock supervisor.
  - `MCU_CLOCK_IMO`: IMO clock
  - `MCU_CLOCK_EXTCLK`: External clock
  - `MCU_CLOCK_ECO`: ECO clock
  - `MCU_CLOCK_ALTHF`: ALTHF clock

The MCU clock settings configuration holds the following containers:

- `McuClockPathSettings` (see section 4.5.2.1 MCU clock path settings)
- `McuFllSettings` (see section 4.5.2.2 MCU FLL clock settings)
- `McuPllSettings` (see section 4.5.2.3 MCU PLL clock settings)
- `McuSscgPllSettings` (see section 4.5.2.4 MCU SSCG PLL clock settings)
- `McuClockRootSettings` (see section 4.5.2.5 MCU clock root settings)
- `McuPclkGroupSettings` (see section 4.5.2.6 MCU PCLK group settings)
- `McuPeriGroupSettings` (see section 4.5.2.9 MCU peripheral group settings)
- `McuLfClockSettings` (see section 4.5.2.11 MCU LF clock settings)

- McuPumpClockSettings (see section 4.5.2.12 MCU pump clock settings)
- McuTimerClockSettings (see section 4.5.2.13 MCU timer clock settings)
- McuClockOutputSettings (see section 4.5.2.14 MCU clock output settings)
- McuCsvSettings (see section 4.5.2.15 MCU clock supervisor settings)

4.5.2.1 MCU clock path settings

The McuClockPathSettings container has the following parameters to configure the clock path:

- McuClockPath specifies the clock path.
  - MCU_CLOCK_PATH<n>: Clock path <n> (<n> = 0 … 15).

Note: *In the same McuClockSettingConfig container, McuClockPath must be unique.*

Note: *Selectable clock paths depend on the subderivative. The clock path not used for FLL clock, PLL clock, and SSCG PLL clock can be set.*

- McuClockPathFrequency is the frequency of the clock path specified by McuClockPath (in Hz).

Note: *McuClockPathFrequency automatically displays the resulting frequency calculated by the following formula:*
McuClockPathFrequency = (The frequency of the clock specified by McuClockPathSource)

- McuClockPathSource specifies the source clock for the clock path specified by McuClockPath.
  - MCU_CLOCK_IMO: IMO clock
  - MCU_CLOCK_EXTCLK: External clock
  - MCU_CLOCK_ECO: ECO clock
  - MCU_CLOCK_LPECO: LPECO clock
  - MCU_CLOCK_ILO0: ILO0 clock
  - MCU_CLOCK_ILO1: ILO1 clock
  - MCU_CLOCK_WCO: WCO clock
  - MCU_CLOCK_ALTHF: ALTHF clock
  - MCU_CLOCK_ALTLF: ALTLF clock
  - MCU_CLOCK_DSI<n>: DSI output <n> clock (<n> = 0 … 15).

Note: *Selectable source clocks depend on the subderivative.*

4.5.2.2 MCU FLL clock settings

The McuFllSettings container has the following parameters to configure the FLL clock:

- McuFllEnable enables or disables the FLL clock.

Note: *If this parameter is TRUE, McuFllCcoEnable must be set to TRUE.*

- McuFllStopForUpdate enables or disables to stop the FLL clock once before setting.

Note: *If `McuFllStopForUpdate` is FALSE, setting of the FLL clock will be skipped when it is running.*

- `McuFllAutoDistributeEnable` enables or disables the automatic distribution of the FLL clock.

Note: *If `McuFllEnable` is TRUE, this parameter should be set to TRUE.*

- `McuFllAutoDistributeType` specifies the automatic distribution type of the FLL clock.
  - `MCU_DISTRIBUTE_AFTER_LOCKED`: The FLL clock will be automatically distributed after being locked. If it is unlocked after being locked, it will be switched to its reference input clock automatically (bypass mode).
  - `MCU_DISTRIBUTE_ONLY_LOCKED`: The FLL clock will be automatically distributed after being locked. If it is unlocked after being locked, it will be gated OFF.

Note: *If `McuFllEnable` is TRUE, this parameter should be set to `MCU_DISTRIBUTE_AFTER_LOCKED`.*

- `McuFllStabilizationTimeout` specifies the timeout count value used when verifying whether the FLL clock has stabilized.
  - 1 - 4294967295: Timeout count value used when verifying whether the FLL clock has stabilized.
- `McuFllFrequency` is the frequency of the FLL clock (in Hz).

Note: *`McuFllFrequency` automatically displays the resulting frequency calculated by the following formula:*
*`McuFllFrequency` = ((The frequency of the clock specified by `McuFllSource` / `McuFllReferenceDivision`) * `McuFllMultiplication`) / `McuFllOutputDivision`*

- `McuFllSource` specifies the source clock of the FLL clock:
  - `MCU_CLOCK_IMO`: IMO clock
  - `MCU_CLOCK_EXTCLK`: External clock
  - `MCU_CLOCK_ECO`: ECO clock
  - `MCU_CLOCK_LPECO`: LPECO clock
  - `MCU_CLOCK_ILO0`: ILO0 clock
  - `MCU_CLOCK_ILO1`: ILO1 clock
  - `MCU_CLOCK_WCO`: WCO clock
  - `MCU_CLOCK_ALTHF`: ALTHF clock
  - `MCU_CLOCK_ALTLF`: ALTLF clock
  - `MCU_CLOCK_DSI<n>`: DSI output <n> clock (<n> = 0 … 15)

Note: *Selectable source clocks depend on the subderivative.*

- `McuFllReferenceDivision` specifies the reference division value of the FLL clock.
  - 1 - 8191: FLL clock reference division value.
- `McuFllOutputDivision` specifies the output division value of the FLL clock.
  - 1 - 2: FLL clock output division value.
- `McuFllMultiplication` specifies the multiplication value of the FLL clock.
  - 0 - 262143: FLL clock multiplication value.

- `McuFllCcoEnable` enables or disables the CCO.
- `McuFllCcoOffset` specifies the allowed maximum value of the CCO offset.
  - 0 - 255: CCO offset allowed maximum value.
- `McuFllCcoAutoUpdateDisable` enables or disables the CCO frequency update by the FLL hardware.
- `McuFllCcoFrequencyCode` specifies the CCO frequency code.
  - 0 - 511: CCO frequency code.
- `McuFllCcoStabilizationTimeout` specifies the timeout count value used when verifying whether the CCO has stabilized.
  - 1 - 4294967295: Timeout count value used when verifying that the CCO has stabilized.

Note: Even if `McuFllCcoStabilizationTimeout` is deactivated, the CCO status will be checked once.

- `McuFllLockTolerance` specifies the lock tolerance, which is the error threshold when the FLL output is considered locked to the reference input.
  - 1 - 256: Lock tolerance value.
- `McuFllUpdateTolerance` specifies the update tolerance, which is the error threshold for when the FLL will update the CCO frequency settings.
  - 0 - 254: Update tolerance value.
- `McuFllSettlingCount` specifies the number of undivided reference clock cycles to wait after changing the CCO trim until the loop measurement restarts.
  - 0 - 8191: Reference clock cycle.
- `McuFllLoopFilterIGain` specifies the FLL loop filter integral gain setting.
  - `MCU_FLL_LOOP_FILTER_GAIN_1_BY_256`: 1/256
  - `MCU_FLL_LOOP_FILTER_GAIN_1_BY_128`: 1/128
- `McuFllLoopFilterPGain` specifies the FLL loop filter proportional gain setting.
  - `MCU_FLL_LOOP_FILTER_GAIN_1_BY_256`: 1/256
  - `MCU_FLL_LOOP_FILTER_GAIN_1_BY_128`: 1/128

4.5.2.3 MCU PLL clock settings

The `McuPllSettings` container has the following parameters to configure the PLL clock:

- `McuPllType` specifies the PLL clock.
  - `MCU_CLOCK_PLL<n>`: PLL<n> clock (<n> = 0 … 14).

Note: In the same `McuClockSettingConfig` container, `McuPllType` must be unique.

Selectable PLL clocks depend on the subderivative.

- `McuPllEnable` enables or disables the PLL clock specified by `McuPllType`.
- `McuPllStopForUpdate` enables or disables stopping the PLL clock specified by `McuPllType` once before setting.

Note: If `McuPllStopForUpdate` is FALSE, setting the PLL clock specified by `McuPllType` will be skipped when it is running.

- McuPllAutoDistributeEnable enables or disables the automatic distribution of the PLL clock specified by McuPllType.

Note: *If McuPllAutoDistributeEnable is TRUE, the PLL clock specified by McuPllType will be automatically distributed after locked and the manual distribution process in Mcu_DistributePllClock() will be skipped.*

- McuPllAutoDistributeType specifies the automatic distribution type of the PLL clock specified by McuPllType.
  - MCU_DISTRIBUTE_AFTER_LOCKED: The PLL clock specified by McuPllType will be automatically distributed after locked. If it becomes unlocked after locked, it will be automatically switched to its reference input clock (bypass mode).
  - MCU_DISTRIBUTE_ONLY_LOCKED: The PLL clock specified by McuPllType will be automatically distributed after locked. If it becomes unlocked after locked, it will be gated OFF.
- McuPllStabilizationTimeout specifies the timeout count value used when verifying the PLL clock specified by McuPllType has stabilized.
  - 1 - 4294967295: Timeout count value used when verifying that the PLL clock has stabilized.
- McuPllFrequency is the frequency of the PLL clock (in Hz).

Note: *McuPllFrequency automatically displays the resulting frequency calculated by the following formula:*
*McuPllFrequency = ((The frequency of the clock specified by McuPllSource / McuPllReferenceDivision) * McuPllFeedbackDivision) / McuPllOutputDivision*

- McuPllSource specifies the source clock of the PLL clock specified by McuPllType.
  - MCU_CLOCK_IMO: IMO clock.
  - MCU_CLOCK_EXTCLK: External clock.
  - MCU_CLOCK_ECO: ECO clock.
  - MCU_CLOCK_LPECO: LPECO clock.
  - MCU_CLOCK_ILO0: ILO0 clock.
  - MCU_CLOCK_ILO1: ILO1 clock.
  - MCU_CLOCK_WCO: WCO clock.
  - MCU_CLOCK_ALTHF: ALTHF clock.
  - MCU_CLOCK_ALTLF: ALTLF clock.
  - MCU_CLOCK_DSI<n>: DSI output <n> clock (<n> = 0 … 15).

Note: *Selectable source clocks depend on the subderivative.*

- McuPllReferenceDivision specifies the reference division value of the PLL clock specified by McuPllType.
  - 1 - 20: PLL clock reference division value.
- McuPllOutputDivision specifies the output division value of the PLL clock specified by McuPllType.
  - 2 - 16: PLL clock output division value.
- McuPllFeedbackDivision specifies the feedback division value of the PLL clock specified by McuPllType.

  - 22 - 112: PLL clock feedback division value.
- `McuPllLockSensitivity` specifies the sensitivity of the lock detection of the PLL clock specified by `McuPllType`.
  - `MCU_LOCK_SENSITIVITY_NORMAL`: Normal sensitivity.
  - `MCU_LOCK_SENSITIVITY_REDUCED`: Reduced sensitivity.

4.5.2.4 MCU SSCG PLL clock settings

The `McuSscgPllSettings` container has the following parameters to configure the SSCG PLL clock:

- `McuSscgPllType` specifies the SSCG PLL clock.
  - `MCU_CLOCK_SSCG_PLL<n>`: SSCG PLL<n> clock (<n> = 0 … 14).

Note: *In the same `McuClockSettingConfig` container, `McuSscgPllType` must be unique.*

Note: *Selectable SSCG PLL clock depend on the subderivative.*

- `McuSscgPllEnable` enables or disables the SSCG PLL clock specified by `McuSscgPllType`.
- `McuSscgPllStopForUpdate` enables or disables to stop SSCG PLL clock specified by `McuSscgPllType` once before setting.

Note: *If `McuSscgPllStopForUpdate` is FALSE, setting the SSCG PLL clock specified by `McuSscgPllType` will be skipped when it is running.*

- `McuSscgPllAutoDistributeEnable` enables or disables the automatic distribution of the SSCG PLL clock specified by `McuSscgPllType`.

Note: *If `McuSscgPllAutoDistributeEnable` is TRUE, the SSCG PLL clock specified by `McuSscgPllType` will be automatically distributed after locked and the manual distribution process in `Mcu_DistributePllClock()` will be skipped.*

- `McuSscgPllAutoDistributeType` specifies the automatic distribution type of the SSCG PLL clock specified by `McuPllType`.
  - `MCU_DISTRIBUTE_AFTER_LOCKED`: The SSCG PLL clock specified by `McuSscgPllType` will be automatically distributed after locked. If it becomes unlocked after locked, it will be automatically switched to its reference input clock (bypass mode).
  - `MCU_DISTRIBUTE_ONLY_LOCKED`: The SSCG PLL clock specified by `McuSscgPllType` will be automatically distributed after locked. If it becomes unlocked after locked, it will be gated OFF.
- `McuSscgPllStabilizationTimeout` specifies the timeout count value used when verifying the SSCG PLL clock specified by `McuSscgPllType` has stabilized.
  - 1 - 4294967295: Timeout count value used when verifying that the SSCG PLL clock has stabilized.
- `McuSscgPllFrequency` is the frequency of the SSCG PLL clock (in Hz).

Note: *`McuSscgPllFrequency` automatically displays the resulting frequency calculated by the following formula:*
*`McuSscgPllFrequency` = ((The frequency of the clock specified by `McuSscgPllSource` / `McuSscgPllReferenceDivision`) * `McuSscgPllFeedbackDivision`) / `McuSscgPllOutputDivision`*

Note: *If `McuSscgPllModulationEnable` is TRUE, `McuSscgPllFrequency` displays the average of the modulated frequencies.*

- `McuSscgPllSource` specifies the source clock of the SSCG PLL clock specified by `McuSscgPllType`.
  - `MCU_CLOCK_IMO`: IMO clock.
  - `MCU_CLOCK_EXTCLK`: External clock.
  - `MCU_CLOCK_ECO`: ECO clock.
  - `MCU_CLOCK_LPECO`: LPECO clock.
  - `MCU_CLOCK_ILO0`: ILO0 clock.
  - `MCU_CLOCK_ILO1`: ILO1 clock.
  - `MCU_CLOCK_WCO`: WCO clock.
  - `MCU_CLOCK_ALTHF`: ALTHF clock.
  - `MCU_CLOCK_ALTLF`: ALTLF clock.
  - `MCU_CLOCK_DSI<n>`: DSI output <n> clock (<n> = 0 … 15).

Note: *Selectable source clocks depend on the subderivative.*

- `McuSscgPllReferenceDivision` specifies the reference division value of the SSCG PLL clock specified by `McuSscgPllType`.
  - 1 - 16: SSCG PLL clock reference division value.
- `McuSscgPllOutputDivision` specifies the output division value of the SSCG PLL clock specified by `McuSscgPllType`.
  - 2 - 16: SSCG PLL clock output division value.
- `McuSscgPllFeedbackDivision` specifies the feedback division value of the SSCG PLL clock specified by `McuSscgPllType`.
  - 16.0 - 200.999999940395355: SSCG PLL clock feedback division value.
- `McuSscgPllLockSensitivity` specifies the sensitivity of the lock detection of the SSCG PLL clock specified by `McuSscgPllType`.
  - `MCU_LOCK_SENSITIVITY_INTEGER`: Integer divider mode without spreading.
  - `MCU_LOCK_SENSITIVITY_FRACTIONAL_OR_SPREADING`: Fractional divider mode or spreading mode.
- `McuSscgPllFractionalDivisionEnable` enables or disables the fractional feedback division of the SSCG PLL clock specified by `McuSscgPllType`.
- `McuSscgPllFractionalDivisionDitheringEnable` enables or disables the dithering for the fractional feedback division of the SSCG PLL clock specified by `McuSscgPllType`.
- `McuSscgPllModulationEnable` enables or disables the SSCG modulation of the SSCG PLL clock specified by `McuSscgPllType`.
- `McuSscgPllModulationMode` specifies the SSCG modulation mode of the SSCG PLL clock specified by `McuSscgPllType`.
  - `MCU_SSCG_MODE_DOWN_SPREAD`: Down spread mode.
- `McuSscgPllModulationDepth` specifies the SSCG modulation depth of the SSCG PLL clock specified by `McuSscgPllType` as a percentage of the non-modulated clock.
  - `MCU_SSCG_DEPTH_0_5_PERCENT`: -0.5% for down spread mode.
  - `MCU_SSCG_DEPTH_1_0_PERCENT`: -1.0% for down spread mode.
  - `MCU_SSCG_DEPTH_2_0_PERCENT`: -2.0% for down spread mode.

  - MCU_SSCG_DEPTH_3_0_PERCENT: -3.0% for down spread mode.
- McuSscgPllModulationRate specifies the SSCG modulation rate of the SSCG PLL clock specified by McuSscgPllType.
  - MCU_SSCG_RATE_FPFD_BY_4096: Modulation rate is fPFD / 4096.
  - MCU_SSCG_RATE_FPFD_BY_2048: Modulation rate is fPFD / 2048.
  - MCU_SSCG_RATE_FPFD_BY_1024: Modulation rate is fPFD / 1024.
  - MCU_SSCG_RATE_FPFD_BY_512: Modulation rate is fPFD / 512.
  - MCU_SSCG_RATE_FPFD_BY_256: Modulation rate is fPFD / 256.

Note: *Configuring MCU_SSCG_RATE_FPFD_BY_256 is possible only when fPFD is 8 MHz.*

- McuSscgPllModulationDitheringEnable enables or disables the dithering for the SSCG modulation of the SSCG PLL clock specified by McuSscgPllType.

Note: *McuSscgPllModulationDitheringEnable is not supported and is always disabled.*

4.5.2.5 MCU clock root settings

The McuClockRootSettings container has the following parameters to configure the clock root:

- McuClockRoot specifies the clock root.
  - MCU_CLOCK_ROOT<n>: clock root <n> (<n> = 0 … 15).

Note: *In the same McuClockSettingConfig container, McuClockRoot must be unique.*

Selectable clock roots depend on the subderivative.

- McuClockRootEnable enables or disables the clock root specified by McuClockRoot.

Note: *If McuClockRoot is MCU_CLOCK_ROOT0, McuClockRootEnable must be set to TRUE.*

- McuClockRootFrequency is the frequency of the clock root specified by McuClockRoot (in Hz).

Note: *McuClockRootFrequency automatically displays the resulting frequency calculated by the following formula:*
McuClockRootFrequency = (The frequency of the clock specified by McuClockRootSource) / McuClockRootDivision

- McuClockRootSource specifies the source clock of the current clock root.
  - MCU_CLOCK_FLL: FLL clock.
  - MCU_CLOCK_SSCG_PLL: SSCG PLL clock.
  - MCU_CLOCK_PLL: PLL clock.
  - MCU_CLOCK_PATH: Clock path.
  - MCU_CLOCK_IMO: IMO clock
- McuClockRootSscgPllRef selects the SSCG PLL clock from McuSscgPllSettings to refer as the source clock of the clock root specified by McuClockRoot.

Note: This parameter is available only if `McuClockRootSource` is `MCU_CLOCK_SSCG_PLL`.

- `McuClockRootPllRef` selects the PLL clock from `McuPllSettings` to refer as the source clock of the clock root specified by `McuClockRoot`.

Note: This parameter is available only if `McuClockRootSource` is `MCU_CLOCK_PLL`.

- `McuClockRootPathRef` selects the clock path from `McuClockPathSettings` to refer as the source clock of the clock root specified by `McuClockRoot`.

Note: This parameter is available only if `McuClockRootSource` is `MCU_CLOCK_PATH`.

- `McuClockRootDivision` specifies the division value of the clock root specified by `McuClockRoot`.
  - `MCU_CLK_DIV_1`: Divided by 1.
  - `MCU_CLK_DIV_2`: Divided by 2.
  - `MCU_CLK_DIV_4`: Divided by 4.
  - `MCU_CLK_DIV_8`: Divided by 8.

Note: According to the hardware silicon errata 237, when using B-H-8M and B-H-16M devices, CLK_HF4 and CLK_HF5 must have the same PLL frequency if used together as they share a common clock source.
For restrictions other than MCAL, please refer to the errata.

4.5.2.6 MCU PCLK group settings

The `McuPclkGroupSettings` container has the following parameters to configure the PCLK group:

- `McuPclkGroup` specifies the PCLK group.
  - `MCU_PCLK_GROUP<n>`: PCLK group <n> (<n> = 0 … 15).

Note: Selectable PCLK groups depend on the subderivative.

Note: In the same `McuClockSettingConfig` container, `McuPclkGroup` must be unique.

The MCU PCLK group settings configuration holds the following containers:

- `McuPclkDividerSettings` (see section 4.5.2.7 MCU PCLK divider settings)
- `McuPclkSettings` (see section 4.5.2.8 MCU PCLK settings)

4.5.2.7 MCU PCLK divider settings

The `McuPclkDividerSettings` container has the following parameters to configure the PCLK divider:

- `McuPclkDividerType` specifies the PCLK divider type.
  - `MCU_PCLK_DIVIDER_8`: 8.0 clock divider.
  - `MCU_PCLK_DIVIDER_16`: 16.0 clock divider.
  - `MCU_PCLK_DIVIDER_16_5`: 16.5 clock divider.
  - `MCU_PCLK_DIVIDER_24_5`: 24.5 clock divider.

Note: *Selectable PCLK dividers depend on the subderivative.*

- `McuPclkDividerIndex` specifies the index of the PCLK divider specified by `McuPclkDividerType`.

Note: *In the same `McuClockSettingConfig` container, `McuPclkDividerIndex` must be unique for each PCLK divider type.*

- `McuPclkDividerEnable` enables or disables the PCLK divider.
- `McuPclkDividerStopForUpdate` stops an already running PCLK divider, once, specified by `McuPclkDividerType` and `McuPclkDividerIndex` before setting the clock.

Note: *If `McuPclkDividerStopForUpdate` is FALSE, setting the PCLK divider will be skipped when it is running.*

- `McuPclkDividerValue` specifies the division value of the PCLK divider specified by `McuPclkDividerType` and `McuPclkDividerIndex`.
- Configurable division value depends on the `McuPclkDividerType`.
  - 1 - 256: In case of `MCU_PCLK_DIVIDER_8`.
  - 1 - 65536: In case of `MCU_PCLK_DIVIDER_16`.
  - 1 - 65536.96875: In case of `MCU_PCLK_DIVIDER_16_5`.
  - 1 - 16777216.96875: In case of `MCU_PCLK_DIVIDER_24_5`.

Note: *If `McuPclkDividerType` is `MCU_PCLK_DIVIDER_8` or `MCU_PCLK_DIVIDER_16`, this parameter must be an integer value.*

Note: *If `McuPclkDividerType` is `MCU_PCLK_DIVIDER_16_5` or `MCU_PCLK_DIVIDER_24_5`, the value after the decimal point of this parameter must be five digits or less.*

Note: *If the fractional part of this parameter is not a multiple of 1/32, the value obtained by dividing it by 1/32 is truncated and set to the hardware register. For example, if the value after the decimal point of this parameter is 0.96874, the value obtained by dividing it by 1/32 will be 30.99968 and then the value 30 is set to the hardware register.*

- `McuPclkPhaseAlignDividerRef` selects PCLK divider from `McuPclkDividerSettings` to reference for phase alignment.

Note: *If `McuPclkPhaseAlignDividerRef` is deactivated, the PCLK divider specified by `McuPclkDividerType` and `McuPclkDividerIndex` will be aligned with peripheral clock.*

Note: *The `McuPclkDividerSettings` preceding the current one must be selected.*

4.5.2.8 MCU PCLK settings

The `McuPclkSettings` container has the following parameters to configure the PCLK:

- `McuPclk` specifies the PCLK.
  - `MCU_PCLK_CPUSS_CLOCK_TRACE_IN`: Trace clock.
  - `MCU_PCLK_SMARTIO0_CLOCK`: SMART IO #0.

Note: *Selectable PCLKs depend on the subderivative.*

Note: *In the same `McuPclkGroupSettings` container, `McuPclk` must be unique.*

- `McuPclkEnable` enables or disables the PCLK clock specified by `McuPclk`.
- `McuPclkFrequency` is the frequency of the PCLK specified by `McuPclk` (in Hz).

Note: *`McuPclkFrequency` automatically displays the resulting frequency calculated by the following formula:*
If PCLK divider is in PCLK group 0, `McuPclkFrequency` = `McuPeriClockFrequency`/ (`McuPclkDividerValue` of the PCLK divider selected by `McuPclkDividerRef`)
If PCLK divider is in PCLK group 1, `McuPclkFrequency` = (The value of `McuClockRootFrequency` for which the corresponding `McuClockRoot` is set to `MCU_CLOCK_ROOT2`) / (`McuPclkDividerValue` of the PCLK divider selected by `McuPclkDividerRef`)

- `McuPclkDividerRef` selects PCLK divider from `McuPclkDividerSettings` to refer as the divider of PCLK specified by `McuPclk`.

4.5.2.9 MCU peripheral group settings

The `McuPeriGroupSettings` container has the following parameters to configure the peripheral group:

- `McuPeriGroup` specifies the peripheral group.
  - `MCU_PERI_GROUP<n>_<peripheral group name>`: Peripheral group <n> (<n> = 0 … 15).

Note: *Selectable peripheral groups depend on the subderivative.*

Note: *In the same `McuClockSettingConfig` container, `McuPeriGroup` must be unique.*

Note: *The configuration of the `McuClockSettingConfig` container may affect access to the hardware register with access restrictions such as the PERI_GR2_SL_CTL register protected by PPU.*
For more information about the hardware registers with access restrictions, see the hardware technical reference manual.

- `McuPeriGroupClockFrequency` is the frequency of the peripheral group clock specified by `McuPeriGroup` (in Hz).

Note: *`McuPeriGroupClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
If `McuPeriGroup` starts with `MCU_PERI_GROUP0_`, `MCU_PERI_GROUP1_`, or `MCU_PERI_GROUP2_`, then `McuPeriGroupClockFrequency` = `McuSlowClockFrequency`.
If `McuPeriGroup` starts with groups other than above, i.e..: `MCU_PERI_GROUP3_`, `MCU_PERI_GROUP4_`, or `MCU_PERI_GROUP5_` and so on, then `McuPeriGroupClockFrequency` = `McuPeriClockFrequency`/ `McuPeriGroupClockDivision` or `McuPeriGroupClockFrequency` = (The value of `McuClockRootFrequency` for which the corresponding `McuClockRoot` is set to `MCU_CLOCK_ROOT2`)/ `McuPeriGroupClockDivision`.

- `McuPeriGroupClockDivision` specifies the division value of the peripheral group clock specified by `McuPeriGroup`.
  - 1 - 20: Peripheral group clock division value.

MCU peripheral group settings configuration holds the following containers:

- `McuPeriGroupSlaveSettings` (see section 4.5.2.10 MCU peripheral group slave settings)

4.5.2.10 MCU peripheral group slave settings

The `McuPeriGroupSlaveSettings` container has the following parameters to configure the slave of the peripheral group:

- `McuPeriGroupSlaveName`
  - `MCU_PERI_GROUP<n>_SLAVE<m>_<peripheral group slave name>`: Slave <m> of the peripheral group <n> (<n> = 0 … 15, <m> = 0 … 15).

Note: *Selectable peripheral group slaves depend on the subderivative.*

Note: *In the same `McuPeriGroupSettings` container, `McuPeriGroupSlaveName` must be unique.*

- `McuPeriGroupSlaveEnable` enables or disables the slave of the peripheral group specified by `McuPeriGroupSlaveName`.

Note: *If `McuPeriGroupSlaveName` starts with `MCU_PERI_GROUP0_SLAVE0_` or `MCU_PERI_GROUP0_SLAVE1_`, `McuPeriGroupSlaveEnable` must be set to TRUE. Also, if `McuPeriGroupSlaveName` starts with `MCU_PERI_GROUP0_SLAVE2_`, then `McuPeriGroupSlaveEnable` must be set to TRUE if the device supports Arm® Cortex®-M7 CPU.*

4.5.2.11 MCU LF clock settings

The `McuLfClockSettings` container has the following parameters to configure the LF clock:

- `McuLfClockFrequency` is the frequency of the LF clock (in Hz).

Note: *`McuLfClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
`McuLfClockFrequency` = (The frequency of the clock specified by `McuLfClockSource`)

- `McuLfClockSource` specifies the source clock of the LF clock.
  - `MCU_CLOCK_ILO0`: ILO0 clock.
  - `MCU_CLOCK_ILO1`: ILO1 clock.
  - `MCU_CLOCK_ECO_PRESCALE`: Prescaled ECO clock.
  - `MCU_CLOCK_LPECO_PRESCALE`: Prescaled LPECO clock.
  - `MCU_CLOCK_WCO`: WCO clock.
  - `MCU_CLOCK_ALTLF`: ALTLF clock.

Note: *`MCU_CLOCK_ECO_PRESCALE` must not be set to `McuLfClockSource` when the configuration is used for DeepSleep mode.*

4.5.2.12 MCU pump clock settings

The `McuPumpClockSettings` container has the following parameters to configure the pump clock:

Note: *`McuPumpClockSettings` is not supported and is always disabled.*

- `McuPumpClockEnable` enables or disables the pump clock.
- `McuPumpClockStopForUpdate` stops a running pump clock before setting the clock.

Note: *If `McuPumpClockStopForUpdate` is FALSE, setting the pump clock will be skipped when it is running.*

- `McuPumpClockFrequency` is the frequency of the pump clock (in Hz).

Note: *`McuPumpClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
`McuPumpClockFrequency` = The frequency of the clock specified by `McuPumpClockSource`/ `McuPumpClockDivision`.

- `McuPumpClockSource` specifies the source clock of the pump clock.
  - `MCU_CLOCK_FLL`: FLL clock.
  - `MCU_CLOCK_SSCG_PLL`: SSCG PLL clock.
  - `MCU_CLOCK_PLL`: PLL clock.
  - `MCU_CLOCK_PATH`: Clock path.
- `McuPumpClockSscgPllRef` selects the SSCG PLL clock from `McuSscgPllSettings` as the source clock of the pump clock.
- `McuPumpClockPllRef` selects the PLL clock from `McuPllSettings` as the source clock of the pump clock.
- `McuPumpClockPathRef` selects the clock path from `McuClockPathSettings` as the source clock of the pump clock.
- `McuPumpClockDivision` specifies the division value of the pump clock.
  - `MCU_CLK_DIV_1`: Divided by 1.
  - `MCU_CLK_DIV_2`: Divided by 2.
  - `MCU_CLK_DIV_4`: Divided by 4.
  - `MCU_CLK_DIV_8`: Divided by 8.
  - `MCU_CLK_DIV_16`: Divided by 16.

4.5.2.13 MCU timer clock settings

The `McuTimerClockSettings` container has the following parameters to configure the timer clock:

- `McuTimerClockEnable` enables or disables the timer clock.
- `McuTimerClockStopForUpdate` stops the running timer clock before setting the clock.

Note: *If `McuTimerClockStopForUpdate` is FALSE, setting the timer clock will be skipped when it is running.*

- `McuTimerClockFrequency` is the frequency of the timer clock (in Hz).

Note: *`McuTimerClockFrequency` automatically displays the resulting frequency calculated by the following formula:*
If `McuTimerClockSource` is `MCU_CLOCK_IMO`, then `McuTimerClockFrequency` = `McuImoFrequency` / `McuTimerClockDivision`.
If `McuTimerClockSource` is `MCU_CLOCK_HF0DIV`, then `McuTimerClockFrequency` = ((The value of `McuClockRootFrequency` for which the corresponding `McuClockRoot` is set to `MCU_CLOCK_ROOT0`) / `McuTimerClockInputDivision`) / `McuTimerClockDivision`.

- `McuTimerClockSource` specifies the source clock of the timer clock.
  - `MCU_CLOCK_HF0DIV`: HF0 (clock root 0) clock divided by `McuTimerClockInputDivision`.
  - `MCU_CLOCK_IMO`: IMO clock.
- `McuTimerClockInputDivision` specifies the HF0 clock division value for the source clock of the timer clock.
  - `MCU_CLK_DIV_1`: Divided by 1.
  - `MCU_CLK_DIV_2`: Divided by 2.
  - `MCU_CLK_DIV_4`: Divided by 4.
  - `MCU_CLK_DIV_8`: Divided by 8.

Note: *This parameter is available only if `McuTimerClockSource` is `MCU_CLOCK_HF0DIV`.*

- `McuTimerClockDivision` specifies the division value of the timer clock.
  - 1 - 256: Timer clock division value.

4.5.2.14 MCU clock output settings

The `McuClockOutputSettings` container has the following parameters to configure the clock output:

Note: The clock output function uses the same hardware registers as the clock calibration functions. Therefore, enabling `McuClockOutputSettings` may cause unexpected behavior of the clock calibration function.

- `McuClockOutput0Enable` enables or disables the clock output 0.

Note: Because the clock output function enabled by `McuClockOutput0Enable` is for testing purposes only, `McuClockOutput0Enable` must not be set TRUE for production.

Note: A warning message will be reported if `McuClockOutput0Enable` is TRUE. This message indicates that the configuration in PORT module for the port pin used by the clock output 0 function will be ignored.

- `McuClockOutput0Frequency` is the frequency of the clock output 0 (in Hz).

Note: `McuClockOutput0Frequency` automatically displays the resulting frequency calculated by the following formula:
`McuClockOutput0Frequency` = (The frequency of the clock specified by `McuClockOutput0Source`) / `McuClockOutput0Division`.

- `McuClockOutput0Source` specifies the source clock of the clock output 0.
  - `MCU_CLOCK_LOW`: Disabled and output is fixed low.
  - `MCU_CLOCK_ECO`: ECO clock.

Note: Selectable source clocks depend on the subderivative.

- `McuClockOutput0Division` specifies the division value of the clock output 0.
  - `MCU_CLK_DIV_1`: Divided by 1.
  - `MCU_CLK_DIV_2`: Divided by 2.
  - `MCU_CLK_DIV_4`: Divided by 4.
  - `MCU_CLK_DIV_8`: Divided by 8.
- `McuClockOutput1Enable` enables or disables the clock output 1.

Note: Because the clock output function enabled by `McuClockOutput1Enable` is for testing purposes only, `McuClockOutput1Enable` must not be set TRUE for production.

Note: A warning message will be reported if `McuClockOutput1Enable` is TRUE. This message indicates that the configuration in PORT module for the port pin used by the clock output 1 function will be ignored.

- `McuClockOutput1Frequency` is the frequency of the clock output 1 (in Hz).

Note: *`McuClockOutput1Frequency` automatically displays the resulting frequency calculated by the following formula:*
`McuClockOutput1Frequency` = (The frequency of the clock specified by `McuClockOutput1Source`) / `McuClockOutput1Division`.

- `McuClockOutput1Source` specifies the source clock of the clock output 1.
  - `MCU_CLOCK_LOW`: Disabled and output is fixed LOW.
  - `MCU_CLOCK_ECO`: ECO clock.

Note: *Selectable source clocks depend on the subderivative.*

- `McuClockOutput1Division` specifies the division value of the clock output 1.
  - `MCU_CLK_DIV_1`: Divided by 1.
  - `MCU_CLK_DIV_2`: Divided by 2.
  - `MCU_CLK_DIV_4`: Divided by 4.
  - `MCU_CLK_DIV_8`: Divided by 8.

4.5.2.15 MCU clock supervisor settings

The `McuCsvSettings` container has the following parameters to configure the clock supervisor:

- `McuCsvClock` specifies the monitoring clock of the clock supervisor.
  - `MCU_CLOCK_CSVREF`: Reference clock of the clock supervisor.
  - `MCU_CLOCK_LF`: LF clock.
  - `MCU_CLOCK_ILO0`: ILO0 clock.
  - `MCU_CLOCK_BACKUP`: Backup clock.
  - `MCU_CLOCK_ROOT<n>`: clock root <n> (<n> = 0 … 15).

Note: *Selectable monitoring clocks depend on the subderivative.*

Note: *In the same `McuClockSettingConfig` container, `McuCsvClock` must be unique.*

- `McuCsvEnable` enables or disables the clock supervisor specified by `McuCsvClock`.

Note: *If this parameter is TRUE, monitoring clock and reference clock must be enabled.*

- `McuCsvPeriod` specifies the number of monitored clock cycles within a period.
  - 1 - 256: In case of `MCU_CLOCK_LF` and `MCU_CLOCK_ILO0`.
  - 1 - 65536: In case of `MCU_CLOCK_CSVREF` and `MCU_CLOCK_ROOT<n>`.
- `McuCsvStartupDelay` specifies the startup delay of the clock supervisor in reference clock cycles.
  - 1 - 256: In case of `MCU_CLOCK_LF` and `MCU_CLOCK_ILO0`.
  - 1 - 512: In case of `MCU_CLOCK_LF` and `MCU_CLOCK_ILO0` and `MCU_CLOCK_BACKUP`.
  - 1 - 65536: In case of `MCU_CLOCK_CSVREF` and `MCU_CLOCK_ROOT<n>`.

Note: *The valid range of `MCU_CLOCK_LF` and `MCU_CLOCK_ILO0` is device specific. About the details, see the hardware register technical reference.*

- McuCsvLowerLimit specifies the lower limit of the clock supervisor in reference clock cycles.
  - 1 - 256: In case of MCU_CLOCK_LF and MCU_CLOCK_ILO0.
  - 1 - 65536: In case of MCU_CLOCK_CSVREF and MCU_CLOCK_ROOT<n>.

Note: *McuCsvLowerLimit must be less than McuCsvUpperLimit - 1.*

- McuCsvUpperLimit specifies the upper limit of the clock supervisor in reference clock cycles.
  - 1 - 256: In case of MCU_CLOCK_LF and MCU_CLOCK_ILO0.
  - 1 - 65536: In case of MCU_CLOCK_CSVREF and MCU_CLOCK_ROOT<n>.
- McuCsvAction specifies the action executed when the error is detected by the clock supervisor specified by McuCsvClock.
  - MCU_CSV_ACTION_FAULT: Fault report.
  - MCU_CSV_ACTION_RESET: Reset.

Note: *When MCU_CSV_ACTION_FAULT is configured, you should handle the fault report of the clock supervisor.*

4.5.3 MCU clock reference point

The McuClockReferencePoint container has the following parameters to configure the clock references:

- McuClock selects the clock type for this clock reference point.
  - MCU_CLOCK_IMO: IMO clock.
  - MCU_CLOCK_ECO: ECO clock.

Note: *Selectable clocks depend on the subderivative.*

- McuClockReferencePointFrequency specifies the clock frequency of the selected by McuClock (in Hz). It is referenced by other modules.
- McuClockReferencePointFrequency will display the resulting frequency. These settings are evaluated and displayed in the resulting clock frequencies. This value will be assigned to the following definitions:
  - The definitions derived from the McuModuleConfiguration container short name, the McuClockSettingConfig container short name, McuClockReferencePoint container short name, and the McuClock parameter value are concatenated with "_" and prefixed with "MCU_".

Example:
MCU_McuModuleConfiguration_0_McuClockSettingConfig_0_McuClockReferencePoint_0_MCU_CLOCK_IMO.

4.6 MCU mode settings configuration

The McuModeSettingConf container has the following parameters to configure the mode settings:

- McuMode is a logical ID of the mode setting. This value will be assigned to the following symbolic names:
  - The symbolic name derived from the McuModeSettingConf container short name is prefixed with "McuConf_McuModeSettingConf_".

Example:
McuConf_McuModeSettingConf_McuModeSettingConf_0.

Note: In the same McuModuleConfiguration container, McuMode must be unique and consecutive.

- McuModeCoreAssignment specifies the reference to the McuCoreConfiguration for assigning the core to McuModeSettingConf.

Note: McuModeCoreAssignment must have the valid reference to the McuCoreConfiguration.

Note: McuCoreConfiguration corresponding to the value of McuTargetCpu must be configured to this parameter.

Note: The value of McuModeCoreAssignment must be same for all McuModeSettingConf with McuUpdateSystemResource whose value is set to TRUE.

- McuTargetCpu specifies the CPU which applies the mode specified by McuCpuPowerMode.
  - MCU_CPU_CM0P: Arm® Cortex®-M0+ CPU
  - MCU_CPU_CM4: Arm® Cortex®-M4 CPU
  - MCU_CPU_CM7_0: Arm® Cortex®-M7 CPU 0
  - MCU_CPU_CM7_1: Arm® Cortex®-M7 CPU 1
  - MCU_CPU_CM7_2: Arm® Cortex®-M7 CPU 2
  - MCU_CPU_CM7_3: Arm® Cortex®-M7 CPU 3

Note: The mode setting must be applied on the CPU specified by McuTargetCpu.

- McuCpuPowerMode specifies the CPU power mode.
  - MCU_CPUMODE_ACTIVE: CPU Active mode.
  - MCU_CPUMODE_SLEEP: CPU Sleep mode.
  - MCU_CPUMODE_DEEPSLEEP: CPU Deep Sleep mode.
  - MCU_CPUMODE_HIBERNATE: System Hibernate mode

Note: To set to low-power mode, you must set all cores to Sleep or DeepSleep mode.

- McuEnableLowPowerTransition specifies whether enter the low-power state or not.
- McuMainCore0PowerMode specifies the power mode of the Main Core 0 CPU power domain.
  - MCU_POWERMODE_ENABLED: Switch ON.
  - MCU_POWERMODE_OFF: Switch OFF.
  - MCU_POWERMODE_RESET: Reset.

- MCU_POWERMODE_RETAINED: Put in retained mode.

Note: This parameter is available only if McuTargetCpu is MCU_CPU_CM0P.

Note: MCU_POWERMODE_RETAINED can be effective only when Main Core 0 is in CPU DeepSleep mode.

- McuMainCore1PowerMode specifies the power mode of the Main Core 1 CPU power domain.
  - MCU_POWERMODE_ENABLED: Switch ON.
  - MCU_POWERMODE_OFF: Switch OFF.
  - MCU_POWERMODE_RESET: Reset.
  - MCU_POWERMODE_RETAINED: Put in retained mode.

Note: This parameter is available only if McuTargetCpu is MCU_CPU_CM0P and the target device has an Arm® Cortex®-M7 CPU 1.

Note: MCU_POWERMODE_RETAINED can be effective only when Main Core 1 is in CPU DeepSleep mode.

- McuMainCore2PowerMode specifies the power mode of the main core 2 CPU power domain.
  - MCU_POWERMODE_ENABLED: Switch ON.
  - MCU_POWERMODE_OFF: Switch OFF.
  - MCU_POWERMODE_RESET: Reset.
  - MCU_POWERMODE_RETAINED: Put in retained mode.

Note: This parameter is available only if McuTargetCpu is MCU_CPU_CM0P and the target device has an Arm® Cortex®-M7 CPU 2.

Note: MCU_POWERMODE_RETAINED can be effective only when main core 2 is in CPU DeepSleep mode.

- McuMainCore3PowerMode specifies the power mode of the main core 3 CPU power domain.
  - MCU_POWERMODE_ENABLED: Switch ON.
  - MCU_POWERMODE_OFF: Switch OFF.
  - MCU_POWERMODE_RESET: Reset.
  - MCU_POWERMODE_RETAINED: Put in retained mode.

Note: This parameter is available only if McuTargetCpu is MCU_CPU_CM0P and the target device has an Arm® Cortex®-M7 CPU 3.

Note: MCU_POWERMODE_RETAINED can be effective only when main core 3 is in CPU DeepSleep mode.

- McuSleepOnExitIsrEnable enables or disables the CPU entering Sleep state on exiting from an ISR.
- McuWakeupByPendingInterruptEnable enables or disables the CPU waking up by an interrupt transition from an inactive state to the pending state.
- McuEnableCacheFlushBeforeModeChange enables or disables flushing cache before changing mode.

Note: If this parameter is TRUE, the stack and static data of the MCU driver must be allocated to a non-cached memory area.

- `McuRamWriteBufferTimeout` specifies the timeout count value used when checking whether the RAM write buffer status is empty.
  - 1 - 4294967295: Timeout count value used when verifying that the RAM write buffer is empty.
- `McuFreezeIoRelease` enables or disables releasing the I/O freeze.

Note: *If I/O freeze is enabled when entering Hibernate mode, after wakeup, I/O freeze should be released by applying the mode configuration with this parameter set to TRUE.*

- `McuUpdateSystemResource` specifies whether to update the system resources or not.

Note: *At least one McuModeSettingConf with this parameter whose value is set to TRUE must be configured in order to assign the system resources to the core.*

The following parameters are related to system resources controlled by this parameter. If this parameter is FALSE, the following parameters are not applied. The system resources should be updated from only one (master) CPU core.

  - `McuReferenceClockSetting`
  - `McuLinearCoreRegulatorDisable`
  - `McuLinearCoreRegulatorEnableTimeout`
  - `McuDeepSleepRegulatorDisable`
  - `McuVoltageReferenceBufferDisable`
  - `McuVoltageReferenceBufferReadyTimeout`
  - `McuReferenceCurrentGeneratorDisable`
  - `McuReferenceCurrentGeneratorEnableTimeout`
  - `McuBandgapReferencePowerMode`
  - `McuBypassPllLevelShifter`
  - `McuHvLvdSettings`
- `McuReferenceClockSetting` selects the clock setting configuration from `McuClockSettingConfig`, which is applied to its mode configuration.

Note: *The value of `McuClockCoreAssignment` included in `McuClockSettingConfig` referenced by this parameter must be same as the value of `McuModeCoreAssignment`.*

- `McuMainCore0PowerUpDelay` specifies the delay after power up of Main Core 0 power domain in clock cycles.
  - 0 - 1023: Delay count in cycles.
- `McuMainCore1PowerUpDelay` specifies the delay after power up of Main Core 1 power domain in clock cycles.
  - 0 - 1023: Delay count in cycles.
- `McuMainCore2PowerUpDelay` specifies the delay after power up of main core 2 power domain in clock cycles.
  - 0 - 1023: Delay count in cycles.
- `McuMainCore3PowerUpDelay` specifies the delay after power up of main core 3 power domain in clock cycles.
  - 0 - 1023: Delay count in cycles.

- McuRam0Macro<n>PowerMode (<n> = 0 … 15) specifies the RAM0 Macro <n> power mode.
  - MCU_POWERMODE_OFF: Switch OFF.
  - MCU_POWERMODE_RETAINED: Put in retained mode.
  - MCU_POWERMODE_ENABLED: Switch ON.

Note: *Selectable RAM0 macros depend on the subderivative.*

Note: *Some SRAM areas may be used by the SROM API. Therefore, the power of those SRAM areas should not be disabled when the SROM API is used. If some of the SRAM0 areas are used by the SROM API, McuRam0Macro<n>PowerMode corresponding to those areas should not be configured to MCU_POWERMODE_OFF.*

Note: *If this parameter is MCU_POWERMODE_OFF or MCU_POWERMODE_RETAINED, the stack and static data of the MCU driver must not be allocated to the SRAM0 area corresponding to the RAM0 macro <n>.*

- McuRam1PowerMode specifies the RAM1 power mode.
  - MCU_POWERMODE_OFF: OFF mode
  - MCU_POWERMODE_RETAINED: Retained mode
  - MCU_POWERMODE_ENABLED: ON mode

Note: *Some SRAM areas may be used by the SROM API. Therefore, the power of those SRAM areas should not be disabled when the SROM API is used. If the SRAM1 areas are used by the SROM API, McuRam1PowerMode should not be configured to MCU_POWERMODE_OFF.*

Note: *If this parameter is MCU_POWERMODE_OFF or MCU_POWERMODE_RETAINED, the stack and static data of the MCU driver must not be allocated to the SRAM1 area.*

- McuRam2PowerMode specifies the RAM2 power mode.
  - MCU_POWERMODE_OFF: OFF mode
  - MCU_POWERMODE_RETAINED: Retained mode
  - MCU_POWERMODE_ENABLED: ON mode

Note: *Some SRAM areas may be used by the SROM API. So, the power of those SRAM areas should not be disabled when the SROM API is used. If the SRAM2 areas are used by the SROM API, McuRam2PowerMode should not be configured to MCU_POWERMODE_OFF.*

Note: *If this parameter is MCU_POWERMODE_OFF or MCU_POWERMODE_RETAINED, the stack and static data of the MCU driver must not be allocated to the SRAM2 area.*

- McuRamPowerUpDelay specifies the delay after power up of all RAM power domain in cycles.
  - 0 - 1023: Delay count in cycles.
- McuLowPowerReadyTimeout specifies the timeout count value used when verifying that low-power functions are ready.
  - 1 - 4294967295: Timeout count value used when verifying that the low-power functions are ready.
- McuLinearCoreRegulatorDisable enables or disables the linear core regulator.

Note: *This parameter must be set to FALSE; otherwise an error would occur in the configuration phase.*

- `McuLinearCoreRegulatorEnableTimeout` specifies the timeout count value used when verifying that the linear core regulator is ready.
  - 1 - 4294967295: Timeout count value used when verifying that the linear core regulator is ready.
- `McuDeepSleepRegulatorDisable` disables or enables the DeepSleep regulator.

Note: *If this parameter is TRUE, the DeepSleep regulator will be disabled. Once the DeepSleep regulator is disabled, it will not be enabled again later.*

Note: *It cannot be used on some derivates.*

- `McuVoltageReferenceBufferDisable` enables or disables the voltage reference buffer.

Note: *If this parameter is TRUE, the voltage reference buffer will be disabled.*

Note: *Do not call Mcu_SetMode API with a Mode config with this parameter set to TRUE while using ECO and/or PLL.*

- `McuVoltageReferenceBufferReadyTimeout` specifies the timeout count value used when verifying that the voltage reference buffer is ready.
  - 1 - 4294967295: Timeout count value used when verifying that the voltage reference buffer is ready.
- `McuReferenceCurrentGeneratorDisable` disables or enables the reference current generator.

Note: *If this parameter is TRUE, the reference current generator will be disabled.*

Note: *This parameter must be set to FALSE.*

- `McuReferenceCurrentGeneratorEnableTimeout` specifies the timeout count value used when verifying that the reference current generator is ready.
  - 1 - 4294967295: Timeout count value used when verifying that the reference current generator is ready.
- `McuBandgapReferencePowerMode` specifies the power mode of the bandgap reference circuits.
  - `MCU_POWERMODE_NORMAL`: Normal mode
  - `MCU_POWERMODE_LOWPOWER`: Low-power mode

Note: *ILO0 is required to be active for proper operation of `MCU_POWERMODE_LOWPOWER`. When switching from `MCU_POWERMODE_LOWPOWER` to `MCU_POWERMODE_NORMAL`, ILO0 needs to stay active for at least five more clock cycles.*

- `McuBypassPllLevelShifter` specifies whether bypass level shifter is inside the PLL or not.

MCU mode settings configuration holds the following containers.

- `McuHibernateSettings` (see section 4.6.1 MCU hibernate mode settings)
- `McuSupplySupervisionSettings` (see section 4.6.3 MCU supply supervision settings)
- `McuHvLvdSettings` (see section 4.6.2 MCU HVLVD settings)
- `McuRegHcSettings` (see section 4.6.4 MCU REGHC settings)

- McuPmicSettings (see section 4.6.5 MCU PMIC settings)
- McuDmaSettings (see section 4.6.6 MCU DMA settings)

4.6.1 MCU hibernate mode settings

The McuHibernateSettings container has the following parameters to configure Hibernate mode:

- McuHibernateClearPendingWakeup enables or disables clearing the pending wakeup.

Note: *If McuHibernateClearPendingWakeup is TRUE, all wakeup causes are cleared regardless of the value of McuEnableLowPowerTransition.*

- McuHibernateFreezeIoEnable enables or disables the I/O freeze when entering Hibernate mode.
- McuHibernateWakeupByBackupAlarmEnable enables or disables the wakeup from Hibernate mode by an RTC interrupt.
- McuHibernateWakeupByWatchdogEnable enables or disables the wake up from Hibernate mode by a WDT.
- McuHibernateWakeupByBackupCsvEnable enables or disables the wake up from Hibernate mode by a backup clock supervisor.
- McuHibernateWakeupSenseMode enables or disables the wake up from Hibernate mode by the pending interrupt.
- McuHibernateWakeupByWakeupPin<n>Enable (<n> = 0 … 23) enables or disables the wakeup from Hibernate mode by the wakeup pin input. The wakeup will occur when its input matches McuHibernateWakeupPin<n>Polarity.
- McuHibernateWakeupPin<n>Polarity (<n> = 0 … 23) specifies the active polarity of the corresponding wakeup pin.
  - MCU_PIN_POLARITY_LOW: Pin input of 0 will trigger the wakeup from Hibernate mode.
  - MCU_PIN_POLARITY_HIGH: Pin input of 1 will trigger the wakeup from Hibernate mode.

Note: *This container is available only if McuCpuPowerMode is MCU_CPUMODE_HIBERNATE.*

4.6.2 MCU HVLVD settings

The McuHvLvdSettings container has the following parameters to configure the HVLVD:

- McuHvLvdType specifies the HVLVD type.
  - MCU_HVLVD_HVLVD1: HVLVD1
  - MCU_HVLVD_HVLVD2: HVLVD2

Note: *In the same McuModeSettingConf container, McuHvLvdType must be unique.*

- McuHvLvdEnable enables or disables the HVLVD.
- McuHvLvdOnDeepSleepEnable keeps the HVLVD specified by McuHvLvdType enabled during DeepSleep mode.

Note: *If this parameter is TRUE, McuHvLvdEnable must be TRUE.*

- McuHvLvdStopForUpdate stops the HVLVD specified by McuHvLvdType once before setting the HDLVD by configuration.

Note: If McuHvLvdStopForUpdate is FALSE, setting the HVLVD will be skipped when it is running.

- McuHvLvdThreshold specifies the threshold value of the HVLVD specified by McuHvLvdType.
  - MCU_HVLVD_THRESHOLD_2_8V_TO_2_825V: 2.8 [V] to 2.825 [V]
  - MCU_HVLVD_THRESHOLD_2_9V_TO_2_925V: 2.9 [V] to 2.925 [V]
- McuHvLvdAction specifies the action executed when the error is detected by the HVLVD specified by McuHvLvdType.
  - MCU_LVD_ACTION_FAULT: Fault report.
  - MCU_LVD_ACTION_INTERRUPT: Interrupt.

Note: When MCU_LVD_ACTION_FAULT is configured, you should handle the fault report of HVLVD.

- McuHvLvdInterruptEnable enables or disables the interrupt of the HVLVD specified by McuHvLvdType.
- McuHvLvdTriggerEdge specifies the edge which triggers an action when the threshold is crossed.
  - MCU_HVLVD_EDGE_RISING: Rising edge.
  - MCU_HVLVD_EDGE_FALLING: Falling edge.
  - MCU_HVLVD_EDGE_BOTH: Both edges.

4.6.3 MCU supply supervision settings

The McuSupplySupervisionSettings container has the following parameters to configure supply supervision:

- McuVdddBodEnable enables or disables BOD on VDDD.

Note: The BOD on VDDD cannot be disabled, so this parameter is always TRUE.

- McuVdddBodThreshold specifies the threshold value of BOD on VDDD.
  - MCU_BOD_THRESHOLD_2_7V: 2.7 [V]
  - MCU_BOD_THRESHOLD_3_0V: 3.0 [V]
- McuVddaBodEnable enables or disables BOD on VDDA.
- McuVddaBodThreshold specifies the threshold value of BOD on VDDA.
  - MCU_BOD_THRESHOLD_2_7V: 2.7 [V]
  - MCU_BOD_THRESHOLD_3_0V: 3.0 [V]
- McuVddaBodAction specifies BOD on VDDA action.
  - MCU_BOD_ACTION_NONE: No action.
  - MCU_BOD_ACTION_RESET: Reset.
  - MCU_BOD_ACTION_FAULT: Fault report.

Note: When MCU_BOD_ACTION_FAULT is configured, you should handle the fault report of BOD.

- McuVccdBodEnable enables or disables BOD on VCCD.

Note: *BOD on VCCD cannot be disabled, so this parameter is always TRUE.*

- `McuVdddOvdEnable` enables or disables OVD on VDDD.

Note: *OVD on VDDD cannot be disabled, so this parameter is always TRUE.*

- `McuVdddOvdThreshold` specifies the threshold value of OVD on VDDD.
  - `MCU_OVD_THRESHOLD_5_0V`: 5.0 [V]
  - `MCU_OVD_THRESHOLD_5_5V`: 5.5 [V]
- `McuVddaOvdEnable` enables or disables OVD on VDDA.
- `McuVddaOvdThreshold` specifies the threshold value of OVD on VDDA.
  - `MCU_OVD_THRESHOLD_5_0V`: 5.0 [V]
  - `MCU_OVD_THRESHOLD_5_5V`: 5.5 [V]
- `McuVddaOvdAction` specifies OVD on VDDA action.
  - `MCU_OVD_ACTION_NONE`: No action.
  - `MCU_OVD_ACTION_RESET`: Cause a reset.
  - `MCU_OVD_ACTION_FAULT`: Cause a fault report.

Note: *When `MCU_OVD_ACTION_FAULT` is configured, you should handle the fault report of OVD.*

- `McuVccdOvdEnable` enables or disables OVD on VCCD.

Note: *OVD on VCCD cannot be disabled, so this parameter is always TRUE.*

4.6.4 MCU REGHC settings

The `McuRegHcSettings` container has the following parameters to configure the REGHC:

Note: *The usage of this functionality in MCAL is prohibited; otherwise an error would occur in the configuration phase. For the implementation of REGHC, see AN226698 - External Power Supply Design Guide for TRAVEO™ T2G family.*

The following parameters are no longer valid:

- `McuRegHcEnable` enables or disables the REGHC.
- `McuRegHcOnDeepSleepEnable` keeps the REGHC enabled during DeepSleep mode.
- `McuRegHcPmicVadjDisable` disables or enables the PMIC VADJ for REGHC.
- `McuRegHcStabilizationTimeout` specifies the timeout count value used when checking whether the REGHC is stabilized.
  - 1 - 4294967295: Timeout count value used when checking whether the REGHC is stabilized.

4.6.5 MCU PMIC settings

The `McuPmicSettings` container has the following parameters to configure the PMIC:

Note: *The usage of this functionality in MCAL is prohibited; otherwise an error would occur in the configuration phase. For the implementation of PMIC, see AN226698 - External power supply design guide for TRAVEO™ T2G family.*

The following parameters are no longer valid.

- `McuPmicEnable` enables or disables the PMIC.
- `McuPmicOnDeepSleepEnable` keeps the PMIC enabled during DeepSleep mode.
- `McuPmicVadjDisable` disables or enables the PMIC VADJ.
- `McuPmicStabilizationTimeout` specifies the timeout count value used when checking whether the PMIC is stabilized.
  - 1 - 4294967295: Timeout count value used when checking whether the PMIC is stabilized.

4.6.6 MCU DMA settings

The `McuDmaSettings` container has the following parameters to configure DMA:

- `McuDmaEnable` enables or disables DMA.
- `McuDataWire0Enable` enables or disables the Data Wire 0.
- `McuDataWire1Enable` enables or disables the Data Wire 1.

4.7 MCU RAM section configuration

The `McuRamSectorSettingConf` container has the following parameters to configure RAM section settings:

- `McuRamCoreAssignment` specifies the reference to the `McuCoreConfiguration` for assigning the core to `McuRamSectorSettingConf`.

Note: *`McuRamCoreAssignment` must have the valid reference to the `McuCoreConfiguration`.*

- `McuRamSectionBaseAddress` specifies the address where the RAM section to initialize starts.
- `McuRamSectionSize` specifies the size of this RAM section.
- `McuRamDefaultValue` specifies the initialization value (8 bits) for the RAM section.

4.8 MCU multicore

- `McuCoreConsistencyCheckEnable` enables core consistency check during run time. If enabled, the MCU function checks whether the parameter provided (clock, mode, RAM sector) is allowed on the current core.

Note: *Development error detect must be enabled in MCU driver to enable this parameter.*

- `McuGetCoreIdFunction` specifies the API to be called to get the core ID. For example, `GetCoreID()`.

Note: *`McuGetCoreIdFunction` must be a valid C function name.*

- `McuMasterCoreReference` specifies the reference to the master core configuration.

Note: *`McuMasterCoreReference` must have the reference to the valid `McuCoreConfiguration`.*

MCU multicore configuration holds the following containers:

- `McuCoreConfiguration` (see section 4.8.1 MCU core configuration)

4.8.1 MCU core configuration

- `McuCoreConfigurationId` is a zero-based, consecutive integer value. This is used as a logical core ID.

Note: *`McuCoreConfigurationId` must be unique across `McuCoreConfiguration`.*

- `McuCoreId` is core ID assigned to clocks, modes, and RAM sectors. This ID is returned from configured `McuGetCoreIdFunction` execution to identify the executing core.

Note: *`McuCoreId` must be unique across `McuCoreConfiguration`.*
The combination of `McuCoreConfigurationId` and `McuCoreId` must be unique across `McuCoreConfiguration`.

Note: *`McuCoreConfiguration` can also be configured without MCU resource assignment.*

4.9 MCU published information

The `McuPublishedInformation` container has different types of reset reasons that can be retrieved from the `Mcu_GetResetReason()` API. This container is not editable. The `McuResetReason` values are assigned to the following symbolic name.

The symbolic name derived from the `McuResetReasonConf` container short name is prefixed with "`McuConf_McuResetReasonConf_`".

Example:
`McuConf_McuResetReasonConf_MCU_RESET_UNDEFINED.`

Table 2 List of reset reasons

Container	McuResetReason value
MCU_RESET_UNDEFINED	0
MCU_POWER_ON_RESET	1
MCU_WATCHDOG_RESET	2
MCU_ACT_FAULT_RESET	3
MCU_DPSLP_FAULT_RESET	4
MCU_TEST_DEBUG_RESET	5
MCU_SW_RESET	6
MCU_MCWDT0_RESET	7
MCU_MCWDT1_RESET	8
MCU_MCWDT2_RESET	9
MCU_MCWDT3_RESET	10
MCU_XRES_RESET	11
MCU_BOD_VDDD_RESET	12

Container	McuResetReason value
MCU_BOD_VDDA_RESET	13
MCU_BOD_VCCD_RESET	14
MCU_OVD_VDDD_RESET	15
MCU_OVD_VDDA_RESET	16
MCU_OVD_VCCD_RESET	17
MCU_OCD_ACTIVE_REGULATOR_RESET	18
MCU_OCD_DEEPSLEEP_REGULATOR_RESET	19
MCU_STRUCTURAL_XRES_RESET	20
MCU_CSV_HF_RESET	21
MCU_CSV_REF_RESET	22
MCU_WAKEUP_RESET	23
MCU_REGHC_OCD_RESET	24
MCU_REGHC_PMIC_RESET	25
MCU_PXRES_RESET	26

5 Functional description

5.1 Inclusion

The file *Mcu.h* includes all necessary external identifiers. Therefore, the application only needs to include *Mcu.h* to make all API functions and data types available.

The clock setting is done by the `Mcu_InitiClock` API function; the low-power mode setting is done by the `Mcu_SetMode` API function. Both CPU cores need to be initialized, so the application in each code must include *Mcu.h*.

5.2 Initialization

The MCU driver provides an initialization function for initializing the microcontroller‘s CPU core. The MCU driver must be initialized once on each core before use. Also, `Mcu_Init()` must be called on the master core before any other cores are initialized. If `Mcu_Init()` is called on the satellite core, the master core must be already initialized. The same configuration set must be specified on all cores during initialization. If no resource is assigned to the satellite core, `Mcu_Init()` is not required on that core.

```
Mcu_Init(&Mcu_Config[0]);
```

Example:

A clock setup can be accomplished by calling the following function:

```
Mcu_InitClock(McuConf_McuClockSettingConfig_MY_CLOCK);
```

Note: *See Appendix B – Access register table for the registers that will be initialized by the MCU module. If you need to initialize the registers other than those listed in Appendix B – Access register table, they should be initialized by each MCAL module or startup.*

Example:

This initializes the clock with the selected configuration. On this architecture, a switch to the PLL is already performed during the initialization of the clock when a configuration with PLL is given.

```
Mcu_DistributePllClock();
```

Note: *Clock settings that are not set by the MCU module configuration are not set by the MCU module API. If it is necessary to disable the specific clock, you must disable that clock in the configuration. If you need to set the clock trimming values, control the values in startup or user code. If you need to disable the slave of peripheral group as a system, control the slave of the peripheral group in startup.*

This function distributes the FLL, PLLs, SSCG PLLs, or all.

Note: *Only the FLL, PLLs, and/or SSCG PLLs which are set by preceding `Mcu_InitClock()` or `Mcu_SetMode()` are processed in `Mcu_GetPllStatus()` and `Mcu_DistributePllClock()`. `Mcu_InitClock()` and `Mcu_DistributePllClock()` must be called on the core that `McuClockSettingConfig` is allocated.*

5.3 MCU mode

The MCU driver provides a function that sets the microcontroller to a low-power mode:

```
Mcu_SetMode(McuConf_McuModeSettingConf_MY_MODE);
```

Example:

This function sets the microcontroller with the specified mode.

Note: *If you need to disable Hibernate mode permanently in the system, control Hibernate mode in startup. Set the `HIBERNATE_DISABLE` bit of the `PWR_HIBERNATE` register to disable Hibernate mode.*
When entering Hibernate mode, you should execute the WFI instruction on all cores except for the core that `Mcu_SetMode()` is called.
If you use the DW or the DMA for other modules, you can enable them by using the MCU driver. The MCU driver does not control each DW channel and DMA channel. They would be enabled by other modules that use them.
When FLL, PLLs, SSCG PLLs, or all are enabled by `Mcu_SetMode()`, it may be necessary to call `Mcu_GetPllStatus()` and `Mcu_DistributePllClock()` after calling `Mcu_SetMode()`. For example, FLL, PLLs, SSCG PLLs or all are not waited for stabilization.
Basically, only `Mcu_Init()` and `Mcu_SetMode()` can be called from the slave CPU core. `Mcu_SetMode()` must be called on the core that `McuModeSettingConf` is allocated.
When entering DeepSleep mode, you should not enable any FLL and PLL which uses ECO or LPECO as a reference clock.

Note: *According to Silicon Errata 218, if the LVD trip selection bits (PWR_LVD_CTL/2.HVLVD1/2_TRIPSEL_HT) are changed in a step size greater than 1, it may cause an OVD reset.*

To avoid this reset, as described in Workaround2 in the errata, when changing the McuHvLvdThreshold value with `Mcu_SetMode()`, the MCU driver implement so that it changes in steps of 1 every 10 µs.

This may result in longer execution times for `Mcu_SetMode()`.

5.4 API parameter checking

The MCU driver's services perform error checks.

When an error occurs, the error hook routine (configured via `McuErrorCalloutFunction`) is called, and the error code, service ID, module ID, and instance ID are passed as parameters.

If development error detection is enabled, all errors are reported to the DET, a central error hook function within the AUTOSAR environment. The checking itself cannot be deactivated for safety reasons.

The following development error checks are performed by the services of the MCU driver:

- The API function `Mcu_Init()` When called on the master core, it checks if the given parameter is within the valid range to select a configuration. If the parameter is invalid, the `MCU_E_INIT_FAILED` error is reported.
- The API function `Mcu_Init()` When called on the satellite core, it checks if the master core is already initialized. If the master core is not initialized yet, the `MCU_E_INIT_FAILED` error is reported.

- The API function `Mcu_Init()` When called on the satellite core, it checks if the given parameter is different from the initialized configuration of the master core. If the parameter is different, the `MCU_E_DIFFERENT_CONFIG` error is reported.
- All API functions except `Mcu_Init()` and `Mcu_GetVersionInfo()` report the error `MCU_E_UNINIT` if the MCU driver has not been initialized properly yet.
- The API function `Mcu_Init()` called on the master core checks if the any cores are already initialized. If any cores are already initialized, `MCU_E_ALREADY_INITIALIZED` error is reported.
- The API function `Mcu_Init()` called on the satellite core checks if the own core is already initialized. If the own core is already initialized, `MCU_E_ALREADY_INITIALIZED` error is reported.
- `Mcu_SetMode()` and `Mcu_CheckModeStatus()` check if the given parameter is within the valid range to select a configuration. If the parameter is invalid, the `MCU_E_PARAM_MODE` error is reported.
- `Mcu_InitClock()` and `Mcu_CheckClockStatus()` check if the given parameter is within the valid range to select a configuration. If the parameter is invalid, the `MCU_E_PARAM_CLOCK` error is reported.
- `Mcu_InitRamSection()` checks if the given parameter is within the valid range to select a configuration. If the parameter is invalid, the `MCU_E_PARAM_RAMSECTION` error is reported.
- `Mcu_GetVersionInfo()` checks if the function is called with a NULL pointer. If so, reports `MCU_E_PARAM_POINTER`.
- `Mcu_CheckModeStatus()` checks if the given parameter is a NULL pointer. If so, reports `MCU_E_PARAM_POINTER`.
- `Mcu_DistributePllClock()` reports the error `MCU_E_PLL_NOT_LOCKED` if the status of the PLL is not locked.
- `Mcu_PerformReset()` reports the error `MCU_E_RESET_NOT_PERFORMED` if `McuResetSelect` is not configured.
- `Mcu_SetMode()` reports the error `MCU_E_PARAM_MODE`, if the clock setting fails.
- `Mcu_SetMode()` reports the error `MCU_E_SYSTEM_RESOURCE_UPDATE_NOT_COMPLETED` if the update of the system common resources is not completed.
- `Mcu_Init()`, `Mcu_GetPllStatus()`, `Mcu_GetResetReason()`, `Mcu_GetResetRawValue()`, `Mcu_PerformReset()`, `Mcu_CheckClockStatus()`, and `Mcu_CheckModeStatus()` check if the functions are called from a valid core. If the core is invalid, `MCU_E_INVALID_CORE` error is reported.
- `Mcu_InitRamSection()`, `Mcu_InitClock()`, `Mcu_DistributePllClock()`, and `Mcu_SetMode()` check if the functions are called from the expected core. If the core is unexpected, `MCU_E_INVALID_CORE` error is reported.

5.5 Production error detection

If clock source failure occurs, `MCU_E_CLOCK_FAILURE` is reported to the DEM.

If reset failure occurs, `MCU_E_RESET_FAILURE` is reported to the DEM.

When an error occurs, the error hook routine (configured via `McuErrorCalloutFunction`) is also called and the error code (`MCU_E_CLOCK_FAILURE_FOR_CALLOUT` or `MCU_E_RESET_FAILURE_FOR_CALLOUT`), service ID, module ID, and instance ID are passed as parameters.

5.6 Reentrancy

The following functions are reentrant to each other and itself. All other API functions of the MCU driver are not reentrant:

- `Mcu_GetResetRawValue()`
- `Mcu_GetResetReason()`
- `Mcu_GetPllStatus()`
- `Mcu_GetVersionInfo()`
- `Mcu_CheckClockStatus()`
- `Mcu_CheckModeStatus()`
- `Mcu_GetCoreID()`

5.7 Debugging support

The MCU driver does not support debugging.

5.8 Functions available without core dependency

The following functions are available on any core without any restriction:

- `Mcu_GetResetRawValue()`
- `Mcu_GetResetReason()`
- `Mcu_GetPllStatus()`
- `Mcu_PerformReset()`
- `Mcu_GetVersionInfo()`
- `Mcu_CheckClockStatus()`
- `Mcu_CheckModeStatus()`
- `Mcu_GetCoreID()`

Note: *If `McuForcedResetEnable` is enabled and `Mcu_PerformReset()` is called on the core that the system resource is not assigned by `McuModeSettingConf` with `McuUpdateSystemResource` whose value is set to TRUE, reset is performed without retaining all RAM areas.*

5.9 APIs require privileged execution

Following APIs require privileged execution because they access the registers which requires privileged access:

- `Mcu_SetMode()`
- `Mcu_PerformReset()`

6 Hardware resources

6.1 Timer

The MCU driver does not use hardware timers.

6.2 Interrupts

The MCU driver uses the nonmaskable interrupts for low-voltage detection. The ISR should be allocated to the same core as mode configuration is allocated. The ISR must be declared in the AUTOSAR OS as Category 1 Interrupt or Category 2 Interrupt.

Note: *Vector numbers depend on the subderivative.*

To define the ISR, the IRQ name of the nonmaskable interrupt for low-voltage detection must be `Mcu_Lvd_Isr_Cat1()` for Category 1 ISR or `Mcu_Lvd_Isr_Cat2()` for Category 2 ISR.

Note: *`Mcu_SyscNmiCsv_Cat2()` and `Mcu_SyscNmiLvd_Cat2()` must be called from the (OS) interrupt service routine.*
For Category 1 usage, the address of `Mcu_SyscNmiCsv_Cat1()` and `Mcu_SyscNmiLvd_Cat1()` must be the entry in the (OS) NMI interrupt vector table.

Example: Category 1 ISR for LVD located in file *generate/src/Mcu_Irq.c*:

```
ISR_NATIVE(Mcu_Lvd_Isr_Cat1)
{
...
}
```

Example: Category 2 ISR for LVD located in file *generate/src/Mcu_Irq.c*:

```
ISR(Mcu_Lvd_Isr_Cat2)
{
...
}
```

Note: *On the Arm® Cortex®-M4 CPU, priority inversion of interrupts may occur under specific timing conditions in the integrated system with TRAVEO™ T2G MCAL. For more details, see the following errata notice.*

Arm® Cortex®-M4 Software Developers Errata Notice - 838869:
“Store immediate overlapping exception return operation might vector to incorrect interrupt”

If the user application cannot tolerate the priority inversion, a DSB instruction should be added at the end of the interrupt function to avoid the priority inversion.

TRAVEO™ T2G MCAL interrupts are handled by an ISR wrapper (handler) in the integrated system. Thus, if necessary, the DSB instruction should be added just before the end of the handler by the integrator.

6.3 Fault report structure

The MCU driver does not use fault report structure.

But, the hardware configured by the MCU driver can use the fault report structure to report errors. For example, when `McuCsvAction` is configured to `MCU_CSV_ACTION_FAULT` and the clock supervisor detects the error, the fault report structure reports the error.

To handle this, you should implement the handler for the fault report structure.

For details on the fault report structure and its assignment, see the architecture TRM and the datasheet.

7 Appendix A – API reference

7.1 Data types

7.1.1 Mcu_ConfigType

Type

`typedef struct`

Description

`Mcu_ConfigType` defines a structure which holds the MCU driver configuration set.

7.1.2 Mcu_PllStatusType

Type

```
typedef enum
{
  MCU_PLL_STATUS_UNDEFINED,
  MCU_PLL_UNLOCKED,
  MCU_PLL_LOCKED
} Mcu_PllStatusType;
```

Description

`Mcu_PllStatusType` defines the values that describe the status of the PLL.

7.1.3 Mcu_ClockType

Type

`uint8`

Description

`Mcu_ClockType` defines the range of different clock settings provided in the configuration structure. It is used as an index for selecting clock configurations for `Mcu_InitClock()`.

7.1.4 Mcu_ResetType

Type

`typedef enum` (see Table 2 for contents)

Description

`Mcu_ResetType` defines the subset of reset types.

7.1.5 Mcu_RawResetType

Type

`uint32`

Description

`Mcu_RawResetType` defines the reset reason in raw register format that is read from a reset status register. The values of `Mcu_RawResetType` depend on the hardware. For details of these values, see the information on the reset result register in the hardware manual.

7.1.6 Mcu_ModeType

Type

`uint8`

Description

`Mcu_ModeType` defines the range of different MCU modes provided in the configuration structure.

7.1.7 Mcu_RamSectionType

Type

`uint16`

Description

`Mcu_RamSectionType` defines the range of different RAM sections provided in the configuration structure.

7.1.8 Mcu_RamStateType

Type

```
typedef enum
{
  MCU_RAMSTATE_INVALID,
  MCU_RAMSTATE_VALID
} Mcu_RamStateType;
```

Description

`Mcu_RamStateType` defines the values that describe the status of the RAM.

7.1.9 Mcu_StatusType

Type

```
typedef struct
{
  Mcu_CpuStatusType  Cm0Status;
  Mcu_CpuStatusType  MainCoreStatus[4];
  Mcu_SysStatusType  SysStatus;
}Mcu_StatusType;
```

Description

`Mcu_StatusType` defines the result of status check.

`Cm0Status`: CM0P CPU status.

`MainCoreStatus[4]`: Main core CPU status. If there is only one core in the system, only the first of the array is used.

`SysStatus`: System status.

7.1.10 Mcu_CpuStatusType

Type

`uint8`

Description

`Mcu_CpuStatusType` defines the CPU status.

7.1.11 Mcu_SysStatusType

Type

`uint8`

Description

`Mcu_SysStatusType` defines the system status.

7.2 Constants

7.2.1 Error codes

The service might return the error codes, shown in Table 3, if development error detection is enabled:

Table 3 Error codes

Name	Value	Description
`MCU_E_PARAM_CLOCK`	0x0B	`ClockSetting` is not a valid parameter.
`MCU_E_PARAM_MODE`	0x0C	`McuMode` is not a valid parameter.
`MCU_E_PARAM_RAMSECTION`	0x0D	`RamSection` is not a valid parameter.
`MCU_E_PLL_NOT_LOCKED`	0x0E	PLL not locked yet.
`MCU_E_UNINIT`	0x0F	MCU has not been initialized yet.
`MCU_E_PARAM_POINTER`	0x10	`versioninfo` is a NULL pointer.
`MCU_E_INIT_FAILED`	0x11	The `Mcu_Init()` is called with a wrong parameter on the master core or the master core is not initialized yet.
`MCU_E_CLOCK_FAILURE_FOR_CALLOUT`	0x40	Clock source failure occurred. This error ID is used to call the error callout handler.
`MCU_E_RESET_FAILURE_FOR_CALLOUT`	0x41	Reset failure occurred. This error ID is used to call the error callout handler

Name	Value	Description
MCU_E_RESET_NOT_PERFORMED	0x60	Mcu_PerformReset did not perform reset.
MCU_E_SYSTEM_RESOURCE_UPDATE_NOT_COMPLETED	0x80	System resource update did not complete.
MCU_E_INVALID_CORE	0xA0	API is called on the invalid or unexpected core.
MCU_E_DIFFERENT_CONFIG	0xA1	Mcu_Init() is called with a wrong parameter on the satellite core.
MCU_E_ALREADY_INITIALIZED	0xA2	The MCU driver is already initialized on the executing core.

7.2.2 Version information

Table 4 lists the version information published in the driver's header file.

Table 4 Version information

Name	Value	Description
MCU_SW_MAJOR_VERSION	Refer to release notes	Major version number
MCU_SW_MINOR_VERSION	Refer to release notes	Minor version number
MCU_SW_PATCH_VERSION	Refer to release notes	Patch version number

7.2.3 Module information

Table 5 Module information

Name	Value	Description
MCU_MODULE_ID	101	Module ID
MCU_VENDOR_ID	66	Vendor ID

7.2.4 API service IDs

The API service IDs, listed in Table 6, are published in the driver's header file.

Table 6 API service IDs

Name	Value	Description
MCU_API_SERVICE_INIT	0x0	Service ID of Mcu_Init
MCU_API_SERVICE_INIT_RAM_SECTION	0x1	Service ID of Mcu_InitRamSection
MCU_API_SERVICE_INIT_CLOCK	0x2	Service ID of Mcu_InitClock
MCU_API_SERVICE_DISTRIBUTE_PLL_CLOCK	0x3	Service ID of Mcu_DistributePllClock
MCU_API_SERVICE_GET_PLL_STATUS	0x4	Service ID of Mcu_GetPllStatus
MCU_API_SERVICE_GET_RESET_REASON	0x5	Service ID of Mcu_GetResetReason
MCU_API_SERVICE_GET_RESET_RAW_VALUE	0x6	Service ID of Mcu_GetResetRawValue
MCU_API_SERVICE_PERFORM_RESET	0x7	Service ID of Mcu_PerformReset
MCU_API_SERVICE_SET_MODE	0x8	Service ID of Mcu_SetMode
MCU_API_SERVICE_GET_VERSION_INFO	0x9	Service ID of Mcu_GetVersionInfo

Name	Value	Description
MCU_API_SERVICE_CHECK_CLOCK_STATUS	0x20	Service ID of Mcu_CheckClockStatus
MCU_API_SERVICE_CHECK_MODE_STATUS	0x21	Service ID of Mcu_CheckModeStatus
MCU_API_SERVICE_LVD_ISR	0x30	Service ID of ISR for low-voltage detection interrupt
MCU_API_SERVICE_GET_CORE_ID	0x40	Service ID of Mcu_GetCoreID

7.2.5 Core ID value

Name	Value	Description
MCU_CM0_CORE	0	ID of Arm® Cortex®-M0+. This value is returned from Mcu_GetCoreID().
MCU_CM4_OR_CM7_0_CORE	1	ID of Arm® Cortex®-M4 or Arm® Cortex®-M7 CPU0. This value is returned from Mcu_GetCoreID().
MCU_CM7_1_CORE	2	ID of Arm® Cortex®-M7 CPU1. This value is returned from Mcu_GetCoreID().
MCU_CM7_2_CORE	3	ID of Arm® Cortex®-M7 CPU2. This value is returned from Mcu_GetCoreID().
MCU_CM7_3_CORE	4	ID of Arm® Cortex®-M7 CPU3. This value is returned from Mcu_GetCoreID().
MCU_INVALID_CORE	255	Invalid core ID.

7.3 Functions

7.3.1 Mcu_Init

Syntax

```
void Mcu_Init(
    const Mcu_ConfigType* ConfigPtr
)
```

Service ID

0x0

Parameters (in)

ConfigPtr

Parameters (out)

None

Return value

None

DET errors

`MCU_E_INIT_FAILED` – Invalid parameter is passed to the master core, or the master core is not initialized yet when `Mcu_Init()` is called on the satellite core.

`MCU_E_ALREADY_INITIALIZED` – Cores are already initialized when `Mcu_Init()` is called on the master core, or the MCU driver on the satellite core is already initialized.

`MCU_E_DIFFERENT_CONFIG` – Invalid parameter is passed to the `Mcu_Init()` called on the satellite core.

`MCU_E_INVALID_CORE` – Called on the invalid core.

DEM errors

`MCU_E_CLOCK_FAILURE` – Clock source failure occurred.

Description

This function initializes the MCU driver and shows the configuration settings for power down, clock, and RAM sections within the MCU driver.

7.3.2 Mcu_InitRamSection

Syntax

```
Std_ReturnType Mcu_InitRamSection(
    Mcu_RamSectionType RamSection
)
```

Service ID

0x1

Parameters (in)

`RamSection` – Selects the RAM memory section provided in the configuration set.

Parameters (out)

None

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `MCU_E_PARAM_RAMSECTION` – Invalid parameter.
- `MCU_E_UNINIT` – The module is uninitialized.
- `MCU_E_INVALID_CORE` – Called on the unexpected core.

DEM errors

None

Description

This function initializes the RAM section wise.

7.3.3 Mcu_InitClock

Syntax

```
Std_ReturnType Mcu_InitClock(
    Mcu_ClockType ClockSetting
)
```

Service ID

0x2

Parameters (in)

`ClockSetting` – Clock setting.

Parameters (out)

None

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `MCU_E_PARAM_CLOCK` – Invalid parameter.
- `MCU_E_UNINIT` – The module is uninitialized.
- `MCU_E_INVALID_CORE` – Called on the unexpected core.

DEM errors

`MCU_E_CLOCK_FAILURE` – Clock source failure.

Description

This function initializes the PLL and other MCU-specific clock options.

7.3.4 Mcu_DistributePllClock

Syntax

```
Std_ReturnType Mcu_DistributePllClock(
    void
)
```

Service ID

0x3

Parameters (in)

None

Parameters (out)

None

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `MCU_E_PLL_NOT_LOCKED` - The status of the PLL is not locked.
- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_INVALID_CORE` - Called on the unexpected core.

DEM errors

None

Description

This function activates the FLL, PLLs, SSCG PLLs or all to MCU clock distribution. This function is executed if the MCU needs a separate request to activate the FLL, PLLs, or both after the FLL, PLLs, SSCG PLLs or all are locked.

7.3.5 Mcu_GetPllStatus

Syntax

```
Mcu_PllStatusType Mcu_GetPllStatus(
    void
)
```

Service ID

0x4

Parameters (in)

None

Parameters (out)

None

Return value

The lock status of the PLL clock.

DET errors

- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_INVALID_CORE` - Called on the invalid core.

DEM errors

None

Description

This function provides the lock status of the PLLs, SSCG PLLs, or FLL.

7.3.6 Mcu_GetResetReason

Syntax

```
Mcu_ResetType Mcu_GetResetReason(
    void
)
```

Service ID

0x5

Parameters (in)

None

Parameters (out)

None

Return value

Reset reason

DET errors

- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_INVALID_CORE` - Called on the invalid core.

DEM errors

None

Description

This function returns the reset reason, if supported by hardware. A call to the API service returns exactly one reset reason. If no more reset reasons are available, the reset cause `MCU_RESET_UNDEFINED` is returned.

7.3.7 Mcu_GetResetRawValue

Syntax

```
Mcu_RawResetType Mcu_GetResetRawValue(
    void
)
```

Service ID

0x6

Parameters (in)

None

Parameters (out)

None

Return value

Raw reset type

DET errors

- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_INVALID_CORE` - Called on the invalid core.

DEM errors

None

Description

This function reads the reset type from the hardware register, if supported.

7.3.8 Mcu_PerformReset

Syntax

```
void Mcu_PerformReset(
    void
)
```

Service ID

0x7

Parameters (in)

None

Parameters (out)

None

Return value

None

DET errors

- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_RESET_NOT_PERFORMED` - `McuResetSelect` is not configured.
- `MCU_E_INVALID_CORE` - Called on the invalid core.

DEM errors

`MCU_E_RESET_FAILURE` - Reset failure occurred.

Description

This function performs a microcontroller reset, whereby the hardware feature of the microcontroller is used.

7.3.9 Mcu_SetMode

Syntax

```
void Mcu_SetMode(
    Mcu_ModeType McuMode
)
```

Service ID

0x8

Parameters (in)

`McuMode` - Selects the mode configured in the configuration set.

Parameters (out)

None

Return value

None

DET errors

- `MCU_E_PARAM_MODE` - Invalid parameter.
- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_SYSTEM_RESOURCE_UPDATE_NOT_COMPLETED` - System resource update error.
- `MCU_E_INVALID_CORE` - Called on the unexpected core.

DEM errors

`MCU_E_CLOCK_FAILURE` - Clock source failure has occurred.

Description

This function sets the microcontroller into a low-power mode.

7.3.10 Mcu_GetVersionInfo

Syntax

```
void Mcu_GetVersionInfo(
    Std_VersionInfoType* versioninfo
)
```

Service ID

0x9

Parameters (in)

None

Parameters (out)

`versioninfo` - Version information of the MCU driver.

Return value

None

DET errors

`MCU_E_PARAM_POINTER` - Parameter `versioninfo` is a NULL pointer.

DEM errors

None

Description

This function returns the version of this module.

7.3.11 Mcu_CheckClockStatus

Syntax

```
Std_ReturnType Mcu_CheckClockStatus(
    Mcu_ClockType ClockSettingId,
)
```

Service ID

0x20

Parameters (in)

`ClockSettingId` - Clock setting ID for checking.

Parameters (out)

None

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_PARAM_CLOCK` - Invalid parameter.
- `MCU_E_INVALID_CORE` - Called on the invalid core.

DEM errors

None

Description

This service checks whether the register has a value corresponding to the clock configuration.

7.3.12 Mcu_CheckModeStatus

Syntax

```
Std_ReturnType Mcu_CheckModeStatus(
    Mcu_ModeType ModeSettingId,
    Mcu_StatusType* StatusPtr
)
```

Service ID

0x21

Parameters (in)

`ModeSettingId` - Mode setting ID for checking.

Parameters (out)

`StatusPtr` - Result of status check.

Return value

`E_OK` or `E_NOT_OK`

DET errors

- `MCU_E_UNINIT` - The module is uninitialized.
- `MCU_E_PARAM_MODE` - Invalid parameter.
- `MCU_E_PARAM_POINTER` - Parameter `StatusPtr` is a NULL pointer.
- `MCU_E_INVALID_CORE` - Called on the invalid core.

DEM errors

None

Description

This service checks whether the register has a value corresponding to the mode configuration.

7.3.13 Mcu_GetCoreID

Syntax

```
Uint8 Mcu_GetCoreID(
    void
)
```

Service ID

0x40

Parameters (in)

None

Parameters (out)

None

Return value

ID of the executing core.

0: Arm® Cortex®-M0+

1: Arm® Cortex®-M4 or Arm® Cortex®-M7 CPU 0

2: Arm® Cortex®-M7 CPU 1

3: Arm® Cortex®-M7 CPU 2

4: Arm® Cortex®-M7 CPU 3

255: Invalid

DET errors

`None`

DEM errors

None

Description

This function returns the ID of the executing core.

7.4 Required callback functions

7.4.1 DET

If development error detection is enabled, the MCU driver uses the following callback function provided by DET. If you do not use DET, you must implement this function within your application.

Det_ReportError

Syntax

```
Std_ReturnType Det_ReportError
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` - Module ID of calling module.
- `InstanceId` - `McuCoreConfigurationId` of the core that calls this function or `MCU_INVALID_CORE`.
- `ApiId` - ID of the API service that calls this function.
- `ErrorId` - ID of the detected development error.

Return value

Returns always `E_OK` (is required for services).

Description

Service for reporting development errors.

7.4.2 DEM

If DEM notifications are enabled, the MCU driver uses the following callback function provided by DEM. If you do not use DEM, you must implement this function within your application.

Dem_ReportErrorStatus

Syntax

```
void Dem_ReportErrorStatus
(
    Dem_EventIdType EventId,
    Dem_EventStatusType EventStatus
)
```

Reentrancy

Reentrant

Parameters (in)

- `EventId` - Identification of an event by the assigned event ID.
- `EventStatus` - Monitor test result of the given event.

Return value

None

Description

Service for reporting diagnostic events.

7.4.3 Callout functions

7.4.3.1 Error callout API

The AUTOSAR MCU module requires an error callout handler. Each error is reported to this handler; error checking cannot be switched off. The name of the function to be called can be configured with the `McuErrorCalloutFunction` parameter.

Syntax

```
void Error_Handler_Name
(
    uint16 ModuleId,
    uint8 InstanceId,
    uint8 ApiId,
    uint8 ErrorId
)
```

Reentrancy

Reentrant

Parameters (in)

- `ModuleId` - Module ID of the calling module.
- `InstanceId` - `McuCoreConfigurationId` of the core that calls this function or `MCU_INVALID_CORE`.
- `ApiId` - ID of the API service that calls this function.
- `ErrorId` - ID of the detected error.

Return value

None

Description

Service for reporting errors.

Get core ID API

The AUTOSAR MCU module requires a function to get valid core ID. This function is being used to determine from which core the code is getting executed. The name of the function to be called can be configured by the `McuGetCoreIdFunction` parameter.

Syntax

`uint8` *`GetCoreID_Function_Name`* `(void)`

Reentrancy

Reentrant

Parameters (in)

None

Return value

- `CoreId` - ID of the current core.

Description

Service for getting a valid core ID.

Note: *This function shall return the core ID configured in the `McuMulticore/McuCoreConfiguration/McuCoreId`.*
For example: Two cores are configured in the `McuCoreConfiguration`.

Executing core	McuCoreConfigurationId	McuCoreId
CM7_0	0	15
CM7_1	1	16

- Upon calling this function from core CM7_0, it shall return 15.
- Upon calling this function from core CM7_1, it shall return 16.

8 Appendix B – Access register table

8.1 PERI

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
TIMEOUT_CTL	31:0	Word (32 bits)	0x00000000 | timeout value	Timeout control register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFFF	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
DIV_CMD	31:0	Word (32 bits)	Depends on configuration value.	Divider command register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (PCLK divider type << 8) | (PCLK divider index)	Clock control register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
DIV_8_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8)	Divider control (for 8.0 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF01	0x0000**0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
DIV_16_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8)	Divider control (for 16.0 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00FFFF01	0x00****0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
DIV_16_5_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8) | (fractional divider value << 3)	Divider control (for 16.5 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00FFFFF9	0x00****** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
DIV_24_5_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8) | (fractional divider value << 3)	Divider control (for 24.5 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0xFFFFFFF9	0x******** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PERI_GROUP_STRUCT.CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (divider value << 8)	Clock control of peripheral group register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF00	0x0000**00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PERI_GROUP_STRUCT.SL_CTL	31:0	Word (32 bits)	0x00000000 | (slave enable << slave n) (n = 0 - 15)	Peripheral group, slave n disable	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFFF	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PCLK_GROUP.DIV_CMD	31:0	Word (32 bits)	Depends on configuration value.	Divider command register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PCLK_GROUP.CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (PCLK divider type << 8) | (PCLK divider index)	Clock control register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PCLK_GROUP.DIV_8_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8)	Divider control (for 8.0 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF01	0x0000**0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PCLK_GROUP.DIV_16_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8)	Divider control (for 16.0 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00FFFF01	0x00****0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
PCLK_GROUP.DIV_16_5_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8) | (fractional divider value << 3)	Divider control (for 16.5 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00FFFFF9	0x00****** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PCLK_GROUP.DIV_24_5_CTL	31:0	Word (32 bits)	0x00000000 | (integer divider value << 8) | (fractional divider value << 3)	Divider control (for 24.5 divider) register	Mcu_Init Mcu_InitClock Mcu_SetMode	0xFFFFFFF9	0x******** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

8.2 CPUSS

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
IDENTITY	31:0	Word (32 bits)	-	Identity	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CM4_STATUS	31:0	Word (32 bits)	-	CM4 status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CM4_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (fast clock divider value << 8)	CM4 clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF00	0x0000**00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CM0_CTL	31:0	Word (32 bits)	-	CM0+ control	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CM0_STATUS	31:0	Word (32 bits)	-	CM0+ status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CM0_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (peri clock divider value << 24) | (slow clock divider value << 8)	CM0+ clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0xFF00FF00	0x**00**00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CM4_PWR_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	CM4 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
CM4_PWR_DELAY_CTL	31:0	Word (32 bits)	0x00000000 | power up delay	CM4 power control	Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_SetMode. Digit * depends on configuration value.)
RAM0_CTL	31:0	Word (32 bits)	0x00000000 | (wait cycle for fast domain << 8) | (wait cycle for slow domain)	RAM 0 control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000303	0x00000*0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
RAM0_STATUS	31:0	Word (32 bits)	-	RAM 0 status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
RAM0_PWR_MACRO_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	RAM 0 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
RAM1_CTL	31:0	Word (32 bits)	0x00000000 | (wait cycle for fast domain << 8) | (wait cycle for slow domain)	RAM 1 control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000303	0x00000*0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
RAM1_STATUS	31:0	Word (32 bits)	-	RAM 1 status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
RAM1_PWR_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	RAM 1 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
RAM2_CTL	31:0	Word (32 bits)	0x00000000 | (wait cycle for fast domain << 8) | (wait cycle for slow domain)	RAM 2 control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000303	0x00000*0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
RAM2_STATUS	31:0	Word (32 bits)	-	RAM 2 status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
RAM2_PWR_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	RAM 2 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
RAM_PWR_DELAY_CTL	31:0	Word (32 bits)	0x00000000 | power up delay	Power up delay used for all SRAM power domains	Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_SetMode. Digit * depends on configuration value.)
ROM_CTL	31:0	Word (32 bits)	0x00000000 | (wait cycle for fast domain << 8) | (wait cycle for slow domain)	ROM control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000303	0x00000*0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
SYSTICK_CTL	31:0	Word (32 bits)	-	SysTick timer control	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CM0_SYSTEM_INT_CTL	31:0	Word (32 bits)	-	CM0+ system interrupt control	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CM4_SYSTEM_INT_CTL	31:0	Word (32 bits)	-	CM4 system interrupt control	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CM7_0_STATUS	31:0	Word (32 bits)	-	CM7_0 status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
FAST_0_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (fast 0 clock integer divider value << 8) | (fast 0 clock fractional divider value << 3)	Fast 0 clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFF8	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
TRC_DBG_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (trace debug clock divider value << 8)	Trace debug clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF00	0x0000**00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CM7_1_STATUS	31:0	Word (32 bits)	-	CM7_1 status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
FAST_1_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (fast 1 clock integer divider value << 8) | (fast 1 clock fractional divider value << 3)	Fast 1 clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFF8	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
FAST_2_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (fast 2 clock integer divider value << 8) | (fast 2 clock fractional divider value << 3)	Fast 2 clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFF8	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on the configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
FAST_3_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (fast 3 clock integer divider value << 8) | (fast 3 clock fractional divider value << 3)	Fast 3 clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFF8	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on the configuration value.)
SLOW_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (slow clock divider value << 8)	Slow clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF00	0x0000**00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PERI_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (peri clock divider value << 8)	Peripheral clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF00	0x0000**00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
MEM_CLOCK_CTL	31:0	Word (32 bits)	0x00000000 | (mem clock divider value << 8)	Memory clock control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF00	0x0000**00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CM7_0_PWR_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	CM7_0 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
CM7_0_PWR_DELAY_CTL	31:0	Word (32 bits)	0x00000000 | power up delay	CM7_0 power control	Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_SetMode. Digit * depends on configuration value.)
CM7_1_PWR_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	CM7_1 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
CM7_1_PWR_DELAY_CTL	31:0	Word (32 bits)	0x00000000 | power up delay	CM7_1 power control	Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CM7_2_PWR_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	CM7_2 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
CM7_2_PWR_DELAY_CTL	31:0	Word (32 bits)	0x00000000 | power up delay	CM7_2 power control	Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_SetMode. Digit * depends on configuration value.)
CM7_3_PWR_CTL	31:0	Word (32 bits)	0x00000000 | (register key << 16) | power mode	CM7_3 power control	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
CM7_3_PWR_DELAY_CTL	31:0	Word (32 bits)	0x00000000 | power up delay	CM7_3 power control	Mcu_SetMode	0x000003FF	0x00000*** (After Mcu_SetMode. Digit * depends on configuration value.)
CM7_0_SYSTEM_INT_CTL	31:0	Word (32 bits)	-	CM7_0 system interrupt control	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CM7_1_SYSTEM_INT_CTL	31:0	Word (32 bits)	-	CM7_1 system interrupt control	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

8.3 DW

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CTL	31:0	Word (32 bits)	0x00000000 | (DW enable << 31)	Control	Mcu_SetMode	0x80000000	0x*0000000 (After Mcu_SetMode. Digit * depends on configuration value.)

8.4 DMAC

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CTL	31:0	Word (32 bits)	0x00000000 | (DMAC enable << 31)	Control	Mcu_SetMode	0x80000000	0x*0000000 (After Mcu_SetMode. Digit * depends on configuration value.)

8.5 FLASHC

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
FLASH_CTL	31:0	Word (32 bits)	0x00000000 | wait cycle	Flash control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000000F	0x0000000* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

8.6 FLASHC1

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
FLASH_CTL	31:0	Word (32 bits)	0x00000000 | wait cycle	Flash control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000000F	0x0000000* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on the configuration value.)

8.7 SRSS

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
PWR_LVD_CTL	31:0	Word (32 bits)	Depends on configuration value.	High voltage / low voltage detector (HVLVD) configuration register	Mcu_SetMode	0x0007DF00	0x000***00 (After Mcu_SetMode. Digit * depends on configuration value.)
PWR_LVD_CTL2	31:0	Word (32 bits)	Depends on configuration value.	High voltage / low voltage detector (HVLVD) configuration register #2	Mcu_SetMode	0x0007DF00	0x000***00 (After Mcu_SetMode. Digit * depends on configuration value.)
CLK_DSI_SELECT	31:0	Word (32 bits)	0x00000000 | DSI source	Clock DSI select register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000001F	0x000000** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_OUTPUT_FAST	31:0	Word (32 bits)	Depends on configuration value.	Fast clock output select register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0FFF0FFF	0x0***0*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_OUTPUT_SLOW	31:0	Word (32 bits)	Depends on configuration value.	Slow clock output select register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x000000FF	0x000000** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_CAL_CNT1	31:0	Word (32 bits)	-	Clock calibration counter 1	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CLK_CAL_CNT2	31:0	Word (32 bits)	-	Clock calibration counter 2	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SRSS_INTR	31:0	Word (32 bits)	0x00000000 | (HVLVD2 interrupt << 2) | (HVLVD1 interrupt << 1)	SRSS interrupt register	Mcu_SetMode Mcu_Lvd_Isr_Cat1 Mcu_Lvd_Isr_Cat2	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SRSS_INTR_SET	31:0	Word (32 bits)	-	SRSS interrupt set register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SRSS_INTR_MASK	31:0	Word (32 bits)	0x00000000 | (HVLVD2 interrupt << 2) | (HVLVD1 interrupt << 1)	SRSS interrupt mask register	Mcu_SetMode	0x00000003	0x0000000* (After Mcu_SetMode. Digit * depends on configuration value.)
SRSS_INTR_MASKED	31:0	Word (32 bits)	-	SRSS interrupt masked register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PWR_CTL	31:0	Word (32 bits)	-	Power mode control	Read only.	0x00000000 (monitoring is not needed.)	The value does not care due to it changes dynamically.
PWR_CTL2	31:0	Word (32 bits)	Depends on configuration value.	Power mode control 2	Mcu_SetMode	0x81100011	0x***000** (After Mcu_SetMode. Digit * depends on configuration value.)
PWR_HIBERNATE	31:0	Word (32 bits)	Depends on configuration value.	HIBERNATE mode register	Mcu_SetMode	0xBFFEFFFF	0x****3AFF (After Mcu_SetMode to Hibernate mode. Digit * depends on configuration value.) 0x00000000 (After Mcu_SetMode to Sleep or DeepSleep mode.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
PWR_HIB_WAKE_CTL	31:0	Word (32 bits)	Depends on the configuration value.	HIBERNATE Wakeup mask register	Mcu_SetMode	0xE0FFFFFF	0x******** (After Mcu_SetMode. Digit * depends on the configuration value.)
PWR_HIB_WAKE_CTL2	31:0	Word (32 bits)	Depends on the configuration value.	HIBERNATE Wakeup polarity register	Mcu_SetMode	0x00FFFFFF	0x******** (After Mcu_SetMode. Digit * depends on the configuration value.)
PWR_HIB_WAKE_CAUSE	31:0	Word (32 bits)	0xE0FFFFFF	HIBERNATE Wakeup cause register	Mcu_SetMode	0x00000000 (Monitoring is not required.)	0x00000000 (Monitoring is not needed.)
PWR_SSV_CTL	31:0	Word (32 bits)	Depends on configuration value.	Supply supervision control register	Mcu_SetMode	0x09D909D9	0x******** (After Mcu_SetMode. Digit * depends on configuration value.)
TST_DDFT_FAST_CTL	31:0	Word (32 bits)	Depends on configuration value.	Fast digital DFT control register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x66003F3F	0x**00**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
TST_DDFT_SLOW_CTL	31:0	Word (32 bits)	Depends on configuration value.	Slow digital DFT control register	Mcu_Init Mcu_InitClock Mcu_SetMode	0xC0009F9F	0x*000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_PATH_SELECT	31:0	Word (32 bits)	0x00000000 | clock path source	Clock path select register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000007	0x0000000* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CLK_ROOT_SELECT	31:0	Word (32 bits)	0x00000000 | (root clock enable << 31) | (root clock direct mux << 8) | (root clock divider value << 4) | root clock source	Clock root select register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x8000013F	0x*0000*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_HF_STRUCTS.CSV_ACT_STRUCT.REF_CTL	31:0	Word (32 bits)	0x00000000 | (CSV enable << 31) | (CSV action << 30) | CSV startup delay	Clock supervision reference control for root clocks	Mcu_Init Mcu_InitClock Mcu_SetMode	0xC000FFFF	0x*000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_HF_STRUCTS.CSV_ACT_STRUCT.REF_LIMIT	31:0	Word (32 bits)	0x00000000 | (CSV upper threshold << 16) | CSV lower threshold	Clock supervision reference limits for root clocks	Mcu_Init Mcu_InitClock Mcu_SetMode	0xFFFFFFFF	0x******** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_HF_STRUCTS.CSV_ACT_STRUCT.MON_CTL	31:0	Word (32 bits)	0x00000000 | CSV period	Clock supervision monitor control for root clocks	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFFF	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_HF_STRUCTS.CSV_ACT_STRUCT.CNT_STAT	31:0	Word (32 bits)	-	Clock supervision counters for root clocks	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CLK_SELECT	31:0	Word (32 bits)	Depends on configuration value.	Clock selection register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FF03	0x0000**0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CLK_TIMER_CTL	31:0	Word (32 bits)	Depends on configuration value.	Timer clock control register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x80FF0301	0x*0**0*0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_ILO0_CONFIG	31:0	Word (32 bits)	0x00000000 | (ILO0 enable << 31) | (ILO0 monitor enable << 30) | ILO0 backup enable	ILO0 configuration	Mcu_Init Mcu_InitClock Mcu_SetMode	0xC0000001	0x*000000* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_ILO1_CONFIG	31:0	Word (32 bits)	0x00000000 | (ILO1 enable << 31) | (ILO1 monitor enable << 30)	ILO1 configuration	Mcu_Init Mcu_InitClock Mcu_SetMode	0xC0000000	0x*0000000 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_IMO_CONFIG	31:0	Word (32 bits)	0x00000000 | (IMO enable << 31)	IMO configuration	Mcu_Init Mcu_InitClock Mcu_SetMode	0x80000000	0x*0000000 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_ECO_CONFIG	31:0	Word (32 bits)	0x00000000 | (ECO enable << 31) | (ECO divider enable << 28) | (ECO divider disable << 27) | (AGC enable << 1)	ECO configuration register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x98000002	0x**00000* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CLK_ECO_PRESCALE	31:0	Word (32 bits)	0x00000000 | (ECO integer divider value << 16) | (ECO fractional divider value << 8)	ECO prescaler configuration register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x03FFFF00	0x0*****00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_ECO_STATUS	31:0	Word (32 bits)	-	ECO status register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CLK_FLL_CONFIG	31:0	Word (32 bits)	Depends on configuration value.	FLL configuration register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x8103FFFF	0x**0***** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_FLL_CONFIG2	31:0	Word (32 bits)	Depends on configuration value.	FLL configuration register 2	Mcu_Init Mcu_InitClock Mcu_SetMode	0xFFFF1FFF	0x******** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_FLL_CONFIG3	31:0	Word (32 bits)	Depends on configuration value.	FLL configuration register 3	Mcu_Init Mcu_InitClock Mcu_SetMode Mcu_DistributePllClock	0x301FFFFF	0x*0****** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.) 0x*0000000 (After Mcu_DistributePllClock. Digit * depends on configuration value.)
CLK_FLL_CONFIG4	31:0	Word (32 bits)	Depends on configuration value.	FLL configuration register 4	Mcu_Init Mcu_InitClock Mcu_SetMode	0xC1FF07FF	0x****0*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CLK_FLL_STATUS	31:0	Word (32 bits)	-	FLL status register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CLK_ECO_CONFIG2	31:0	Word (32 bits)	Depends on configuration value.	ECO configuration register 2	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00007FF7	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CLK_PLL_CONFIG	31:0	Word (32 bits)	Depends on configuration value.	PLL configuration register	Mcu_Init Mcu_InitClock Mcu_SetMode Mcu_DistributePllClock	0xB81F1F7F	0x******** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.) 0x*0000000 (After Mcu_DistributePllClock. Digit * depends on configuration value.)
CLK_PLL_STATUS	31:0	Word (32 bits)	-	PLL status register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CSV_REF_SEL	31:0	Word (32 bits)	0x00000000 | CSV reference clock	Select CSV reference clock for Active domain	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000007	0x0000000* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_REF_STRUCT.CSV_ACT_STRUCT.REF_CTL	31:0	Word (32 bits)	0x00000000 | (CSV enable << 31) | (CSV action << 30) | CSV startup delay	Clock supervision reference control for reference clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0xC000FFFF	0x*000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CSV_REF_STRUCT.CSV_ACT_STRUCT.REF_LIMIT	31:0	Word (32 bits)	0x00000000 | (CSV upper threshold << 16) | CSV lower threshold	Clock supervision reference limits for reference clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0xFFFFFFFF	0x******** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_REF_STRUCT.CSV_ACT_STRUCT.MON_CTL	31:0	Word (32 bits)	0x00000000 | CSV period	Clock supervision monitor control for reference clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x0000FFFF	0x0000**** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_REF_STRUCT.CSV_ACT_STRUCT.CNT_STAT	31:0	Word (32 bits)	-	Clock supervision counters for reference clock	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CSV_LF_STRUCT.CSV_DPSLP_STRUCT.REF_CTL	31:0	Word (32 bits)	0x00000000 | (CSV enable << 31) | CSV startup delay	Clock supervision reference control for LF clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x800001FF	0x*0000*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_LF_STRUCT.CSV_DPSLP_STRUCT.REF_LIMIT	31:0	Word (32 bits)	0x00000000 | (CSV upper threshold << 16) | CSV lower threshold	Clock supervision reference limits for LF clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00FF00FF	0x00**00** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_LF_STRUCT.CSV_DPSLP_STRUCT.MON_CTL	31:0	Word (32 bits)	0x00000000 | CSV period	Clock supervision monitor control for LF clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x000000FF	0x000000** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_LF_STRUCT.CSV_DPSLP_STRUCT.CNT_STAT	31:0	Word (32 bits)	-	Clock supervision counters for LF clock	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CSV_ILO_STRUCT.CSV_DPSLP_STRUCT.REF_CTL	31:0	Word (32 bits)	0x00000000 | (CSV enable << 31) | CSV startup delay	Clock supervision reference control for HVILO clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x800001FF	0x*0000*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_ILO_STRUCT.CSV_DPSLP_STRUCT.REF_LIMIT	31:0	Word (32 bits)	0x00000000 | (CSV upper threshold << 16) | CSV lower threshold	Clock supervision reference limits for HVILO clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00FF00FF	0x00**00** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_ILO_STRUCT.CSV_DPSLP_STRUCT.MON_CTL	31:0	Word (32 bits)	0x00000000 | CSV period	Clock supervision monitor control for HVILO clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x000000FF	0x000000** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
CSV_ILO_STRUCT.CSV_DPSLP_STRUCT.CNT_STAT	31:0	Word (32 bits)	-	Clock supervision counters for HVILO clock	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
RES_CAUSE	31:0	Word (32 bits)	0x61FF01FF	Reset cause observation register	Mcu_Init	0x61FF01FF	0x****0*** (After Mcu_Init. Digit * depends on configuration value.)
RES_CAUSE2	31:0	Word (32 bits)	0x0001FFFF	Reset cause observation register 2	Mcu_Init	0x0001FFFF	0x000***** (After Mcu_Init. Digit * depends on configuration value.)
WDT_B_STRUCT.LOCK	31:0	Word (32 bits)	0x00000000 | lock value	WDT lock register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
PLL400M_STRUCT.CONFIG	31:0	Word (32 bits)	Depends on configuration value.	400MHz PLL configuration register	Mcu_Init Mcu_InitClock Mcu_SetMode	0xB61F1FFF	0x******** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
PLL400M_STRUCT .CONFIG2	31:0	Word (32 bits)	Depends on configuration value.	400MHz PLL configuration register 2	Mcu_Init Mcu_InitClock Mcu_SetMode	0xF0FFFFFF	0x*0******
PLL400M_STRUCT .CONFIG3	31:0	Word (32 bits)	Depends on configuration value.	400MHz PLL configuration register 3	Mcu_Init Mcu_InitClock Mcu_SetMode	0x910703FF	0x**0*0*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
PLL400M_STRUCT .STATUS	31:0	Word (32 bits)	-	400MHz PLL status register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

8.8 BACKUP

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CTL	31:0	Word (32 bits)	Depends on configuration value.	Control	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00013308	0x000***0* (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
STATUS	31:0	Word (32 bits)	-	Status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
RTC_RW	31:0	Word (32 bits)	0x00000000 0x00000001 0x00000002	RTC read write register	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
LPECO_CTL	31:0	Word (32 bits)	0x00000000 | (LPECO enable << 31) | (LPECO amplitude detector enable << 30) | (LPECO divider enable << 28) | (LPECO maximum amplitude << 12) | (LPECO frequency range << 8) | (LPECO capacitance range << 4)	Low-power external crystal oscillator control	Mcu_Init Mcu_InitClock Mcu_SetMode	0xD0001130	0x*000***0 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
LPECO_PRESCALE	31:0	Word (32 bits)	0x00000000 | (LPECO integer divider value << 16) | (LPECO fractional divider value << 8)	Low-power external crystal oscillator prescaler	Mcu_Init Mcu_InitClock Mcu_SetMode	0x03FFFF00	0x0*****00 (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on configuration value.)
LPECO_STATUS	31:0	Word (32 bits)	-	Low-power external crystal oscillator status	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.REF_CTL	31:0	Word (32 bits)	0x00000000 | (CSV enable << 31) | CSV startup delay	Clock supervision reference control for backup clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x800001FF	0x*0000*** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on the configuration value.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.REF_LIMIT	31:0	Word (32 bits)	0x00000000 | (CSV upper threshold << 16) | CSV lower threshold	Clock supervision reference limits for backup clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x00FF00FF	0x00**00** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on the configuration value.)
CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.MON_CTL	31:0	Word (32 bits)	0x00000000 | CSV period	Clock supervision monitor control for backup clock	Mcu_Init Mcu_InitClock Mcu_SetMode	0x000000FF	0x000000** (After Mcu_Init, Mcu_InitClock and Mcu_SetMode. Digit * depends on the configuration value.)
CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.CNT_STAT	31:0	Word (32 bits)	-	Clock supervision counters for backup clock	Not used.	0x00000000 (Monitoring is not required.)	0x00000000 (Monitoring is not needed.)

8.9 CM0P_SCS

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SYST_CSR	31:0	Word (32 bits)	-	Cortex®-M0+ SysTick control & status	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CPUID	31:0	Word (32 bits)	-	Cortex®-M0+ CPUID register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
AIRCR	31:0	Word (32 bits)	-	Cortex®-M0+ application interrupt and reset control register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SCR	31:0	Word (32 bits)	0x00000000 | (pending interrupt enable << 4) | (deepsleep enable << 2) | (sleep on exit enable << 1)	Cortex®-M0+ system control register	Mcu_SetMode	0x00000016	0x000000** (After Mcu_SetMode. Digit * depends on configuration value.)

8.10 CM4_SCS

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
ACTLR	31:0	Word (32 bits)	-	Cortex®-M4 Auxiliary control register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SYST_CSR	31:0	Word (32 bits)	-	Cortex®-M4 SysTick control and status register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CPUID	31:0	Word (32 bits)	-	Cortex®-M4 CPUID base register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
AIRCR	31:0	Word (32 bits)	0x00000000 | (register key << 16) | (reset request << 2)	Cortex®-M4 application interrupt and reset control register	Mcu_PerformReset	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SCR	31:0	Word (32 bits)	0x00000000 | (pending interrupt enable << 4) | (deepsleep enable << 2) | (sleep on exit enable << 1)	Cortex®-M4 system control register	Mcu_SetMode	0x00000016	0x000000** (After Mcu_SetMode. Digit * depends on configuration value.)

8.11 CM7_SCS

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
ACTLR	31:0	Word (32 bits)	-	Cortex®-M7 Auxiliary control register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
SYST_CSR	31:0	Word (32 bits)	-	Cortex®-M7 SysTick control and status register	Not used.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CPUID	31:0	Word (32 bits)	-	Cortex®-M7 CPUID base register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
AIRCR	31:0	Word (32 bits)	0x00000000 | (register key << 16) | (reset request << 2)	Cortex®-M7 application interrupt and reset control register	Mcu_PerformReset	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Register	Bit No.	Access size	Value	Description	Timing	Mask value	Monitoring value
SCR	31:0	Word (32 bits)	0x00000000 | (pending interrupt enable << 4) | (deepsleep enable << 2) | (sleep on exit enable << 1)	Cortex®-M7 system control register	Mcu_SetMode	0x00000016	0x000000** (After Mcu_SetMode. Digit * depends on configuration value.)
CCR	31:0	Word (32 bits)	0x00000000 | (I-cache enable << 17) | (D-cache enable << 16)	Cortex®-M7 configuration and control register	Mcu_SetMode Mcu_PerformReset	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CCSIDR	31:0	Word (32 bits)	-	Cortex®-M7 cache size ID register	Read only.	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
CSSELR	31:0	Word (32 bits)	0x00000000	Cortex®-M7 cache size selection register	Mcu_SetMode Mcu_PerformReset	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
ICIALLU	31:0	Word (32 bits)	0x00000000	Cortex®-M7 instruction cache invalidate all	Mcu_SetMode Mcu_PerformReset	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DCISW	31:0	Word (32 bits)	0x00000000 | (way << 30) | (set << 5)	Cortex®-M7 data cache invalidate by set/way	Mcu_SetMode Mcu_PerformReset	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)
DCCSW	31:0	Word (32 bits)	0x00000000 | (way << 30) | (set << 5)	Cortex®-M7 data cache clean by set/way	Mcu_SetMode Mcu_PerformReset	0x00000000 (Monitoring is not needed.)	0x00000000 (Monitoring is not needed.)

Revision history

Revision	Issue date	Description of change
**	2020-08-07	Initial release.
*A	2020-11-19	Changed a memmap file include folder in chapter 2.6. Added section 5.9 APIs Require Privileged Execution. MOVED TO INFINEON TEMPLATE.
*B	2021-05-19	Added the error case description in section 5.4. Added note for McuCsvAction in 4.5.2.15 MCU Clock Supervisor Settings Added note for McuHvLvdAction in 4.6.2 MCU HVLVD Settings Added note for McuVddaBodAction and McuVddaOvdAction in 4.6.3 MCU Supply Supervision Settings Added the section 6.3 Fault report structure. Add description about following configuration parameters. McuHibernateWakeupByBackupCsvEnable in 4.6.1 MCU HIBRENATE Mode Settings McuHibernateWakeupSenseMode in 4.6.1 MCU HIBRENATE Mode Settings Modified following configuration parameters. McuHibernateWakeupByWakeupPin<n>Enable in 4.6.1 MCU HIBRENATE Mode Settings McuHibernateWakeupPin<n>Polarity in 4.6.1 MCU HIBRENATE Mode Settings Added value for following configuration parameters. McuCsvClock in 4.5.2.15 MCU Clock Supervisor Settings McuCsvStartupDelay in 4.5.2.15 MCU Clock Supervisor Settings Added note for following configuration parameters. McuSscgPllModulationDitheringEnable in 4.5.2.4 MCU SSCG PLL Clock Settings McuPumpClockSettings in 4.5.2.12 MCU Pump Clock Settings McuCsvStartupDelay in 4.5.2.15 MCU Clock Supervisor Settings Added description for following registers. PWR_HIB_WAKE_CTL in 8.6 SRSS PWR_HIB_WAKE_CTL2 in 8.6 SRSS PWR_HIB_WAKE_CAUSE in 8.6 SRSS CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.REF_CTL in 8.7 BACKUP CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.REF_LIMIT in 8.7 BACKUP CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.MON_CTL in 8.7 BACKUP CSV_BAK_STRUCT.CSV_DPSLP_STRUCT.CNT_STAT in 8.7 BACKUP Modified description for following registers PWR_HIBERNATE in 8.6 SRSS CSV_LF_STRUCT.CSV_DPSLP_STRUCT.REF_CTL in 8.6 SRSS CSV_ILO_STRUCT.CSV_DPSLP_STRUCT.REF_CTL in 8.6 SRSS Added note for following configuration parameters. McuImoEnable in 4.5.1 MCU Clock Input.

Revision	Issue date	Description of change
		McuIlo0MonitorEnable in 4.5.1.6 MCU ILO Clock Settings McuIlo1MonitorEnable in 4.5.1.7 MCU ILO1 Clock Settings Modified note for following configuration parameters. McuFast0ClockFrequency, McuFast0ClockDivision, McuFast1ClockFrequency, and McuSlowClockFrequency in 4.5.2 MCU Clock Settings McuSscgPllFrequency and McuSscgPllModulationRate in 4.5.2.4 MCU SSCG PLL Clock Settings McuPclkFrequency in 4.5.2.8 MCU PCLK Settings McuPeriGroupClockFrequency in 4.5.2.9 MCU Peripheral Group Settings McuPeriGroupSlaveEnable in 4.5.2.10 MCU Peripheral Group Slave Settings McuLfClockSource in 4.5.2.11 MCU LF Clock Settings McuFreezeIoRelease and McuDeepSleepRegulatorDisable in 4.6 MCU Mode Settings Configuration Deleted note for following configuration parameters. MCU_E_CLOCK_FAILURE and MCU_E_RESET_FAILURE in 4.4 MCU DEM Event Parameter References McuFllAutoDistributeEnable and McuFllAutoDistributeType in 4.5.2.2 MCU FLL Clock Settings McuHvLvdType in 4.6.2 MCU HVLVD Settings Modified description about folllowing configuration parameters. McuEcoAmplitudeTrimValue and McuEcoWatchdogTrimValue in 4.5.1.3 MCU ECO Trim Settings McuSscgPllModulationRate in 4.5.2.4 MCU SSCG PLL Clock Settings
*C	2021-05-25	Added note for following configuration parameters. McuRegHcSettings in 4.6.4 MCU REGHC Settings McuPmicSettings in 4.6.5 MCU PMIC Settings
*D	2021-06-25	Added the definition of WFI in Abbreviations and definitions Added note about Hibernate mode entry in section 5.3 MCU Mode Deleted value for McuSscgPllModulationMode in 4.5.2.4 MCU SSCG PLL Clock Settings Added note for following configuration parameters. McuFllAutoDistributeEnable and McuFllAutoDistributeType in 4.5.2.2 MCU FLL Clock Settings McuLinearCoreRegulatorDisable in 4.6 MCU Mode Settings Configuration Deleted note for McuVoltageReferenceBufferDisable in 4.6 MCU Mode Settings Configuration
*E	2021-08-19	Added a note in 6.2 Interrupts
*F	2021-12-21	Updated to the latest branding guidelines.
*G	2022-02-14	Added a note for following configuration parameters: McuEnableCacheFlushBeforeReset in 4.2 MCU module configuration McuRam0Macro<n>RetainBeforeReset in 4.2 MCU module configuration McuRam1RetainBeforeReset in 4.2 MCU module configuration

Revision	Issue date	Description of change
		McuRam2RetainBeforeReset in 4.2 MCU module configuration McuEnableCacheFlushBeforeModeChange in 4.6 MCU mode settings McuRam0Macro<n>PowerMode in 4.6 MCU mode settings McuRam1PowerMode in 4.6 MCU mode settings McuRam2PowerMode in 4.6 MCU mode settings
*H	2022-07-11	Added description for the McuHibernateClearPendingWakeup configuration parameter in 4.6.1 MCU hibernate settings Added a note for the McuWcoPrescaler configuration parameter in 4.5.1.8 MCU WCO settings Modified a note for the following configuration parameters: McuUpdateSystemResource in 4.6 MCU mode settings McuLinearCoreRegulatorDisable in 4.6 MCU mode settings McuRegHcSettings in 4.6.4 MCU REGHC Settings McuPmicSettings in 4.6.5 MCU PMIC Settings Deleted a note about REGHC in section 5.3 MCU Mode Deleted the description for the following registers: PWR_REGHC_STATUS in 8.6 SRSS PWR_REGHC_CTL in 8.6 SRSS PWR_REGHC_CTL2 in 8.6 SRSS PWR_REGHC_CTL4 in 8.6 SRSS PWR_PMIC_STATUS in 8.6 SRSS PWR_PMIC_CTL in 8.6 SRSS PWR_PMIC_CTL2 in 8.6 SRSS PWR_PMIC_CTL4 in 8.6 SRSS
*I	2022-09-28	Added a note for the following configuration parameters: McuLpEcoAmplitudeDetectorEnable in 4.5.1.4 MCU LPECO clock settings McuBandgapReferencePowerMode in 4.6 MCU mode settings configuration
*J	2023-03-03	Added note about DeepSleep entry in section 5.3 MCU Mode Add description about following configuration parameters: McuFast2ClockFrequency in 4.5.2 MCU clcok settings McuFast2ClockDivision in 4.5.2 MCU clcok settings McuFast3ClockFrequency in 4.5.2 MCU clcok settings McuFast3ClockDivision in 4.5.2 MCU clcok settings McuFlash1WaitCycle in 4.5.2 MCU clcok settings McuMainCore2PowerMode in 4.6 MCU mode settings configuration McuMainCore3PowerMode in 4.6 MCU mode settings configuration McuMainCore2PowerUpDelay in 4.6 MCU mode settings configuration McuMainCore3PowerUpDelay in 4.6 MCU mode settings configuration Added value for following configuration parameters: McuTargetCpu in 4.6 MCU mode settings configuration Added description for following registers: FAST_2_CLOCK_CTL in 8.2 CPUSS

Revision	Issue date	Description of change
		FAST_3_CLOCK_CTL in 8.2 CPUSS CM7_2_PWR_CTL in 8.2 CPUSS CM7_2_PWR_DELAY_CTL in 8.2 CPUSS CM7_3_PWR_CTL in 8.2 CPUSS CM7_3_PWR_DELAY_CTL in 8.2 CPUSS Added section 8.6 FLASHC1 Modified description of Mcu_StatusType in 7.1.9 Mcu_StatusType Added description about MCU_CM7_2_CORE and MCU_CM7_3_CORE in 7.2.5 Core ID value Added description about return value of Mcu_GetCoreID in 7.3.13 Mcu_GetCoreID
*K	2023-06-06	Modified description in 2.6.1 Memory allocation keyword Added a note for the following configuration parameters: McuPeriGroup in 4.5.2.9 MCU peripheral group settings
*L	2023-10-06	Added a note for the McuClockOutputSettings in 4.5.2.14 MCU clock output settings. Corrected core identification keyword on chapter 2.6
*M	2023-12-08	Web release. No content updates.
*N	2024-03-18	Added a note for following configuration parameters. McuDeepSleepRegulatorDisable in 4.6 MCU mode settings configuration McuVoltageReferenceBufferDisable in 4.6 MCU mode settings configuration
*O	2024-09-20	Added note about LVD setting in section 5.3 MCU Mode
*P	2025-04-24	Added a point in section 2.6.1 Memory allocation keyword Added descripton for MCU_CORE[ClockCoreId]_START_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32, MCU_CORE[ClockCoreId]_STOP_SEC_VAR_CLEARED_ASIL_B_GLOBAL_32 Added a note in section 4.5.2.5 MCU clock root settings

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

Edition 2025-04-24

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email:

erratum@infineon.com

Document reference

002-30199 Rev. *P

Important notice

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

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.