MICROSAR MCU

Technical Reference

MCAL Emulation in VTT

Version 1.1.0

Authors	Peter Lang, Christian Leder
Status	Released

Document Information

History

Author	Date	Version	Remarks
Peter Lang	2013-09-17	1.00.00	Initial version
Christian Leder	2015-02-17	1.01.00	> Global renaming of Vip to Vtt > Usage of template 5.11.0 for the Technical reference

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_MCUDriver.pdf	V3.2.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	V4.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.6.0
[5]	AUTOSAR	AUTOSAR_SWS_ECUStateManager.pdf	V3.0.0

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

Contents

Illustrations

Tables

1 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.0.x	Initial version of the Vip MCU driver
2.0.x	Global renaming of Vip to Vtt

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module MCU as specified in [1].

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile	
Vendor ID:	MCU_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	MCU_MODULE_ID	101 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

The MCU driver normally provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required from other MCAL software modules.

The main tasks of the MCU driver are:

- Initialization of MCU clock, PLL, clock prescalers and MCU clock distribution
- Initialization of RAM sections
- Activation of reduced power modes
- Performing resets
- Provide services to get the reset reason from the hardware.

This MCU driver has not the ability to modify clock settings due to the fact that the VTT is an emulated environment. Also, Ram initialisation is not supported.

2.1 Architecture Overview

The following figure shows where the MCU is located in the AUTOSAR architecture.

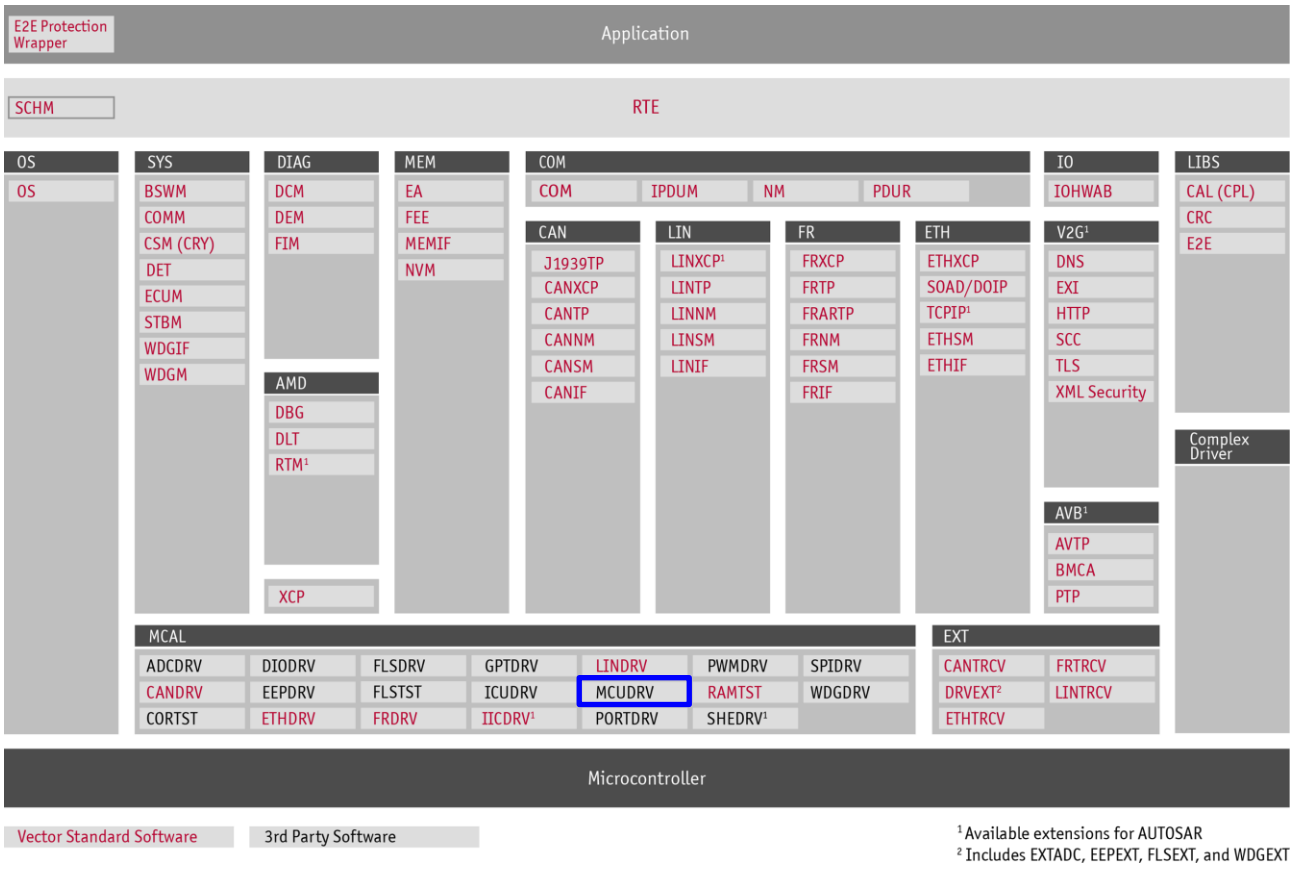

Figure 2-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the MCU. These interfaces are described in chapter 5.

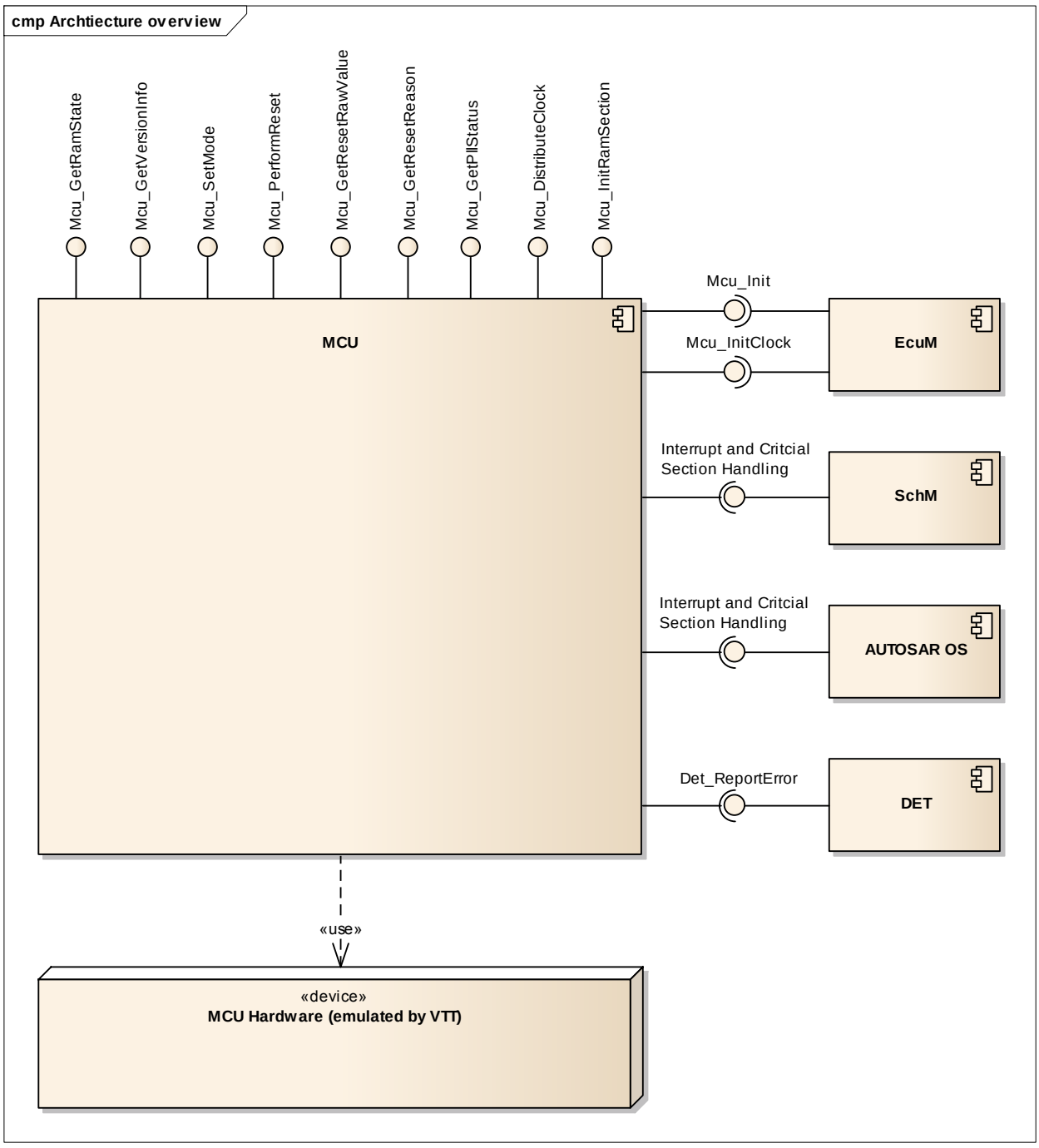

Figure 2-2 Interfaces to adjacent modules of the MCU

Note

Applications do not access the services of the BSW modules directly. Most of the APIs of the MCU module are used by the ECUM as specified in [5].

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the MCU.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 3-1 Supported AUTOSAR standard conform features

> Table 3-2 Not supported AUTOSAR standard conform features

Vector Informatik provides further MCU functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Initialization of MCU clock and PLL (API provided, but without functionality)
MCU Clock distribution (API provided, but without functionality)
Initialization of RAM Sections (API provided, but without functionality)
Activation of µC reduced power modes
Activation of a µC reset
Report of reset reason

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

The following features specified in [1] are not supported:

Not Supported AUTOSAR Standard Conform Features
Ram initialisation
`MCU_E_PLL_NOT_LOCKED` will not be reported to DET module due to the fact that this is an emulation of the MCU driver

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
In addition to the existing checks required by the AUTOSAR standard, the parameter versioninfo passed to the service `Mcu_GetVersionInfo()` is checked for not referencing `NULL_PTR`. If it does, the error `MCU_E_PARAM_VINFO` is reported to DET instead of `MCU_E_PARAM_POINTER`

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.3 Limitations

3.1.3.1 Diagnostic Event Manager

Due to the fact that the MCU is emulated, reporting of hardware errors to the DEM is not supported. Because of compatibility reasons, the DEM has to be configured in DaVinci Configurator.

3.2 Emulation

This driver is an emulation of an MCU module.

Caution

Be careful using while loops in order to poll any status.

The user has to ensure, that the application does not block the emulation. So, within every while loop the following function call has to be called:

```
while(ANY_STATUS == temp_status)
{
  Schedule();
}
```

Use the function call Schedule() which is available once the header file of the module MCU is included.

3.3 Initialization

The MCU module is being initialized by calling `Mcu_Init(&McuModuleConfiguration)`.

All global variables are initialized by calling `Mcu_InitMemory()`. So, `Mcu_InitMemory()` has to be called prior to `Mcu_Init()`.

3.4 States

The MCU module does not implement a state machine.

3.5 Main Functions

The MCU module does not provide any cyclic main functions.

3.6 Error Handling

3.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `MCU_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported MCU ID is 101.

The reported service IDs identify the services which are described in 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	Mcu_Init
0x01	Mcu_InitRamSection
0x02	Mcu_InitClock
0x03	Mcu_DistributeClock
0x04	Mcu_GetPllStatus
0x05	Mcu_GetResetReason
0x06	Mcu_GetResetRawValue
0x07	Mcu_PerformReset
0x08	Mcu_SetMode
0x09	Mcu_GetVersionInfo
0x04	Mcu_GetRamState

Table 3-4 Service IDs

The errors reported to DET are described in the following table:

Error Code		Description
0x0A	`MCU_E_PARAM_CONFIG`	`Mcu_Init` called with ‘`versioninfo`’ referencing `NULL_PTR`
0x0B	`MCU_E_PARAM_CLOCK`	`Mcu_InitClock` called with invalid ‘`ClockSetting`’ parameter
0x0C	`MCU_E_PARAM_MODE`	`Mcu_SetMode` called with invalid ‘`McuMode`’ parameter
0x0D	`MCU_E_PARAM_RAMSECTION`	`Mcu_InitRamSection` called with invalid ‘`RamSection`’ parameter
0x0E	`MCU_E_PLL_NOT_LOCKED`	Not supported
0x0F	`MCU_E_UNINIT`	API called when the MCU driver was not initialized before
0x15	`MCU_E_PARAM_VINFO`	`Mcu_GetVersionInfo` called with ‘`ConfigPtr`’ referencing `NULL_PTR`

Table 3-5 Errors reported to DET

3.6.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 3-6 are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled.

The following table shows which parameter checks are performed on which services:

Service \ Check	MCU_E_PARAM_CONFIG	MCU_E_UNINIT	MCU_E_PARAM_CLOCK	MCU_E_PARAM_MODE	MCU_E_PARAM_RAMSECTION	MCU_E_PARAM_VINFO	MCU_E_PLL_NOT_LOCKED
Mcu_Init	■						
Mcu_InitClock		■	■	■			
Mcu_DistributePllClock		■					
Mcu_InitRamSection		■			■		
Mcu_GetPllStatus		■					
Mcu_GetResetReason		■					
Mcu_PerformReset		■					
Mcu_SetMode		■		■			
Mcu_GetVersionInfo						■	
Mcu_GetResetRawValue		■					

Table 3-6 Development Error Reporting: Assignment of checks to services

3.6.2 Production Code Error Reporting

Info
Production errors are not supported in this emulation.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR MCU into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the MCU contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
Mcu.h	The module header defines the interface of the MCU. This file must be included by upper layer software components
Mcu.c	This C-source contains the implementation of the module's functionalities
DrvMcu_VttCanoe01Asr.jar	This jar-file contains the generator and the validator for the DaVinci Configurator
VTTMcu_bswmd.arxml	Basic Software Module Description according to AUTOSAR for VTT Emulation
Mcu_bswmd.arxml	Optional Basic Software Module Description. Placeholder for real target (semiconductor manufacturer) in VTT only use case

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool DaVinci Configurator.

File Name	Description
Mcu_Cfg.h	The configuration-header contains the static configuration part of this module
Mcu_PBcfg.c	The configuration-source contains the object independent part of the runtime configuration

Table 4-2 Generated files

4.2 Include Structure

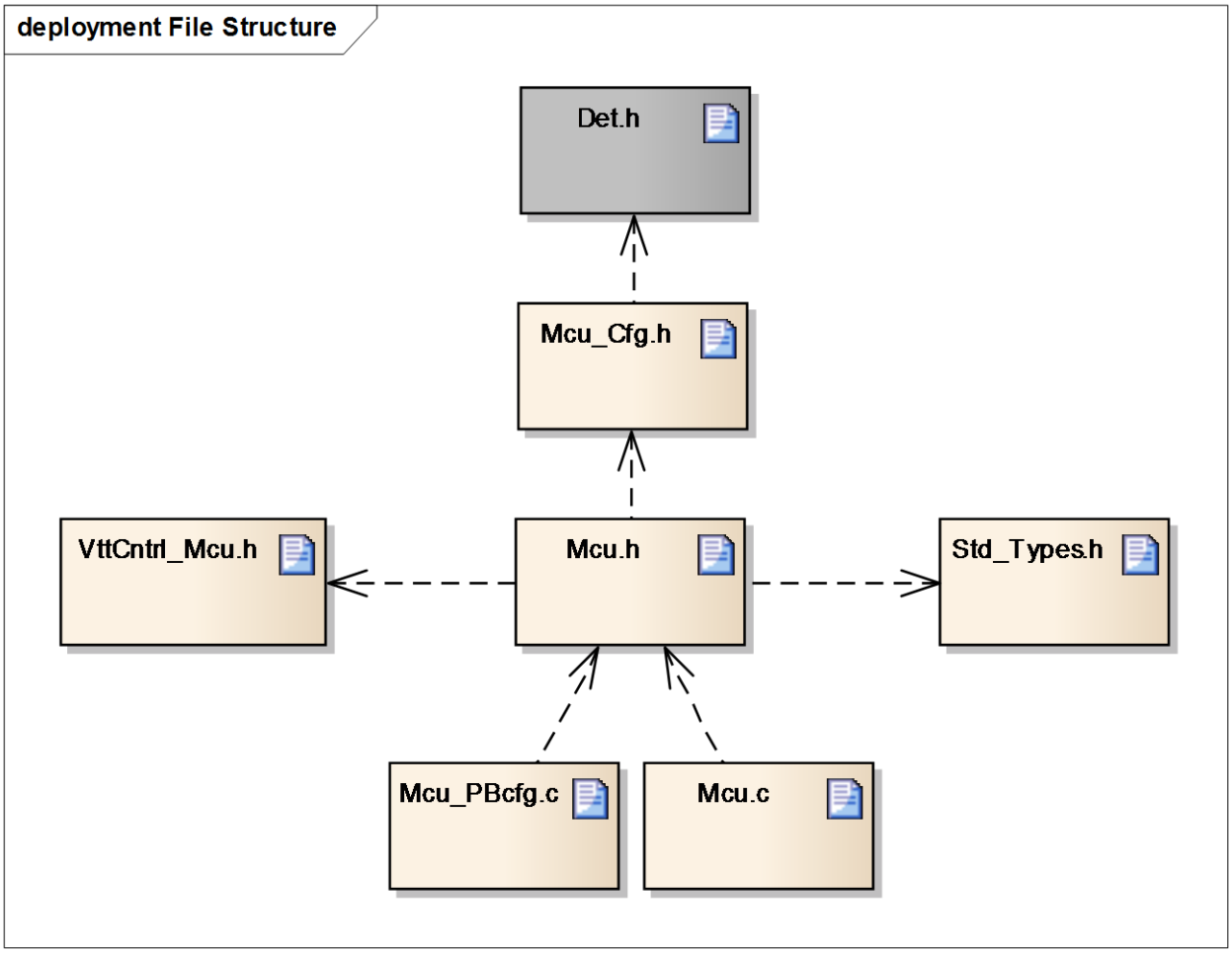

Figure 4-1 Include Structure

4.3 Dependencies on SW Modules

4.3.1 AUTOSAR OS (Optional)

An operating system can be used for task scheduling, interrupt handling, global suspend and restore of interrupts and creating of the Interrupt Vector Table.

4.3.2 DET (Optional)

The MCU module depends on the DET (by default) in order to report development errors. Detection and reporting of development errors can be enabled or disabled by the switch "Enable Development Error Detection".

4.3.3 SchM (Optional)

Beside the AUTOSAR OS the Schedule Manager provides functions that module MCU calls at begin and end of critical sections.

4.3.4 EcuM (Optional)

The ECUM cares for the initialization of the module MCU. The ECUM also uses the module MCU for setting the µC in low power mode or for performing a reset.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Type Definitions

The types defined by the MCU are described in this chapter.

Type Name	C-Type	Description	Value Range
Mcu_PllStatusType	uint8	This is a status value returned by the API service `Mcu_GetPllStatus()` of MCU driver	`MCU_PLL_LOCKED` The PLL is locked
			`MCU_PLL_UNLOCKED` The PLL is not locked
			`MCU_PLL_STATUS_UNDEFINED` The function
			`Mcu_GetPllStatus()` has been called before the module was initialized
Mcu_ResetType	enum	This is the type of the reset enumerator containing the subset of reset types	`MCU_POWER_ON_RESET` The last reset was a power on reset
			`MCU_WATCHDOG_RESET` The last reset was a watchdog reset
			`MCU_ILLEGALADDRESS_RESET` The last reset was an illegal address reset
			`MCU_LOW_VOLTAGE_RESET` The last reset was a low voltage reset
			`MCU_EXTERNAL_RESET` The last reset was an external reset (e.g. external watchdog)
			`MCU_RESET_UNDEFINED` The last reset was undefined
			`MCU_SW_RESET` The last reset was a software reset
Mcu_ClockType	uint8	Specifies the identification (ID) for a clock setting.	0 - 255

Mcu_ModeType	uint8	Specifies the identification (ID) for a mode setting.	`MCU_MODE_SLEEP = 0`
			`MCU_MODE_RESET = 1`
			`MCU_MODE_POWER_OFF = 2`

Mcu_RamSectionType	uint8	Specifies the identification (ID) for a	0 - 255

Type Name	C-Type	Description	Value Range
		RAM section setting.	
Mcu_RamStateType	enum	This is the Ram State data type returned by the function `Mcu_GetRamState` of the MCU module. It is not required that all RAM state types are supported by the hardware.	`MCU_RAMSTATE_INVALID` Ram content is not valid or unknown (default).
			`MCU_RAMSTATE_VALID` Ram content is valid.
Mcu_RawResetType	uint8	Specifies the return value of the function `Mcu_GetRawResetValue()`	0

Table 5-1 Type definitions

5.2 Services provided by MCU

5.2.1 Mcu_InitMemory

Prototype	
`void` **`Mcu_InitMemory`** `(void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
This service initializes the global variables in case the startup code does not work	
Particularities and Limitations	
> This function is synchronous. > This function is nonreentrant. > Module must not be initialized.	
Expected Caller Context	
> Called during startup	

Table 5-2 Mcu_InitMemory

5.2.2 Mcu_Init

Prototype	
`void Mcu_Init (const Mcu_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Pointer to MCU driver configuration set
Return code	
-	-
Functional Description	
This routine initializes the MCU driver. It has no functionality, except setting the module's internal status to 'initialized'.	
Particularities and Limitations	
> This service is synchronous. > This service is non reentrant. > Module must not be initialized.	
Expected Caller Context	
> ECU State Manager or comparable software module, responsible for driver initialization after startup.	

Table 5-3 Mcu_Init

5.2.3 Mcu_InitRamSection

Prototype	
`Std_ReturnType Mcu_InitRamSection (Mcu_RamSectionType RamSection )`	
Parameter	
`RamSection`	Selects RAM memory section provided in configuration set
Return code	
`Std_ReturnType`	`E_OK`, request accepted `E_NOT_OK`, request not accepted
Functional Description	
This function initializes the RAM section wise. It has no functionality, except checking for development errors. The parameter is ignored. The function always returns `E_OK`.	
Particularities and Limitations	
> This service is synchronous. > This service is non reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> ECU State Manager or comparable software module, responsible for driver initialization after startup.	

Table 5-4 Mcu_InitRamSection

5.2.4 Mcu_InitClock

Prototype	
`Std_ReturnType` **`Mcu_InitClock`** `(Mcu_ClockType ClockSetting )`	
Parameter	
`ClockSetting`	Number of the clock setting configuration to be used.
Return code	
`Std_ReturnType`	`E_OK`, initialization successful `E_NOT_OK`, initialization not successful
Functional Description	
This function normally initializes the PLL and other MCU specific clock options. In this case, it has no functionality, except checking for development errors. The parameter is ignored. The function always returns `E_OK`.	
Particularities and Limitations	
> This service is synchronous. > This service is non reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> ECU State Manager or comparable software module, responsible for driver initialization after startup.	

Table 5-5 Mcu_InitClock

5.2.5 Mcu_DistributePllClock

Prototype	
`void` **`Mcu_DistributePllClock`**`(void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
This routine has no functionality, except checking for development errors.	
Particularities and Limitations	
> This service is synchronous. > This service is non reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> Expected to be called in application context	

Table 5-6 Mcu_DistributePllClock

5.2.6 Mcu_GetPllStatus

Prototype	
`Mcu_PllStatusType` **`Mcu_GetPllStatus`**`(void)`	
Parameter	
-	-
Return code	
`MCU_PLL_LOCKED`	PLL locked
Functional Description	
This service has no functionality, except checking for development errors. The function always returns `MCU_E_PLL_UNDEFINED`.	
Particularities and Limitations	
> This service is synchronous. > This service is reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> Expected to be called in application context	

Table 5-7 Mcu_GetPllStatus

5.2.7 Mcu_GetResetReason

Prototype	
`Mcu_ResetType` **`Mcu_GetResetReason`**`(void)`	
Parameter	
-	-
Return code	
`Mcu_ResetType`	Provides the last reset reason value
Functional Description	
Provides the last reset reason	
Particularities and Limitations	
> This service is synchronous. > This service is reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> Expected to be called in application context	

Table 5-8 Mcu_GetResetReason

5.2.8 Mcu_GetResetRawValue

Prototype	
`Mcu_RawResetType` **`Mcu_GetResetRawValue`**`(void)`	
Parameter	
-	-
Return code	
`Mcu_RawResetType`	Provides the last reset reason.
Functional Description	
Funktionality and return value are equal to the Mcu_GetResetReason API	
Particularities and Limitations	
> This service is synchronous. > This service is reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> Expected to be called in application context	

Table 5-9 Mcu_GetResetRawValue

5.2.9 Mcu_PerformReset

Prototype	
`void` **`Mcu_PerformReset`**`(void)`	
Parameter	
-	-
Return code	
-	-
Functional Description	
Performs a reset of the VTT environment	
Particularities and Limitations	
> This service is synchronous. > This service is non reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> Expected to be called in application context	

Table 5-10 Mcu_PerformReset

5.2.10 Mcu_SetMode

Prototype	
`void Mcu_SetMode (Mcu_ModeType McuMode)`	
Parameter	

`McuMode`	Number of the MCU power modes configured in the configuration set.
	Available modes are:
	`MCU_MODE_SLEEP`
	`MCU_MODE_RESET`
	`MCU_MODE_POWER_OFF`

Return code	
-	-
Functional Description	
Sets the VTT environment into the expected mode	
Particularities and Limitations	
> This service is synchronous. > This service is non reentrant. > This service may only be called if the module has been initialized before.	
Expected Caller Context	
> Expected to be called in application context	

Table 5-11 Mcu_SetMode

5.2.11 Mcu_GetVersionInfo

Prototype	

`void Mcu_GetVersionInfo`
`(`
`  P2VAR(Std_VersionInfoType, AUTOMATIC, MCU_APPL_DATA) versioninfo`
`)`	

Parameter	
`versioninfo`	Pointer to the VersionInfo structure
Return code	
-	-
Functional Description	
This function returns the version information of the module. The version information includes: > Module Id > Vendor Id > Software version numbers	
Particularities and Limitations	
> This service is synchronous. > This service is reentrant. > This service is configurable.	

Expected Caller Context
> Expected to be called in application context

Table 5-12 Mcu_GetVersionInfo

5.2.12 Mcu_GetRamState

Prototype	
`Mcu_RamStateType` **`Mcu_GetRamState`** `(void)`	
Parameter	
-	-
Return code	
`Mcu_RamStateType`	Status of the Ram Content
Functional Description	
This service provides the actual status of the microcontroller Ram.	
Particularities and Limitations	
> This service is synchronous. > This service is reentrant.	
Expected Caller Context	
> Expected to be called in application context	

Table 5-13 Mcu_GetRamState

5.3 Services used by MCU

In the following table services provided by other components, which are used by the MCU are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
DET	Det_ReportError

Table 5-14 Services used by the MCU

6 Configuration

6.1 Configuration Variants

The MCU supports the configuration variants

> VARIANT-PRE-COMPILE

The configuration classes of the MCU parameters depend on the supported configuration variants. For their definitions please see the VTTMcu_bswmd.arxml file.

6.2 Configuration with DaVinci Configurator 5

The MCU module is configured with the help of the configuration tool DaVinci Configurator 5 (CFG5). The definition of each parameter is given in the corresponding BSWMD file.

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
CANoe	Tool for simulation and testing of networks and electronic control units.
DaVinci Configurator	Configuration and generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BSWMD	Basic Software Module Description
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
EcuM	ECU State Manager
MCU	Microcontroller Unit
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
OS	Operating System
PLL	Phase-Locked Loop
RTE	Runtime Environment
SchM	BSW Module Scheduler
VTT	vVIRTUALtarget

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com