User's Manual

AUTOSAR MCAL R4.0.3

User's Manual

MCU Driver Component Ver.1.0.8
Embedded User's Manual

Target Device:
RH850/P1x

Renesas Electronics
www.renesas.com

Rev.1.00 Oct 2016

Abbreviations and Acronyms

Abbreviation / Acronym	Description
ADC	Analog to Digital Converter
ANSI	American National Standards Institute
API	Application Programming Interface
AUTOSAR	AUTomotive Open System ARchitecture
CAN	Controller Area Network
CVM	Core Voltage Monitor
CLMA	Clock Monitor
DEM/Dem	Diagnostic Event Manager
DET/Det	Development Error Tracer
DIO	Digital Input Output
DMA	Direct Memory Access
ECU	Electronic Control Unit
EEPROM	Electrically Erasable Programmable Read-Only Memory
ECM/Ecm	Error Control Module
GNU	GNU’s Not Unix
GPT	General Purpose Timer
HW	HardWare
ICU	Input Capture Unit
ID/Id	IDentifier
ISR	Interrupt Service Routine
I/O	Input and Output
KB	Kilo Byte
LIN	Local Interconnect Network
MCAL	Microcontroller Abstraction Layer
MCU/Mcu	MicroController Unit
NA	Not Applicable
NMI	Non Maskable Interrupt
MI	Maskable Interrupt
OS/Os	Operating System
PWM	Pulse Width Modulation
PLL	Phase Locked Loop
RAM/Ram	Random Access Memory
ROM	Read Only Memory
RTE	Run Time Environment
SPI	Serial Peripheral Interface
SW	SoftWare
WDT	WatchDog Timer

Definitions

Term	Represented by
Sl. No.	Serial Number

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

The purpose of this document is to describe the information related to MCU Driver Component for Renesas P1x microcontrollers.

This document shall be used as reference by the users of MCU Driver Component. The system overview of complete AUTOSAR architecture is shown in the below Figure:

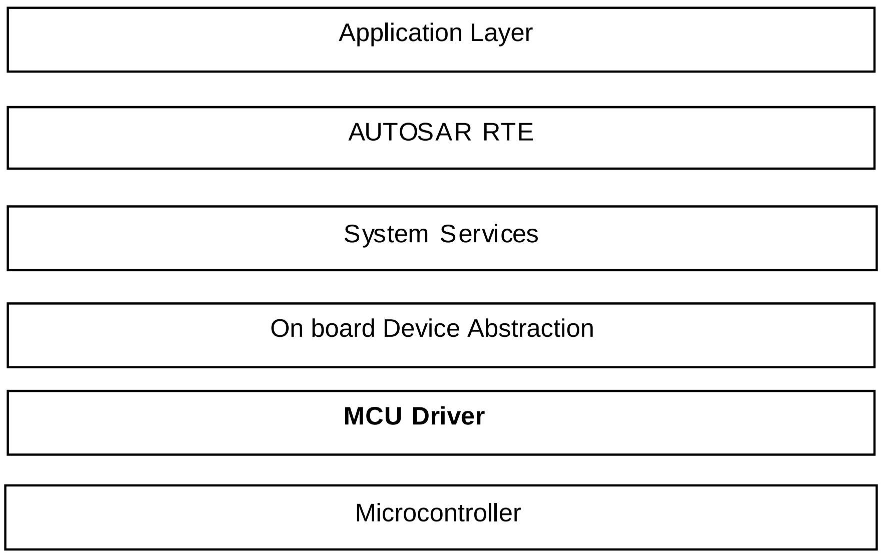

Figure 1-1 System Overview Of AUTOSAR Architecture

The MCU Driver is part of the Microcontroller Abstraction Layer (MCAL), the lowest layer of Basic Software in the AUTOSAR environment.

The Figure in the following page depicts the MCU Driver as part of layered AUTOSAR MCAL Layer:

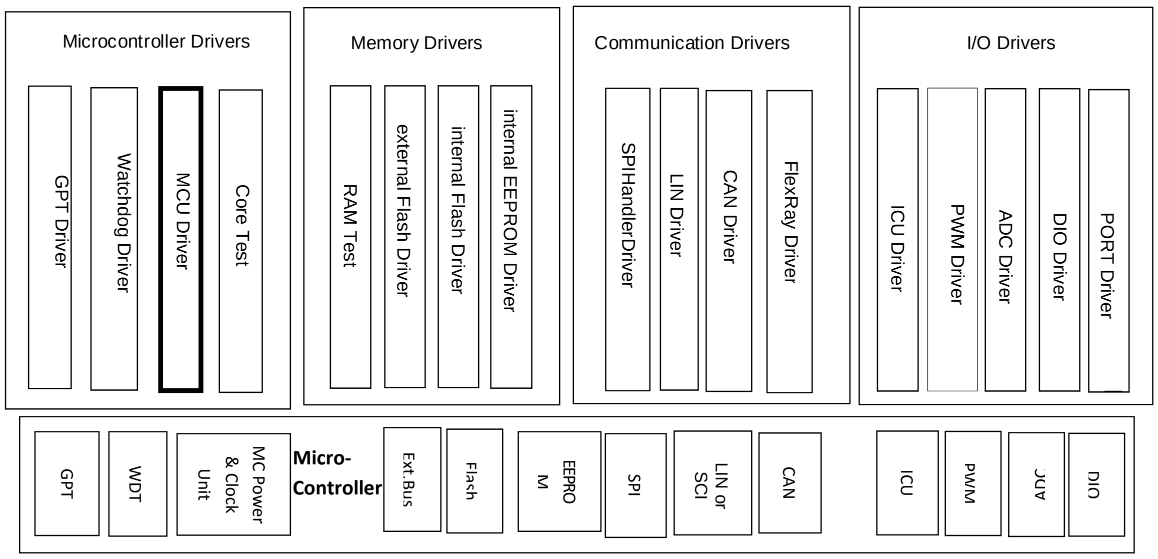

Figure 1-2 System Overview Of The MCU Driver In AUTOSAR MCAL Layer

The RTE provides the encapsulation of Hardware channels and basic services to the Application Software Components. So it is possible to map the Application Software-Components between different ECUs.

The Basic Software Modules are located below the RTE. The Basic Software itself is divided into the subgroups: System Services, Memory, Communication and I/O Hardware-Abstraction. The Complex Drivers are also located below the RTE. Among others, the Operating System (OS), the Watchdog manager and the Diagnostic services are located in the System Services subgroup. The Memory subgroup contains modules to provide access to the non-volatile memories, namely Flash and EEPROM. In the I/O Hardware-Abstraction subgroup the whole MCU Driver Component is provided.

On board Device Abstraction provides an interface to physical values for AUTOSAR software components. It abstracts the physical origin of signals (their paths to the hardware ports) and normalizes the signals with respect to their physical appearance. The Microcontroller driver provides services for basic microcontroller initialization, power down functionality, reset and microcontroller specific functions required from the upper layers.

1.1. Document Overview

The document has been segmented for easy reference. The table below provides user with an overview of the contents of each section:

Section	Contents
Section1 (Introduction)	This section provides an introduction and overview of MCU Driver Component.
Section 2 (Reference Documents)	This section lists the documents referred for developing this document.
Section 3 (Integration And Build Process)	This section explains the folder structure, Makefile structure for MCU Driver Component. This section also explains about the Makefile descriptions, Integration of MCU Driver Component with other components, building the MCU Driver Component along with a sample application.
Section 4 (Forethoughts)	This section provides brief information about the MCU Driver Component, the preconditions that should be known to the user before it is used, data consistency details and deviation list.
Section 5 (Architecture Details)	This section describes the layered architectural details of the MCU Driver Component.
Section 6 (Registers Details)	This section describes the register details of MCU Driver Component.
Section 7 (Interaction between The User And MCU Driver Component)	This section describes interaction of the MCU Driver Component with the upper layers.
Section 8 (MCU Driver Component Header And Source File Description)	This section provides information about the MCU Driver Component source files is mentioned. This section also contains the brief note on the tool generated output file.
Section 9 (Generation Tool Guide)	This section provides information on the MCU Driver Component Code Generation Tool.
Section 10 (Application Programming Interface)	This section explains all the APIs provided by the MCU Driver Component.
Section 11 (Development And Production Errors)	This section lists the DET and DEM errors.
Section 12 (Memory Organization)	This section provides the typical memory organization, which must be met for proper functioning of component.
Section 13 (P1M Specific Information)	This section provides P1M specific information also the information about linker compiler and sample application.
Section 14 (Release Details)	This section provides release details with version name and base version.

Chapter 2 Reference Documents

Sl. No.	Title	Version
1.	AUTOSAR_SWS_MCUDriver.pdf	3.2.0
2.	r01uh0436ej111_rh850p1x.pdf	1.11
3.	AUTOSAR_SWS_MemoryMapping.pdf	1.4.0
4.	AUTOSAR_SWS_PlatformTypes.pdf	2.5.0
5.	AUTOSAR_BSW_MakefileInterface.pdf	0.3
6.	AUTOSAR_SWS_CompilerAbstraction.pdf	3.2.0
7.	AUTOSAR BUGZILLA (http://www.autosar.org/bugzilla) Note: AUTOSAR BUGZILLA is a database, which contains concerns raised against information present in AUTOSAR Specifications.	-

Chapter 3 Integration and Build Process

In this section the folder structure of the MCU Driver Component is explained. Description of the Make files along with samples is provided in this section.

Remark The details about the C Source and Header files that are generated by the MCU Driver Generation Tool are mentioned in the Generation Tool User's Manual "R20UT3721EJ0100-AUTOSAR.pdf".

3.1. MCU Driver Component Makefile

The Makefile provided with the MCU Driver Component consists of the GNU Make compatible script to build the MCU Driver Component in case of any change in the configuration. This can be used in the upper level Makefile (of the application) to link and build the final application executable.

3.1.1 Folder Structure

The files are organized in the following folders:

Remark Trailing slash '\' at the end indicates a folder

```
X1X\P1x\modules\mcu\src
                          \Mcu.c
                          \Mcu_Irq.c
                          \Mcu_Ram.c
                          \Mcu_Version.c

X1X\P1x\modules\mcu\include
                          \Mcu.h
                          \Mcu_Debug.h
                          \Mcu_Irq.h
                          \Mcu_PBTypes.h
                          \Mcu_Ram.h
                          \Mcu_Types.h
                          \Mcu_Version.h
                          \Mcu_RegWrite.h

X1X\P1x\modules\mcu\sample_application\<SubVariant>\make\<compiler>
                                      \App_MCU_P1M_Sample.mak
X1X\P1x\modules\mcu\sample_application\<SubVariant>\obj\<Complier>
X1X\P1x\modules\mcu\generator
                              \Mcu_P1x.dll
                              \R403_MCU_P1x_BSWMDT.arxml
X1X\P1x\common_family\generator
                                \Global_Application_P1x.trxml
                                \Sample_Application_P1x.trxml
                                \P1x_translation.h
```

```
                                         \Test_Application_P1x.trxml
X1X\P1x\modules\mcu\user_manual
(User manuals will be available in this folder)
```

Following stubs are required for the successful compilation of MCU module:

```
X1X\common_platform\generic\stubs\4.0.3\Dem
                                                          \include\Dem.h
                                                          \include\ Dem_Cfg.h
                                                          \src\Dem.c
                                                          \make\dem_defs.mak
                                                          \make\dem_rules.mak
                                                          \xml\Dem_Mcu.arxml
X1X\common_platform\generic\stubs\4.0.3\Det
                                                          \includeDet.h
                                                          \src\Det.c
                                                          \make\det_defs.mak
                                                          \make\det_rules.mak
X1X\common_platform\generic\stubs\4.0.3\Os
                                                          \include\Os.h
                                                          \make\os_defs.mak
X1X\common_platform\generic\stubs\4.0.3\SchM
                                                          \include\Rte.h
                                                          \include\SchM_Mcu.h
                                                          \src\SchM_Mcu.c
                                                          \make\rte_defs.mak
                                                          \make\rte_rules.mak
```

Note: 1. <Complier> can be ghs.
2. <AUTOSAR_version> should be 4.0.3.
3. <SubVariant> can be P1M.

Chapter 4 Forethoughts

4.1. General

Following information will aid the user to use the MCU Driver Component software efficiently:

- The MCU Driver does not enable or disable the ECU or Microcontroller power supply. The upper layer should handle this operation.
- The start-up code is ECU and MCU specific. MCU Driver does not implement the start-up code.
- MCU specific initializations such as reset registers, one time writable registers, interrupt stack pointer, user stack pointer and MCU internal watchdog, MCU specific features of internal memory and registers are not implemented by MCU Driver. These initializations should be implemented by the start-up code.
- MCU Driver does not implement any call-back notification functions.
- MCU Driver does not implement scheduled functions.
- The MCU Driver component is implemented as a Post build variant.
- MCU Driver depends on Scheduler and Wake-up source service Modules for disabling all relevant interrupts to protect writing into the protected registers and invoking the ECU state manager functions.
- In P1x PLL clocks are not configurable and it cannot be controlled by software. It works with default values after main oscillator activated. Hence in P1x Mcu driver code Mcu_DistributePllClock() and Mcu_GetPllStatus()API's none of the action are taken care except DET errors.
- The file Interrupt_VectorTable.c provided is just a Demo and not all interrupts will be mapped in this file. So the user has to update the Interrupt_VectorTable.c as per his configuration
- The container 'McuResetReasonConf' is not used for implementation. Since this is coming under the published information and specific to hardware & implementation, the user must not allowed to configure/rename this. So the other vendor specific containers are introduced here to achieve the same functionality. These containers have multiplicity 1 - 1 and have fixed values depends on the reset type.
- The reset reason information from HW registers shall be cleared after reading and processing the information, in order to avoid multiple reset reasons. This should be done in the APIs Mcu_GetResetReason() and Mcu_GetResetRawValue().
- If the RAM init feature is enabled the API Mcu_InitRamSection follows this procedure:

    - Initialize the RAM section.
    - Unmask the maskable interrupts enabled for RAM error signals (within ECM).

  The procedure requires that the complete RAM is initialized before the RAM state functionality is used.
- If the RAM init feature is not enabled the unmasking of maskable interrupts

enabled for RAM error signals (within ECM) will be handled in API Mcu_Init().

If McuRamSectorSetting = FALSE, the local RAM error sources will be configured as part of Mcu_Init API. If McuRamSectorSetting = TRUE, the local RAM error sources will be configured as part of Mcu_InitRamSection API.

- The accesses to HW registers is possible only in the low level driver layer. The MCAL user does never write or read directly from any register, but uses the AUTOSAR standard API provided by the MCAL.
- The default value for the parameter 'McuLoopCount' is ‘5’. The user shall configure the parameter 'McuLoopCount' to standard default value ‘5’ for avoiding unwanted reporting of DEM due to stabilization issues. The maximum value of this parameter 'McuLoopCount' is '255'.
- The parameter McuEcmDelayTimerOverflowValue specifies to configure the overflow value for the ECM delay timer. This value will be compared with ECM delay timer register. The ECM delay timer uses high speed peripheral clock signal for counting.
- The parameter McuEcmErrorOutputMode used to configure the error output mode either in Dynamic or Non-Dynamic. If this parameter is configured as dynamic and parameter McuEcmErrorOutTimer is enabled, OSTM1 output will be used. If the parameter McuEcmErrorOutTimer is disabled, channel 15 in TAUD1 will be used. User should configure timer channels (OSTM1/TAUD1) outside the MCU driver .If timer is not configured, it will affect the expected output behavior of Dynamic Mode.
- Back up RAM0 (BURAM0) is used in the startup self-test procedure.BURAM0 shall not be shared with other peripherals.
- If the parameter McuStartUpSWResetTest or McuStartUpECMResetTest is configured as true, reset will happen during initialisation.
- For handling Non-Maskable interrupts asm code is used to access system registers. User does not have permission to alter the asm code
- McuCvmDiagLockBit needs to be set to TRUE when using in ASIL applications
- If there are more than one interrupt with same priority setting across multiple MCAL modules, the interrupts are invoked as per the priority of the microcontroller interrupt vector table.
- The functions/variables with "STATIC" as pilot tag, provides an indication to the compiler that the function/variable following this tag is a static type, so the scope of static functions/variables is restricted to the file where they are declared. User should take care of the tag name with respect to compiler used.

4.2. Preconditions

Following preconditions have to be adhered by the user, for proper functioning of the MCU Driver Component:

- The Mcu_Cfg.h, Mcu_Cbk.h and Mcu_Reg.h files generated by the MCU Driver component Code Generation Tool must be compiled and linked along with MCU Driver component source files.
- The application has to be rebuilt, if there is any change in the Mcu_Cfg.h file

generated by the MCU Driver component Generation Tool.

- File Mcu_PBcfg.c generated for single configuration set or multiple configuration sets using MCU Driver component Generation Tool can be compiled and linked independently.
- The authorization of the user for calling the software triggering of a hardware reset is not checked in the MCU Driver. This is the responsibility of the upper layer.
- The MCU Driver component needs to be initialized before accepting any request. The API Mcu_Init should be called by the ECU State Manager Module to initialize MCU Driver Component.

  The user should ensure that MCU Driver component API requests are invoked in the correct and expected sequence and with correct input arguments.
- Input parameters are validated only when the static configuration parameter MCU_DEV_ERROR_DETECT is enabled. Application should ensure that the right parameters are passed while invoking the APIs when MCU_DEV_ERROR_DETECT is disabled.
- There are different clock settings possible. For more details, please refer the respective device specific component user manual.
- If the handle of clock setting passed to the API Mcu_InitClock is not configured to any one of the supported clock settings, then the Development Error Detection function is invoked if the static configuration parameter MCU_DEV_ERROR_DETECT is enabled.
- The MCU Driver initializes the clock generator as per the required configuration settings and provides the configured clock sources for the peripherals as applicable. It is the responsibility of the individual drivers to select and initialize the respective driver specific registers as required for their functionality with reference to the clock source provided by the MCU Driver.
- The API Mcu_InitClock is implemented considering its invocation at run time. Hence, there is a possibility of change in the baud rate set by the peripheral drivers if the clock setting is different. Hence, the initialization of the respective drivers after the invocation of Mcu_InitClock, is the responsibility of the user of MCU Driver services.
- A mismatch in the version numbers of header and the source files results in compilation error. User should ensure that the correct versions of the header and the source files are used.
- User must ensure that the Mcu_Init shall be called only once after power on or after reset, as there is no de-intialization function. Calling Mcu_Init() twice with no reset in between is an invalid call sequence.

4.3. Data Consistency

To support the re-entrance and interrupt services, the MCU Driver will ensure the data consistency while accessing its own RAM storage or hardware registers or to prevent any interrupts between the two write instructions of the write protected register and the corresponding write enable register.

The MCU Driver will use SchM_Enter_Mcu_<Exclusive Area> and SchM_Exit_Mcu_<Exclusive Area> functions.

The SchM_Enter_Mcu_<Exclusive Area> function is called before the data needs to be protected and SchM_Exit_Mcu_<Exclusive Area> function is called after the data is accessed.

The flowchart will indicate the flow with the precompile option “McuCriticalSectionProtection” enabled.

The following exclusive area along with scheduler services is used to provide data integrity for shared resources:
MCU_REGISTER_PROTECTION
MCU_VARIABLE_PROTECTION
The functions SchM_Enter_Mcu_<Exclusive Area> and SchM_Exit_Mcu _<Exclusive Area> can be disabled by disabling the configuration parameter ‘McuCriticalSectionProtection’.

The MCU module will use below macro for exclusive area.

```
#define MCU_ENTER_CRITICAL_SECTION(Exclusive_Area) \
                    SchM_Enter_Mcu_##Exclusive_Area()

#define MCU_EXIT_CRITICAL_SECTION(Exclusive_Area) \
                    SchM_Exit_Mcu_##Exclusive_Area()
```

Table 4-1 MCU Driver Protected Resources List

API Name	Exclusive Area Type	Protected Resources
Mcu_Init	MCU_REGISTER_PROTECTION	HW Registers: ECMMESSTR0 ECMESSTC0 IMR0 ECMDTMCTL
Mcu_InitRamSection	MCU_REGISTER_PROTECTION	HW Registers: ECMESSTC0 ECMMESSTR0 ECMESSTC1 ECMMESSTR1
Mcu_GetRamState	MCU_VARIABLE_PROTECTION	Global variable Mcu_GblRAMInitStatus protected.

Note: The worst case critical section value is 7.575 microseconds for MCU_FEINT_ISR.

4.4. User Mode and Supervisor Mode

The below table specifies the APIs which can run in user mode, supervisor mode or both modes

Table 4-2 Supervisor mode and User mode details

Sl.No.	API Name	User Mode	Supervisor Mode	Known limitation in User mode
1.	Mcu_Init	-	x	1. The enabling of the interrupt will not be possible. 2. Critical section protection cannot be enabled
2.	Mcu_InitRamSection	x	x	Critical section protection cannot be enabled
3.	Mcu_InitClock	x	x	-
4.	Mcu_DistributePllClock	x	x	-
5.	Mcu_GetPllStatus	x	x	-
6.	Mcu_GetResetReason	x	x	-
7.	Mcu_GetResetRawValue	x	x	-
8.	Mcu_PerformReset	x	x	-
9.	Mcu_SetMode	x	x	-
10.	Mcu_GetRamState	x	x	Critical section protection cannot be enabled
11.	Mcu_LockStepSelfDiagnosticTest	x	x	Critical section protection cannot be enabled
12.	Mcu_CvmSelfDiagnosticTest	x	x	-
13.	Mcu_ClmaSelfDiagnosticTest	x	x	-
14.	Mcu_EcmSelfDiagnosticTest	x	x	Critical section protection cannot be enabled
15.	Mcu_SaveResetReason	x	x	Critical section protection cannot be enabled

Note: Implementation of Critical Section is not dependent on MCAL. Hence Critical Section is not considered to the entries for User mode in the above table.

4.5. Deviation List

Table 4-3 MCU Driver Deviation List

Sl. No.	Description	AUTOSAR Bugzilla / Mantis
1.	The parameter McuResetSetting from the sub-container McuModuleConfiguration is not considered.	-
2.	The MCU Driver considers the parameters of RAM section configuration as pre-compile parameters, since the number of RAM settings are not known and hence the generation of handles is not possible at post-build-time.	-
3.	The sub-container McuClockReferencePoint in the Clock setting configuration is not used as the reference frequencies specific to various peripheral devices need to be published by MCU Driver component.	-
4.	The parameter McuClockSettingId range in McuClockSettingConfig container is changed from “1 to 255” to “0 to 255” since 0 is valid minimum value for clock setting ID.	54536
5.	If an invalid database is passed as a parameter to API Mcu_Init, DET Error code MCU_E_INVALID_DATABASE is reported to DET.	-

4.6. RAM Initialization

RAM initialization done by an API call to Mcu_InitRamSection must not overwrite other memory sections of static variables. A dedicated memory section shall be defined in linker directive file.

4.7. Callout API

The MCU_RESET_CALLOUT() API is the call out API from the Mcu module which will be called by Mcu_PerformReset() API for the software reset when configuration parameter McuSwResetCall Api is true. This callout API needs to be filled by user to do the software reset. If the configuration parameter McuSwResetCall Api is false, the callout shall not be available and the software reset shall be handled by the MCU itself using HW feature of the SW reset.

Chapter 5 Architecture Details

The MCU Driver architecture is shown in the following figure. The MCU user shall directly use the APIs to configure and execute the MCU conversions:

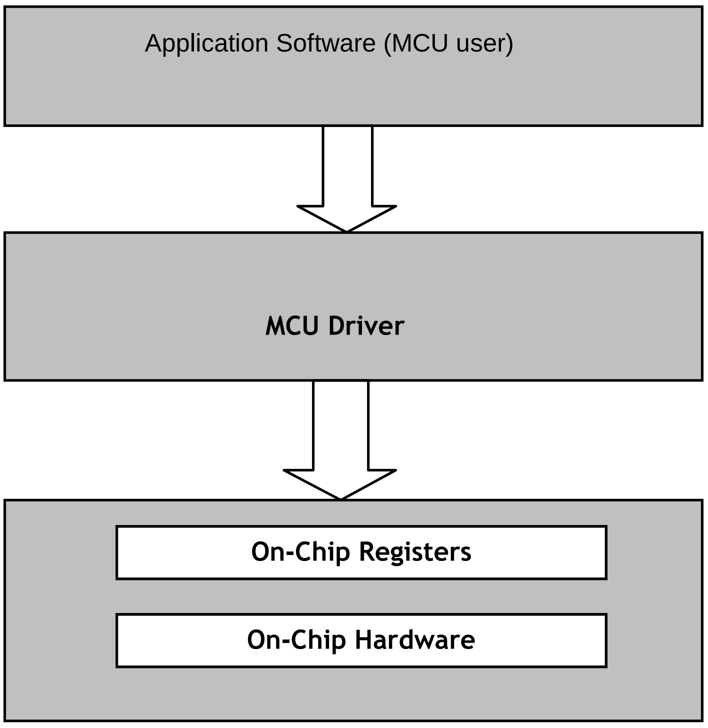

Figure 5-1 MCU Driver Architecture

The MCU driver accesses the microcontroller hardware directly and is located in the MCAL. MCU component provides the functionalities related to PLL Initialization, Clock Initialization and Distribution, RAM sections Initialization, PreScaler Initialization, MCU reduced Power Modes Activation and MCU Reset Activation and Reason.

The component consists of the following sub modules based on the functionality:

- Initialization
- Self-Diagnostic test for ECM, CVM, Clock Monitor and Lock Step.
- Clock Initialization
- RAM sections Initialization and Status Verification
- MCU Reset Activation and Reason
- Version Information

Initialization

This sub module provides the structures and APIs for both global and controller specific initialization. MCU specific initialization is necessary in order to ensure different startup behaviors of the microcontroller. This sub module also checks if the data base is flashed.

Self-Diagnostic test for ECM, CVM, Clock Monitor and Lock Step

This functionality is provided as part MCU module initialization.
Self-diagnostic test for ECM error source is helpful to check the ECM error output signal by creating the real ECM error signal.
Self-diagnostic test for CVM and CLMA is possible in real scenario.

Clock Initialization

The clock initialization sub module provides the functionality for generating all the required clock signals for microcontroller operation from any one of the available sources. It enables the provision for individual clock source selection for CPU and groups of peripherals.

This sub module also provides the functionality for obtaining various frequencies required for individual peripheral devices.

For available clock sources, please refer to the respective device specific component user manual.

RAM sections Initialization and Status Verification

This sub module provides the functionality for initializing the RAM with the any given value, at the selected blocks of the RAM and to verify the status of RAM.

MCU Reset Activation and Reason

The microcontroller reset activation will be performed by forcing a software reset. This functionality will be done by using software reset register. ECM error sources can also be configured for internal reset so that if any error occurs device will activate reset.

To provide the reset reason, this sub module captures the information available with RESF – Reset factor register. This register contains information.

Version Information

This module provides APIs for reading Module Id, Vendor Id and vendor specific version numbers.

Chapter 6 Registers Details

This section describes the register details of MCU Driver Component.

Table 6-1 Register Details

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
Mcu_Init	ECMIRCFG0	32	-	rw	MCU_IRCFG0_INIT_VALUE
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMPS	8	-	r	-
	ECMEPCTL	8	-	rw	MCU_ECM_ERROUT_TIMER, MCU_ZERO
	CVMDEW	8	Mcu_GpConfigPtr->ucCvmIndicationReg	w	-
	PROTCMDCVM	32	-	w	MCU_WRITE_DATA
	PROTSCVM	32	-	r	-
	CVMREN	32	ulCvmResetEnableReg	w	LulCntValue
	CVMDE	8	ucCvmIndicationReg	r	LucCVMCntValue | MCU_THREE
	CVMFC	8	-	w	MCU_CVM_FACTOR_CLEAR
	CVMF	8	-	r	MCU_ZERO
	RESF	32	-	r	MCU_ZERO MCU_ONE MCU_THREE
	BRAMDAT0	32	-	rw	MCU_LONG_WORD_ZERO MCU_LONG_WORD_ONE MCU_LONG_WORD_TWO
	ECMIRCFG0	32	-	rw	MCU_ECM029_MASK_VALUE

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
	SWRESA	32	-	rw	MCU_RES_CORRECT_VAL
	PROT1PHCMD	32	-	w	MCU_WRITE_DATA
	PROT1PS	32	-	r	-
	ECMPS	8	-	r	-
	ECMPE0	32	LulEcmPseudoData	w	-
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	RESFC	32	-	w	MCU_RESF_CLEAR
	ECMESSTC0	32	-	w	MCU_ECMMESSTR0_FULL_MASK
	ECMMESSTR0	32	-	r	MCU_LONG_WORD_ONE
	POF	32	-	r	MCU_POF_RST
	POFC	32	-	w	MCU_POF_CLEAR
	RESF	32	-	r	MCU_ZERO
	RESFC	32	-	w	MCU_RESF_CLEAR
	CVMF	8	-	r	MCU_THREE
	CVMFC	8	LucWriteData	w	-
	PROTCMDCVM	32	-	w	MCU_WRITE_DATA
	PROTSCVM	32	-	r	-
	ECMMESSTR0	32	Mcu_GpEcmSetting->ulEcmInternalResetReg0value	r	-
	ECMMESSTR1	32	Mcu_GpEcmSetting->ulEcmInternalResetReg1valu	r	-

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
			e		
	LVICNT	32	Mcu_GpConfigPtr->ulLVIindicationReg	rw	MCU_LVI_MASK
	LVICNT	32	Mcu_GpConfigPtr->ulLVIindicationReg	rw	MCU_LVI_MASK
	PROT1PHCMD	32	-	w	MCU_WRITE_DATA
	PROT1PS	32	-	r	-
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMCPCMD0	32	-	w	MCU_WRITE_DATA
	ECMMPCMD0	32	-	w	MCU_WRITE_DATA
	ECMMECLR	8	-	w	MCU_ONE
	ECMCECLR	8	-	w	MCU_ONE
	ECMESSTC0	8	-	w	MCU_ECM029_MASK_VALUE
	ECMMESSTR0	32	-	r	MCU_ZERO
	ECMPS	8	-	r	-
	ECMEMK0	32	-	rw	(MCU_ECMEMK0 | MCU_ECM029_MASK_VALUE)
	ECMMICFG0	32	-	rw	(MCU_ECMMICFG0 & (~MCU_ECM029_MASK_VALUE))
	ECMNMICFG0	32	-	rw	(MCU_ECMNMICFG0 & (~MCU_ECM029_MASK_VALUE))
	ECMIRCFG0	32	-	rw	(MCU_ECMIRCFG0 & (~MCU_ECM029_MASK_VALUE))

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
	TESTCOMPREG1	32	-	rw	(~MCU_LOCKSTEP_DUMMY_VALUE)
	TESTCOMPREG0	32	-	rw	MCU_LOCKSTEP_DUMMY_VALUE
	ECMMESSTR0	32	-	r	MCU_TWO
	ECMESSTC0	32	-	w	MCU_TWO
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMPS	8	-	r	-
	ECMEMK0	32	LpEcmSetting->ulEcmErrorMaskReg0Value	rw	-
	ECMEMK1	32	LpEcmSetting->ulEcmErrorMaskReg1Value	rw	-
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMPS	8	-	r	-
	ECMESSTC0	32	LulEcmPseudoData	w	-
	ECMPE0	32	LulEcmPseudoData	w	-
	ECMEMK0	32	-	rw	MCU_ECM029_MASK_VALUE
	ECMMESSTR0	32	LulEcmPseudoData	r	
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMMESSTR1	32	LulEcmPseudoData	r	-
	ECMPS	8	-	r	-
	ECMCESSTR0	32	-	r	-
	ECMPE1	32	LulEcmPseudoData	w	MCU_ERROROUT_STATUS

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
	ECMESSTC1	32	LulEcmPseudoData	w	-
	ECMMESSTR1	32	LulEcmPseudoData	r	-
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMCESSTR1	32	-	r	-
	ECMPS	8	-	r	-
	CVMF	8	-	r	MCU_ZERO
	CVMDMASK	8	-	rw	MCU_ONE
	CVMFC	8	-	w	MCU_CVM_FACTOR_CLEAR
	PROTCMDCVM	32	-	w	MCU_WRITE_DATA
	PROTSCVM	32	-	r	-
	CVMF	8	-	r	MCU_ZERO
	CVMDIAG	8	-	r/w	-
	PROTCMDCVM	32	-	w	MCU_WRITE_DATA
	PROTSCVM	32	-	r	-
	EIBD8	32	-	rw	MCU_EIBD08_CPU1_VALUE
	IMR0EIMK8	32	-	rw	MCU_ENABLE_INTERRUPT
	ECMEPCFG	8	-	rw	-
	ECMPS	8	-	r	-
	ECMPCMD1	32		w	-
	ECMMICFG0	32	LpEcmSetting->ulEcmMaskInterReg0value	rw	-
	ECMMICFG1	32	LpEcmSetting-	rw	-

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
			>ulEcmMaskInterReg1value		
	ECMNMICFG0	32	LpEcmSetting->ulEcmNonMaskInterReg0value	rw	-
	ECMNMICFG1	32	LpEcmSetting->ulEcmNonMaskInterReg1value	rw	-
	ECMIRCFG0	32	LpEcmSetting->ulEcmInternalResetReg0value	rw	-
	ECMIRCFG1	32	LpEcmSetting->ulEcmInternalResetReg1value	rw	-
	ECMEMK0	32	LpEcmSetting->ulEcmErrorMaskReg0Value	rw	-
	ECMEMK1	32	LpEcmSetting->ulEcmErrorMaskReg1Value	rw	-
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMPS	8	-	r	-
	ECMDTMCTL	8	-	w	MCU_ECM_DELY_TIMER_STOP
	ECMDTMCMP	16	-	rw	MCU_ECM_DLYTIMER_VALUE
	ECMPCMD1	32	-	w	MCU_WRITE_DATA

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
	ECMPS	8	-	r	-
	ECMDTMCFG0	32	LpEcmSetting->ulEcmDelayTimerReg0Value	rw	-
	ECMDTMCFG1	32	LpEcmSetting->ulEcmDelayTimerReg1Value	rw	-
	ECMDTMCFG2	32	LpEcmSetting->ulEcmDelayTimerReg2Value	rw	-
	ECMDTMCFG3	32	LpEcmSetting->ulEcmDelayTimerReg3Value	rw	-
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMPS	8	-	r	-
	ECMMESSTR0	32	-	r	MCU_ECMMESSTR0_FULL_MASK
	ECMESSTC0	32	-	w	MCU_ECMMESSTR0_FULL_MASK
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMMESSTR1	32	-	r	MCU_ECMMESSTR1_FULL_MASK
	ECMESSTC1	32	-	w	MCU_ECMMESSTR1_FULL_MASK
	ECMPS	8	-	r	-

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
Mcu_InitRamSection	ECMMESSTR0	32	-	r	MCU_RAM_MASK0_VALUE
	ECMESSTC0	32	-	w	MCU_RAM_MASK0_VALUE
	ECMMESSTR1	32	-	r	MCU_RAM_MASK1_VALUE
	ECMESSTC1	32	-	w	MCU_RAM_MASK1_VALUE
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMPS	8	-	r	-
	ECMMICFG0	32	LpEcmSetting->ulEcmMaskInterReg0value	rw	-
	ECMMICFG1	32	LpEcmSetting->ulEcmMaskInterReg1value	rw	-
	ECMNMICFG0	32	LpEcmSetting->ulEcmNonMaskInterReg0value	rw	-
	ECMNMICFG1	32	LpEcmSetting->ulEcmNonMaskInterReg1value	rw	-
	ECMIRCFG0	32	LpEcmSetting->ulEcmInternalResetReg0value	rw	-
	ECMIRCFG1	32	LpEcmSetting->ulEcmInternal	rw	-

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
			ResetReg1value		
	ECMEMK0	32	LpEcmSetting->ulEcmErrorMaskReg0Value	rw	-
	ECMEMK1	32	LpEcmSetting->ulEcmErrorMaskReg1Value	rw	-
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMPS	8	-	r	-
Mcu_InitClock	CLKD0DIV	32	-	rw	MCU_ZERO
	CKSC0CTL	32	-	rw	MCU_LONG_WORD_ZERO
	CSC0STAT	32	Mcu_GpClockSetting->ucExtClk0SelectedSrcClock	r	-
	CLKD1DIV	32	-	rw	MCU_ZERO
	CKSC1CTL	32	-	rw	MCU_LONG_WORD_ZERO
	CSC1STAT	32	Mcu_GpClockSetting->ucExtClk1SelectedSrcClock	r	-
	CLKD0STAT	32	-	r	MCU_CLKOUT_ACTIVE_SYNC
	CLKD1STAT	32	Mcu_GpClockSetting-	r	MCU_CLKOUT_ACTIVE_SYNC

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
			>ulExternalClk1Divider		
	PROT1PHCMD	32	-	w	MCU_WRITE_DATA
	PROT1PS	32	-	r	-
	CLKD0DIV	32	Mcu_GpClockSetting->ulExternalClk0Divider	rw	-
	CLKD0STAT	32	-	r	MCU_CLKOUT_ACTIVE_SYNC
	CLKD1DIV	32	-	rw	-
	CLKD1STAT	32	Mcu_GpClockSetting->ulExternalClk1Divider	r	MCU_CLKOUT_ACTIVE_SYNC
	ADCKSC0CTL	32	Mcu_GpClockSetting->ucAdcClkSelectCtrlRegValue	rw	-
	ADCKSC0STAT	32	(Mcu_GpClockSetting->ucAdcClkSelectCtrlRegValue) |	r	MCU_CLOCK_ACTIVE
	PROT1PHCMD	32	-	w	MCU_WRITE_DATA
	PROT1PS	32	-	r	-
	CLMA0CMPH	16	LpClmaConfigPtr->usCLMAnCMPHValue	rw	-

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
	CLMA0 CMPL	16	LpClma ConfigPtr->usCLMAnCMPLValue	rw	-
	CLMA0PCMD	8	-	w	MCU_WRITE_DATA
	CLMA0PS	8	-	r	-
	CLMA0CTL0	8	-	rw	MCU_ONE
	CLMA1CMPH	16	LpClmaConfigPtr->usCLMAnCMPHValue	rw	-
	CLMA1CMPL	16	LpClmaConfigPtr->usCLMAnCMPLValue	rw	-
	CLMA1PCMD	8	-	w	MCU_WRITE_DATA
	CLMA1PS	8	-	r	-
	CLMA1CTL0	8	-	rw	MCU_ONE
	CLMA2CMPH	16	LpClmaConfigPtr->usCLMAnCMPHValue	rw	-
	CLMA2CMPL	16	LpClmaConfigPtr->usCLMAnCMPLValue	rw	-
	CLMA2PCMD	8	-	w	MCU_WRITE_DATA
	CLMA2PS	8	-	r	-
	CLMA2CTL0	8	-	rw	MCU_ONE
	CLMA3CMPH	16	LpClmaConfigPtr->usCLMA	rw	-

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
			nCMPHValue		
	CLMA3CMPL	16	LpClmaConfigPtr->usCLMAnCMPLValue	rw	-
	CLMA3PCMD	8	-	w	MCU_WRITE_DATA
	CLMA3PS	8	-	r	-
	CLMA3CTL0	8	-	rw	MCU_ONE
	CLMATEST	32	CLMA_SELF_TEST_VALUE	rw	MCU_FOUR MCU_TWO MCU_ZERO
	CLMATESTS	32	LpClmaConfigPtr->enClmaIdx	r	
	PROT1PHCMD	32	-	w	MCU_WRITE_DATA
	PROT1PS	32	-	r	-
Mcu_DistributePllClock	-	-	-	-	-
Mcu_GetPllStatus	-	-	-	-	-
Mcu_GetResetReason	-	-	-	-	-
Mcu_GetResetRawValue	-	-	-	-	-
Mcu_PerformReset	SWRESA	32	-	rw	MCU_RES_CORRECT_VAL
	PROT1PHCMD	32	-	w	MCU_WRITE_DATA
	PROT1PS	32	-	r	-
Mcu_SetMode	-	-	-	-	-
Mcu_GetRamState	-	-	-	-	-
Mcu_EcmReleaseErrorOutPin	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMCPCMD0	32	-	w	MCU_WRITE_DATA

API Name	Registers Used	Register Access 8/16/32 bits	Config Parameter	Register Access r/w/rw	Macro/Variable
	ECMMPCMD0	32	-	w	MCU_WRITE_DATA
	ECMMECLR	8	-	w	MCU_ONE
	ECMCECLR	8	-	w	MCU_ONE
	ECMESSTC0	8	-	w	MCU_ECM029_MASK_VALUE
	ECMMESSTR0	32	-	r	MCU_ZERO
	ECMPS	8	-	r	-
	ECMMESSTR0	32	-	r	MCU_ECMMESSTR0_FULL_MASK
	ECMESSTC0	32	-	w	MCU_ECMMESSTR0_FULL_MASK
	ECMPCMD1	32	-	w	MCU_WRITE_DATA
	ECMMESSTR1	32	-	r	MCU_ECMMESSTR1_FULL_MASK
	ECMESSTC1	32	-	w	MCU_ECMMESSTR1_FULL_MASK
	ECMPS	8	-	r	-
	CVMF	8	-	r	MCU_ZERO
	CVMDMASK	8	-	rw	MCU_ONE
	CVMFC	8	-	w	MCU_CVM_FACTOR_CLEAR
	PROTCMDCVM	32	-	w	MCU_WRITE_DATA
	PROTSCVM	32	-	r	-

From above table the Register write verification is implemented for following list of registers:

- ECMMESSTR0

- ECMMESSTR1
- ECMIRCFG0
- CVMDE
- CVMF
- CVMDMASK
- CVMDIAG
- CLMAnCTL0
- CLMATEST
- ECMEMK0
- ECMEMK1
- POF
- RESFC
- ADCKSC0CTL
- CLMAnCMPH
- CLMAnCMPL
- CLMAnCTL0
- ECMDTMCTL
- ECMDTMR
- ECMEPCFG
- CKSCnCTL

Chapter 7 Interaction between the User and MCU Driver Component

The details of the services supported by the MCU Driver Component to the upper layers users and the mapping of the channels to the hardware units is provided in the following sections:

7.1. Services Provided By MCU Driver Component To User

The MCU Driver Component provides the following functions to upper layers, if supported by hardware:

- To Perform the Self diagnostic test for the ECM, CVM, Clock Monitor and Lock step.
- To initialize the RAM and to verify the status, section wise.
- To initialize the MCU specific clock options.
- To activate the specific clock to the MCU clock distribution.
- To read the reset type from the hardware.
- To perform the micro controller reset.
- To read the MCU Driver component version information.
- To clear the ERROROUT pin.

Chapter 8 MCU Driver Component Header And Source File Description

This section explains the MCU Driver Component's C Source and C Header files. These files have to be included in the project application while integrating with other modules.

The C header file generated by MCU Driver Generation Tool:

- Mcu_Cfg.h
- Mcu_Reg.h
- Mcu_Cbk.h

The C source file generated by MCU Driver Generation Tool:

- Mcu_PBcfg.c

The MCU Driver Component C header files:

- Mcu.h
- Mcu_Debug.h
- Mcu_Irq
- Mcu_PBTypes.h
- Mcu_Ram.h
- Mcu_Types.h
- Mcu_Version.h
- Mcu_RegWrite.h

The MCU Driver Component source files:

- Mcu.c
- Mcu_Irq.c
- Mcu_Ram.c
- Mcu_Version.c

The Stub C header files:

- Compiler.h
- Compiler_Cfg.h
- MemMap.h
- Platform_Types.h
- rh850_Types.h
- Dem.h
- Dem_Cfg.h
- Det.h
- SchM_Mcu.h
- Os.h
- Rte.h

The description of the MCU Driver Component files is provided in the table below:

Table 8-1 Description of the MCU Driver Component Files

File	Details
Mcu_Cfg.h	This file is generated by the MCU Driver Module Code Generation Tool for MCU Driver Module pre-compile time parameters. The macros and the parameters generated will vary with respect to the configuration in the input ARXML file.
Mcu_Reg.h	This file contains the definitions for addresses of the hardware registers used in the MCU Driver Module.
Mcu_Cbk.h	This file contains the extern declaration of call back functions used in the MCU Driver Module.
Mcu_PBcfg.c	This file contains post-build configuration data. The structures related to MCU Initialization, clock and power mode setting are provided in this file. Data structures will vary with respect to parameters configured.
Mcu.h	This file provides extern declarations for all the MCU Driver Module APIs. This file provides service Ids of APIs, DET Error codes and type definitions for MCU Driver initialization structure. This header file shall be included in other modules to use the features of MCU Driver Module.
Mcu_Irq.h	This file contains the ISR functions prototypes of the MCU Driver Module.
Mcu_Types.h	This file provides data structure and type definitions for initialization of MCU Driver.
Mcu_PBTypes.h	This file contains the macros used for the post build time parameters.
Mcu_Ram.h	This file contains the extern declarations for the global variables that are defined in Mcu_Ram.c file and the version information of the file.
Mcu_Version.h	This file contains the macros of AUTOSAR version numbers of all modules that are interfaced to MCU.
Mcu_Debug.h	This file provides Provision of global variables for debugging purpose.
Mcu_RegWrite.h	This file provides macro definitions for the registers write verify.
Mcu.c	This file contains the implementation of all MCU Driver Module APIs.
Mcu_Irq.c	This file contains the ISR functions of the MCU Driver Module.
Mcu_Ram.c	This file contains the global variables used by MCU Driver Module.
Mcu_Version.c	This file contains the code for checking version of all modules that are interfaced to MCU.
Compiler.h	Provides compiler specific (non-ANSI) keywords. All mappings of keywords, which are not standardized, and/or compiler specific are placed and organized in this compiler specific header.
Compiler_Cfg.h	This file contains the memory and pointer classes.
MemMap.h	This file allows mapping of variables, constants and code of modules to individual memory sections. Memory mapping can be modified as per ECU specific needs.
Platform_Types.h	This file provides provision for defining platform and compiler dependent types.
rh850_Types.h	This file provides macros to perform supervisor mode (SV) write enabled Register ICxxx and IMR register writing using OR/AND/Direct operation

File	Details
Dem.h	This file contains declaration of DEM event id, status and version macros.
Dem_Cfg.h	This file contains macro DEM event id numbers and version macros.
Det.h	This file contains structure for DET errors with module id, instance id, Api id and error id as elements and version macros.
Os.h	This file contains Os timer related declarations and version macros.
Rte.h	This file contains version macros.

Chapter 9 Generation Tool Guide

For more information on the MCU Driver Component Generation Tool, please refer “R20UT3721EJ0100-AUTOSAR.pdf”.

Chapter 10 Application Programming Interface

This section explains the Data types and APIs provided by the MCU Driver Component to the Upper layers.

10.1. Imported Types

This section explains the Data types imported by the MCU Driver Component and lists its dependency on other modules.

10.1.1. Standard Types

In this section all types included from the Std_Types.h are listed:

- Std_ReturnType
- Std_VersionInfoType

10.1.2. Other Module Types

In this chapter all types included from the Dem.h are listed:

- Dem_EventIdType
- Dem_EventStatusType

10.2. Type Definitions

This section explains the type definitions of MCU Driver Component according to AUTOSAR Specification.

For more type definitions refer the SWS of MCU driver as mentioned in chapter 2.

10.2.1 Mcu_ClockType

Name:	Mcu_ClockType
Type:	uint8
Range:	1 to 2
Description:	Type definition for Mcu_ClockType used by the API Mcu_InitClock.

10.2.2 Mcu_RawResetType

Name:	Mcu_RawResetType
Type:	uint32
Range:	0 to 4294967295
Description:	Type definition for Mcu_RawResetType used by the API Mcu_GetResetRawValue.

Note: Mcu_GetResetRawValue API is returning the RESF register status.

10.2.3 Mcu_RamSectionType

Name:	Mcu_RamSectionType
Type:	uint8
Range:	0 to 255
Description:	Type definition for Mcu_RamSectionType used by the API Mcu_InitRamSection.

10.2.4 Mcu_PllStatusTypes

Name:	Mcu_PllStatusType	
Type:	Enumeration	
Range:	MCU_PLL_LOCKED	PLL is locked
	MCU_PLL_UNLOCKED	PLL is unlocked.
	MCU_PLL_STATUS_UNDEFINED	PLL status is unknown
Description:	Status value returned by the API Mcu_GetPllStatus.	

Note: As per CPU manual Mcu_GetPllStatus API is not supporting the PLL clock implementation. Hence Mcu_GetPllStatus is returning always MCU_PLL_LOCKED Status.

10.2.5 Mcu_RamStateType

Following are the type definitions which are specific to R4.0 used by the MCU Driver module:

Name:	Mcu_RamStateType	
Type:	Enumeration	
Range:	MCU_RAMSTATE_INVALID	RAM State is valid.
	MCU_RAMSTATE_VALID	RAM State is invalid.
Description:	Status value returned by the API Mcu_GetRamState	

10.2.6 Mcu_ResetType

Name:	Mcu_ResetType
Type:	Enumeration
Range:	MCU_POWER_ON_CLEAR_RST
	MCU_PIN_RST
	MCU_SW_RST
	MCU_WDT_RST
	MCU_LOCK_STEP_CORE_RST
	MCU_CLMA0_UPPER_LIMIT_RST
	MCU_CLMA0_LOWER_LIMIT_RST
	MCU_CLMA2_UPPER_LIMIT_RST
	MCU_CLMA2_LOWER_LIMIT_RST
	MCU_CLMA1_UPPER_LIMIT_RST
	MCU_CLMA1_LOWER_LIMIT_RST
	MCU_CLMA3_UPPER_LIMIT_RST

	MCU_CLMA3_LOWER_LIMIT_RST
	MCU_LRAM_ECC2_ADDPTY_RST
	MCU_GRAM_ECC2_ADDPTY_RST
	MCU_CASHE_RAM_ECC2_RST
	MCU_CFLH_ECC2_ADDPTY_RST
	MCU_DATA_FLSH_ECC2_RST
	MCU_DTS_RAM_ECC2_RST
	MCU_CSIH_RAM_ECC2_RST
	MCU_CAN_RAM_ECC2_RST
	MCU_FLXR_RAM_ECC2_RST
	MCU_MODE0_RST
	MCU_MODE1_RST
	MCU_MODE2_RST
	MCU_PEGUARD_RST
	MCU_GRAM_GUARD_RST
	MCU_PBUSGUARD_RST
	MCU_SAR_ADC_PTY_RST
	MCU_DATA_PRTY_RST
	MCU_ECM_COMP_RST
	MCU_LVI_RST
	MCU_TEMP_SENSE_RST
	MCU_DMA_TRANSF_RST
	MCU_DMA_REG_PROTECT_RST
	MCU_LRAM_ECC1_PTY_RST
	MCU_GRAM_ECC1_RST
	MCU_CFLH_ECC1_RST
	MCU_DATA_FLSH_ECC1_RST
	MCU_DTS_RAM_ECC1_RST
	MCU_ALL_PERI_RAM_ECC1_RST
	MCU_BIST_ECC1_RST
	MCU_BIST_ECC2_RST
	MCU_FACI_TRANSF_RST
	MCU_ECM_DELY_OVRFLW_RST
	MCU_RESET_UNDEFINED
	MCU_RESET_UNKNOWN
Description:	Type of reset supported by the hardware

Note:

1. All RAM related ECM error sources are enabled for maskable interrupts only after Ram initialization.
2. User should configure only one ECM event for each ECM error source at a time priority level for the ECM event should be as follow:

    - Internal Reset
    - Maskable Interrupt
    - Non Maskable Interrupt

10.2.7 Mcu_ClmaIndexType

Name:	Mcu_ClmaIndexType	
Type:	Enumeration	
Range:	MCU_CLMA0	CLMA0
	MCU_CLMA1	CLMA1
	MCU_CLMA2	CLMA2
	MCU_CLMA3	CLMA3
Description:	Variable of this type is used to pass in Mcu_ClmaSelfDiagnosticTest API	

10.2.8 Mcu_ModeType

Name:	Mcu_ModeType
Type:	uint8
Range:	0 to 2
Description:	Type definition for Mcu_ModeType used by the API Mcu_SetMode.

Note: As per CPU Manual Mcu_SetMode API is not supporting for any standby mode. Hence the Mcu_ModeType parameter is unused for P1x MCU module implementation.

10.3. Function Definitions

Table 10-1 API Provided by MCU Driver Component

Sl. No	API's name
1.	Mcu_Init
2.	Mcu_InitRamSection
3.	Mcu_InitClock
4.	Mcu_DistributePllClock
5.	Mcu_GetPllStatus
6.	Mcu_GetResetReason
7.	Mcu_GetResetRawValue
8.	Mcu_PerformReset
9.	Mcu_SetMode
10.	Mcu_GetRamState
11.	Mcu_GetVersionInfo
12.	Mcu_EcmReleaseErrorOutPin

10.3.1. Mcu_Init

Name:	Mcu_Init		
Prototype:	FUNC(void, MCU_PUBLIC_CODE) Mcu_Init (P2CONST(Mcu_ConfigType, AUTOMATIC, MCU_APPL_CONST) ConfigPtr)		
Service ID:	0x00		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Mcu_ConfigType	ConfigPtr	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	void	NA	
Description:	This service performs initialization of the MCU Driver component.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.2. Mcu_InitRamSection

Name:	Mcu_InitRamSection		
Prototype:	FUNC(Std_ReturnType, MCU_PUBLIC_CODE) Mcu_InitRamSection Mcu_RamSectionType RamSection)		
Service ID:	0x01		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Mcu_RamSectionType	RamSection	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	This function initializes the RAM section as provided from the configuration structure.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.3. Mcu_InitClock

Name:	Mcu_InitClock		
Prototype:	FUNC(Std_ReturnType, MCU_PUBLIC_CODE) Mcu_InitClock (Mcu_ClockType ClockSetting)		
Service ID:	0x02		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Mcu_ClockType	ClockSetting	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	This service initializes the PLL and other MCU specific clock options.		
Configuration Dependency:	None		
Preconditions:	None		

10.3.4. Mcu_DistributePllClock

Name:	Mcu_DistributePllClock		
Prototype:	FUNC(void, MCU_PUBLIC_CODE) Mcu_DistributePllClock (void)		
Service ID:	0x03		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Void	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Std_ReturnType	E_OK, E_NOT_OK	
Description:	This service activates the PLL clock to the MCU clock distribution		
Configuration Dependency:	None		
Preconditions:	None		

10.3.5. Mcu_GetPllStatus

Name:	Mcu_GetPllStatus		
Prototype:	FUNC(void, MCU_PUBLIC_CODE) Mcu_GetPllStatus (void)		
Service ID:	0x04		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Void	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Mcu_PllStatusType	MCU_PLL_LOCKED = 0, MCU_PLL_UNLOCKED, MCU_PLL_STATUS_UNDEFINED	
Description:	This service provides the lock status of the PLL		
Configuration Dependency:	None		
Preconditions:	None		

10.3.6. Mcu_GetResetReason

Name:	Mcu_GetResetReason		
Prototype:	FUNC(Mcu_ResetType, MCU_PUBLIC_CODE) Mcu_GetResetReason (void)		
Service ID:	0x05		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Void	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Mcu_ResetType	Values are read from hardware register and mentioned in file Mcu_Types.h	
Description:	The function reads the reset type from the hardware		
Configuration Dependency:	None		
Preconditions:	None		

10.3.7. Mcu_GetResetRawValue

Name:	Mcu_GetResetRawValue		
Prototype:	FUNC(Mcu_RawResetType, MCU_PUBLIC_CODE) Mcu_GetResetRawValue (void)		
Service ID:	0x06		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Void	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Mcu_RawResetType	32-bit value from hardware register	
Description:	The service return reset type value from the hardware register		
Configuration Dependency:	None		
Preconditions:	None		

10.3.8. Mcu_PerformReset

Name:	Mcu_PerformReset		
Prototype:	FUNC (void, MCU_PUBLIC_CODE) Mcu_PerformReset (void)		
Service ID:	0x07		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Void	NA	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	None	
Description:	This service provides microcontroller reset by accessing the Software reset register		
Configuration Dependency:	None		
Preconditions:	None		

10.3.9. Mcu_SetMode

Name:	Mcu_SetMode		
Prototype:	FUNC (void, MCU_PUBLIC_CODE) Mcu_SetMode (void)		
Service ID:	0x08		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	Mcu_ModeType	McuMode	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	None	
Description:	This service activates the MCU power modes		
Configuration Dependency:	None		
Preconditions:	None		

10.3.10. Mcu_GetVersionInfo

Name:	Mcu_GetVersionInfo		
Prototype:	FUNC(void, MCU_PUBLIC_CODE) Mcu_GetVersionInfo (Std_VersionInfoType* versioninfo)		
Service ID:	0x09		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	None	NA
Parameters InOut:	None	NA	NA
Parameters out:	versioninfo	Pointer to where to store the version information of this module	NA
Return Value:	**Type**	**Possible Return Values**	
	None	None	
Description:	This service returns the version information of this module		
Configuration Dependency:	None		
Preconditions:	None		

10.3.11. Mcu_GetRamState

Name:	Mcu_GetRamState		
Prototype:	FUNC(Mcu_RamStateType, MCU_PUBLIC_CODE) Mcu_GetRamState (void)		
Service ID:	0x0A		
Sync/Async:	Synchronous		
Reentrancy:	Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	None	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	Mcu_RamStateType	MCU_RAMSTATE_VALID = 0, MCU_RAMSTATE_INVALID	
Description:	This service provides the actual status of the microcontroller RAM area		
Configuration Dependency:	None		
Preconditions:	None		

10.3.12. Mcu_EcmReleaseErrorOutPin

Name:	Mcu_EcmReleaseErrorOutPin		
Prototype:	FUNC(void, MCU_PUBLIC_CODE) Mcu_EcmReleaseErrorOutPin(void)		
Service ID:	0x0B		
Sync/Async:	Synchronous		
Reentrancy:	Non-Reentrant		
Parameters In:	**Type**	**Parameter**	**Value/Range**
	None	None	NA
Parameters InOut:	None	NA	NA
Parameters out:	None	NA	NA
Return Value:	**Type**	**Possible Return Values**	
	None	None	
Description:	This is a vendor specific API. This service releases the ERROROUT pin of P1M microcontroller.		
Configuration Dependency:	None		
Preconditions:	None		

Chapter 11 Development And Production Error

In this section the development errors that are reported by the MCU Driver Component are tabulated. The development errors will be reported only when the pre-compiler option McuDevErrorDetect is enabled in the configuration. The production code errors are not supported by MCU Driver Component.

11.1. MCU Driver Component Development Errors

The following table contains the DET errors that are reported by MCU Driver Component. These errors are reported to Development Error Tracer Module when the MCU Driver Component APIs are invoked with wrong input parameters or without initialization of the driver.

Table 11-1 DET Errors of MCU Driver Component

Sl. No.	**1**
Error Code	MCU_E_PARAM_CONFIG
Related API(s)	Mcu_Init
Source of Error	When the API service is called without module initialization.
Sl. No.	**2**
Error Code	MCU_E_PARAM_CLOCK
Related API(s)	Mcu_InitClock
Source of Error	When Clock Setting is not within the settings defined in the configuration data structure.
Sl. No.	**3**
Error Code	MCU_E_PARAM_RAMSECTION
Related API(s)	Mcu_InitRamSection
Source of Error	When RamSection is not within the sections defined in the configuration data structure.
Sl. No.	**4**
Error Code	MCU_E_PLL_NOT_LOCKED
Related API(s)	Mcu_DistributePllClock
Source of Error	When PLL is not locked.
Sl. No.	**5**
Error Code	MCU_E_UNINIT
Related API(s)	Mcu_InitRamSection, Mcu_InitClock, Mcu_DistributePllClock, Mcu_GetPllStatus, Mcu_GetResetReason, Mcu_GetResetRawValue, Mcu_PerformReset, Mcu_SetMode, Mcu_GetRamState
Source of Error	When the APIs are invoked without the initialization of the MCU Driver Component.
Sl. No.	**6**
Error Code	MCU_E_INVALID_DATABASE
Related API(s)	Mcu_Init
Source of Error	When the API is invoked with no database.
Sl. No.	**7**
Error Code	MCU_E_PARAM_MODE

Related API(s)	Mcu_SetMode
Source of Error	When the API is invoked with invalid MCU mode.

11.2. MCU Driver Component Production Errors

In this section the DEM errors identified in the MCU Driver component are listed. MCU Driver component reports these errors to DEM by invoking Dem_ReportErrorStatus API. This API is invoked, when the processing of the given API request fails.

Table 11-2 DEM Errors of MCU Driver Component

Sl. No.	**1**
Error Code	MCU_E_CLOCK_FAILURE
Related API(s)	Mcu_InitClock
Source of Error	When there is failure of the monitored clock frequency.
Sl. No.	**2**
Error Code	MCU_E_WRITE_TIMEOUT_FAILURE
Related API(s)	Mcu_PerformReset, Mcu_ProtectedWrite
Source of Error	When writing to a write-protected register fails
Sl. No.	**3**
Error Code	MCU_E_CVM_SELFDIAG_FAILURE
Related API(s)	Mcu_CvmSelfDiagnosticTest
Source of Error	When there is failure CVM self-diagnostic test.
Sl. No.	4
Error Code	MCU_E_CLM_SELFDIAG_FAILURE
Related API(s)	Mcu_ClmaSelfDiagnosticTest
Source of Error	When there is failure CLMA self-diagnostic test.
Sl. No.	5
Error Code	MCU_E_ECM_SELFDIAG_FAILURE
Related API(s)	Mcu_EcmSelfDiagnosticTest
Source of Error	When there is failure ECM self-diagnostic test.
Sl. No.	**6**
Error Code	MCU_E_LOCKSTEP_SELFDIAG_FAILURE
Related API(s)	Mcu_LockStepSelfDiagnosticTest
Source of Error	When there is failure Lock step self-diagnostic test.
Sl. No.	**7**
Error Code	MCU_E_RESETCONTROLLER_STARTUP_FAILURE
Related API(s)	Mcu_StartUPTest
Source of Error	When there is failure in startup test.
Sl. No.	**8**
Error Code	MCU_E_INT_INCONSISTENT
Related API(s)	MCU_ECM_EIC_ISR
Source of Error	When there is failure in interrupt consistency check.
Sl. No.	**9**
Error Code	MCU_E_REG_WRITE_VERIFY

Related API(s)	All APIs
Source of Error	When there is a failure in Register write.

Chapter 12 Memory Organization

Following picture depicts a typical memory organization, which must be met for proper functioning of MCU Driver Component software.

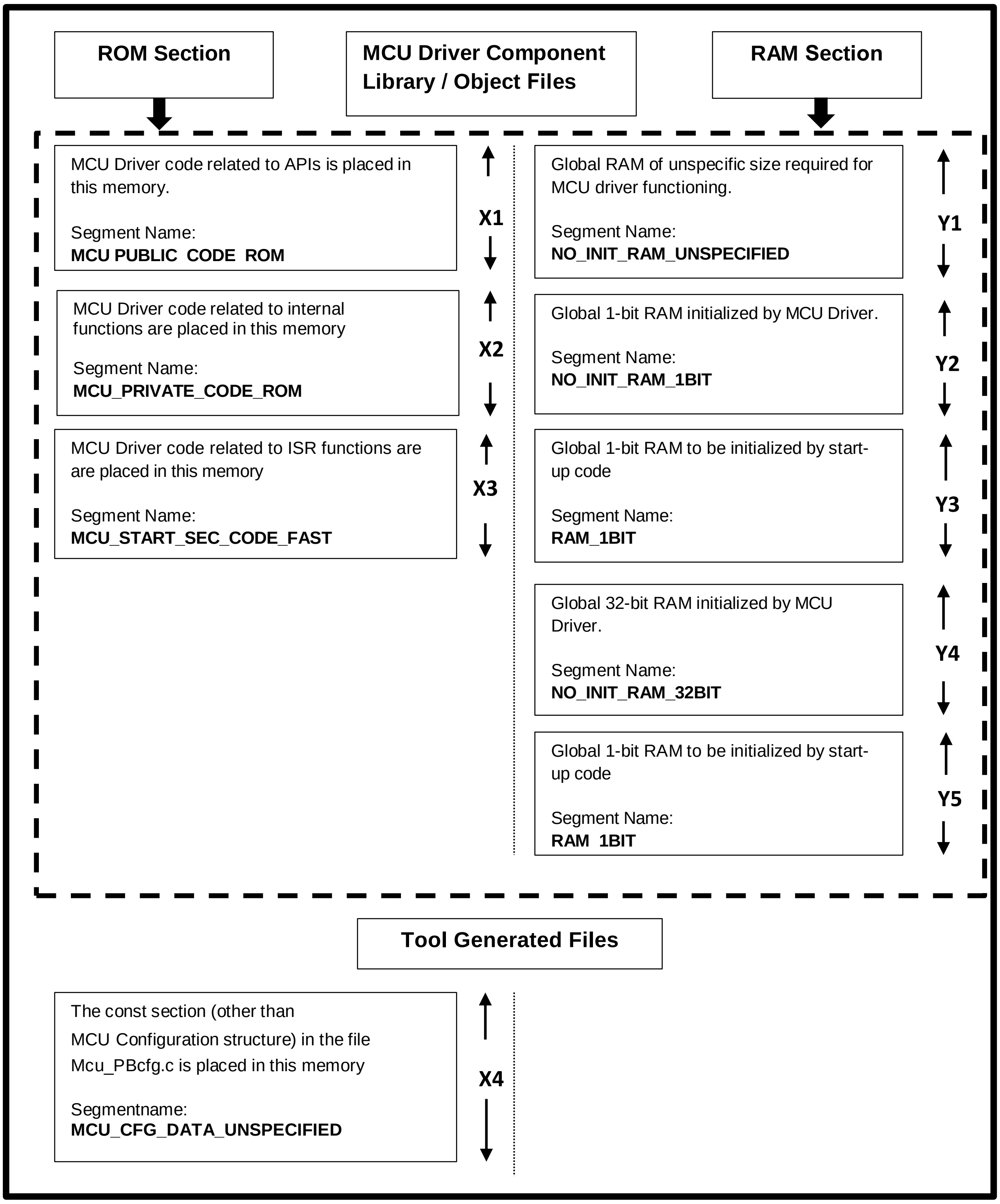
ROM Section
MCU Driver Component Library / Object Files
RAM Section
MCU Driver code related to APIs is placed in this memory.
Segment Name:
MCU PUBLIC CODE ROM
X1
MCU Driver code related to internal functions are placed in this memory
Segment Name:
MCU_PRIVATE_CODE_ROM
X2
MCU Driver code related to ISR functions are are placed in this memory
Segment Name:
MCU_START_SEC_CODE_FAST
X3
Global RAM of unspecific size required for MCU driver functioning.
Segment Name:
NO_INIT_RAM_UNSPECIFIED
Y1
Global 1-bit RAM initialized by MCU Driver.
Segment Name:
NO_INIT_RAM_1BIT
Y2
Global 1-bit RAM to be initialized by start-up code
Segment Name:
RAM_1BIT
Y3
Global 32-bit RAM initialized by MCU Driver.
Segment Name:
NO_INIT_RAM_32BIT
Y4
Global 1-bit RAM to be initialized by start-up code
Segment Name:
RAM 1BIT
Y5
Tool Generated Files
The const section (other than MCU Configuration structure) in the file Mcu_PBcfg.c is placed in this memory
Segmentname:
MCU_CFG_DATA_UNSPECIFIED
X4

Figure 12-1 MCU Driver Component Memory Organization

ROM Section (X1, X2, X3 and X4):

MCU_PUBLIC_CODE_ROM (X1): API(s) of MCU Driver Component, which can be located in code memory.

MCU_PRIVATE_CODE_ROM (X2): Internal functions of MCU Driver Component code that can be located in code memory.

MCU_START_SEC_CODE_FAST (X3): Interrupt functions of MCU Driver Component code that can be located in code memory.

MCU_CFG_DATA_UNSPECIFIED (X4): This section consists of MCU Driver Component constant configuration structures. This can be located in code memory.

RAM Section (Y1, Y2, Y3, Y4 and Y5):

NO_INIT_RAM_UNSPECIFIED (Y1): This section consists of the global RAM variables that are used internally by MCU Driver Component. This can be located in data memory.

NO_INIT_RAM_1BIT (Y2): This section consists of the global RAM variables of 1-bit size that are used internally by MCU Driver Component. This can be located in data memory.

RAM_1BIT (Y3): This section consists of the global RAM variables of 1-bit size that are initialized by start-up code and used internally by MCU Driver Component. This can be located in data memory.

NO_INIT_RAM_32BIT (Y4): This section consists of the global RAM variables of 32-bit size that are used internally by MCU Driver Component. This can be located in data memory.

RAM_32BIT (Y5): This section consists of the global RAM variables of 32-bit size that are initialized by start-up code and used internally by MCU Driver Component. This can be located in data memory.

Remark

- X1, X2, Y1, Y2 , Y3 ,Y4 and Y5 pertain to only MCU Driver Component and do not include memory occupied by Mcu_PBcfg.c file generated by MCU Driver Component Generation Tool.
- User must ensure that none of the memory areas overlap with each other. Even ‘debug’ information should not overlap.

Chapter 13 P1M Specific Information

P1M supports following devices:

- R7F701304
- R7F701305
- R7F701310
- R7F701311
- R7F701312
- R7F701313
- R7F701314
- R7F701315
- R7F701318
- R7F701319
- R7F701320
- R7F701321
- R7F701322
- R7F701323

13.1. Interaction between the User and MCU Driver Component

The details of the services supported by the MCU Driver Component to the upper layers users and the mapping of the channels to the hardware units is provided in the following sections:

13.1.1. Translation Header File

The P1x_translation.h translation header file supports following devices:

- R7F701304
- R7F701305
- R7F701310
- R7F701311
- R7F701312
- R7F701313
- R7F701314
- R7F701315
- R7F701318
- R7F701319
- R7F701320
- R7F701321
- R7F701322
- R7F701323

13.1.2. ISR Function

The table below provides the list of handler addresses corresponding to the hardware unit ISR(s) in MCU Driver Component. The user should configure the ISR functions mentioned below:

Table 13-1 ISR For MCU

Interrupt Source	Name of the ISR Function
INTECM	MCU_FEINT_ISR
	MCU_ECM_EIC_ISR

Note: The functions with "INTERRUPT "as pilot tag, provides an indication to the compiler that the function following this tag is an interrupt function type. The tag name can vary according to the compiler. User should take care of the tag name with respect to compiler used.

13.1.2.1 Interrupt routines for OS

Module's <Module>_Irq.c/h files must include "Os.h" header file to obtain the interrupt category information configured in the OS. Therefore preprocessor definitions shown by below table must be expected to be published in Os.h file by the OS in case of CAT2 or to be used in the interrupt vector table in case of CAT1. In case of CAT2 ISRs the "ISR (Isr_Name)" Keyword must be used in <Module>_Irq.c/h file.

Interrupt Category	Naming Convention
CAT1	<MCAL_INTERRUPT_NAME>_ ISR
CAT2	<MCAL_INTERRUPT_NAME>_CAT2_ISR
CAT2 (In case the handles of the OsIsr container are generated without 'Os_' prefix by Os generation tool)	Os_<MCAL_INTERRUPT_NAME>_CAT2_ISR

Example of module_irq.h:

```
/* Defines the CAT2 interrupt mapping */

#if defined (Os_<MCAL_INTERRUPT_NAME>_CAT2_ISR) || defined
(<MCAL_INTERRUPT_NAME>_CAT2_ISR)

 /* Use ISR() macro from Os.h */

 /* Defines the CAT1 interrupt mapping */

#else

extern FUNC(type, memclass) <MCAL_INTERRUPT_NAME>_ ISR(void);

#endif
```

Example of module_irq.c:

```
/* Defines the CAT2 interrupt mapping */

#if defined  (Os_<MCAL_INTERRUPT_NAME>_CAT2_ISR) || defined
(<MCAL_INTERRUPT_NAME>_CAT2_ISR)

 ISR(<MCAL_INTERRUPT_NAME>_CAT2_ISR)
```

```
/* Defines the CAT1 interrupt mapping */

#else

FUNC(type, memclass) <MCAL_INTERRUPT_NAME>_ ISR(void)

#endif
```

Note: In case if the MCAL modules are to be used standalone without having standard Autosar Os module, the user has to prepare an Os.h stub file with the published handles only for those interrupt names which are to be used as CAT2.

13.1.3. Parameter Definition File

Parameter definition files support information for P1M

Table 13-2 PDF information for P1M

PDF Files	Devices supported
R403_MCU_P1M_04_05.arxml	701304, 701305
R403_MCU_P1M_10_to_15_18_to_23.arxml	701310, 701311, 701312, 701313, 701314, 701315, 7013018, 701319, 701320, 701321, 701322, 7013023

13.2. Sample Application

13.2.1 Sample Application Structure

The Sample Application is provided as reference to the user to understand the method in which the MCU APIs can be invoked from the application.

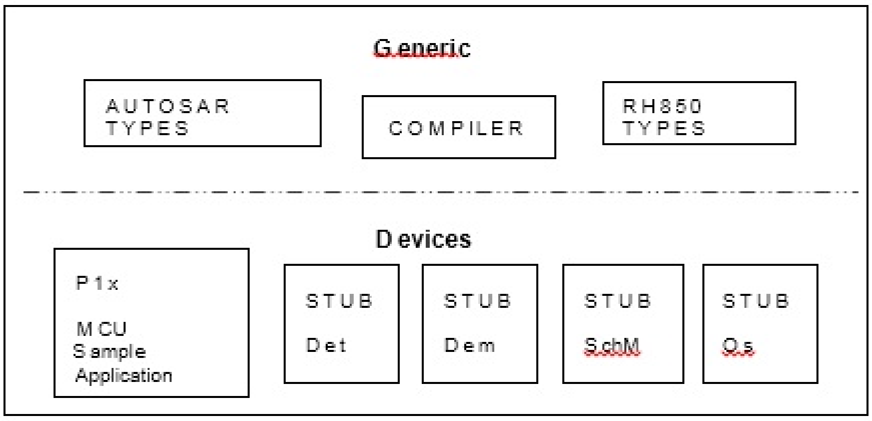

Figure 13-1 Overview of MCU Driver Sample Application

The Sample Application of the P1M is available in the path:

X1X\P1x\modules\mcu\sample_application

The Sample Application consists of the following folder structure:

```
X1X\P1x\modules\mcu\definition\<AUTOSAR_version>\<SubVariant>
\R403_MCU_P1M_04_05.arxml
\R403_MCU_P1M_10_to_15_18_to_23.arxml
X1X\P1x\modules\mcu\sample_application
        \<SubVariant>\<AUTOSAR_version>
                                             \src\Mcu_PBcfg.c
                                             \include\Mcu_Cfg.h
                                             \include\Mcu_Cbk.h
                                             \include\Mcu_Reg.h

              \config\App_MCU_P1M_701304_Sample.arxml

              \config\App_MCU_P1M_701305_Sample.arxml

              \config\App_MCU_P1M_701310_Sample.arxml

              \config\App_MCU_P1M_701311_Sample.arxml

              \config\App_MCU_P1M_701312_Sample.arxml

              \config\App_MCU_P1M_701313_Sample.arxml

              \config\App_MCU_P1M_701314_Sample.arxml

              \config\App_MCU_P1M_701315_Sample.arxml

              \config\App_MCU_P1M_701318_Sample.arxml

              \config\App_MCU_P1M_701319_Sample.arxml

              \config\App_MCU_P1M_701320_Sample.arxml

              \config\App_MCU_P1M_701321_Sample.arxml

              \config\App_MCU_P1M_701322_Sample.arxml

              \config\App_MCU_P1M_701323_Sample.arxml
```

In the Sample Application all the MCU APIs are invoked in the following sequence:

- The API Mcu_Init is invoked with a valid database address for the proper initialization of the MCU Driver, all the MCU Driver control registers and RAM variables will get initialized after this API is called.
- The API Mcu_InitRamSection is invoked to initialize the RAM section wise as provided from the configuration structure.
- The API Mcu_InitClock is invoked to initialize the clock sources Main Osc, High Speed Internal ring Oscillator.
- The API Mcu_GetPllStatus is invoked to provide the lock status of the PLL. This API will return the PLLstatus as MCU_PLL_LOCKED or

MCU_PLL_UNLOCKED.

- The API Mcu_GetResetReason is invoked to read the reset type from the hardware by checking the RESF register and if not supported, returns MCU_POWER_ON_RESET. This API shall clear the reset factor register.
- The API Mcu_GetResetRawValue is invoked to return reset type value from the hardware register RESF
- The API Mcu_GetVersionInfo is invoked to get the version of the MCU Driver module with a variable of Std_VersionInfoType, after the call of this API the passed parameter will get updated with the MCU Driver version details.
- The API Mcu_PerformReset invoked to microcontroller reset is performed by accessing the software reset register.
- The API Mcu_SetMode is invoked to activate the MCU power modes.

Remark To unmask all resets ‘target pinmask k’ command is used.

13.2.2 Building Sample Application

13.2.2.1. Configuration Example

This section contains the typical configuration which is used for measuring RAM/ROM consumption, stack depth and throughput details.

Configuration Details:
App_MCU_<SubVariant>_<Device_Number>_Sample.arxml

For P1M the <Device Number> indicates the device to be compiled, which can be 701304, 701305, 701310, 701311, 701312, 701313, 701314, 701315, 701318, 701319, 701320, 701321, 701322, 701323

13.2.2.2. Debugging the Sample Application

Remark GNU Make utility version 3.81 or above must be installed and available in the path as defined by the environment user variable “GNUMAKE” to complete the build process using the delivered sample files.

Open a Command window and change the current working directory to “make” directory present as mentioned in below path:

“X1X\P1x\common_family\make\<compiler>”

Now execute the batch file SampleApp.bat with following parameters:

```
SampleApp.bat mcu <AUTOSAR_version> <Device_name>
```

- After this, the tool output files will be generated with the configuration as mentioned in App_MCU_P1M_701310_Sample.arxml file available in the path:

  “X1X\P1x\modules\mcu\sample_application\<SubVariant>\<AUTOSAR_version>\config\App_MCU_P1M_701310_Sample.arxml”

- After this, all the object files, map file and the executable file App_MCU_P1M_Sample.out will be available in the output folder: ("X1X\P1x\modules\mcu\sample_application\<SubVariant> \obj\<compiler>")

- The executable can be loaded into the debugger and the sample application can be executed

Remark Executable files with '*.out' extension can be downloaded into the target hardware with the help of Green Hills debugger.

- If any configuration changes (only post-build) are made to the ECU Configuration Description files "X1X\P1x\modules\mcu\sample_application\<SubVariant>\<AUTOSAR_version>\config\App_MCU_P1M_701310_Sample.arxml"

- The database alone can be generated by using the following commands.

  make –f App_MCU_P1M_Sample.mak generate_mcu_config

  make –f App_MCU_P1M_Sample.mak App_MCU_P1M_Sample.s37

- After this, a flash able Motorola S-Record file App_MCU_P1M_Sample.s37 is available in the output folder.

Note: The <Device_name> indicates the device to be compiled, which can be 701304, 701305, 701310, 701311, 701312, 701313, 701314, 701315, 701318, 701319, 701320, 701321, 701322, 701323.

13.3. Memory and Throughput

13.3.1. ROM/RAM Usage

The details of memory usage for the typical configuration, with DET disabled as provided in Section 13.2.2.1 *Configuration Example* are provided in this section.

Table 13-3 ROM/RAM Details without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes in GHS
1.	ROM	MCU_PUBLIC_CODE_ROM	428
		MCU_PRIVATE_CODE_ROM	5518
		MCU_FAST_CODE_ROM	648
		MCU_CFG_DATA_UNSPECIFIED	584
		ROM.RAM_1BIT	3
		ROM.RAM_32BIT	8
2.	RAM	NO_INIT_RAM_UNSPECIFIED	28
		RAM_1BIT	3
		RAM_32BIT	8

The details of memory usage for the typical configuration, with DET enabled and all other configurations as provided in 13.2.2.1 *Configuration Example* are provided in this section.

Table 13-4 ROM/RAM Details with DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes in GHS
1.	ROM	MCU_PUBLIC_CODE_ROM	768
		MCU_PRIVATE_CODE_ROM	5518
		MCU_FAST_CODE_ROM	648
		MCU_CFG_DATA_UNSPECIFIED	584
		ROM.RAM_1BIT	3
		ROM.RAM_32BIT	8
2.	RAM	NO_INIT_RAM_UNSPECIFIED	28
		RAM_1BIT	3
		RAM_32BIT	8

13.3.2. Stack Depth

The worst-case stack depth for MCU Driver Component for the typical configuration provided in Section 13.2.2.1 Configuration Example is approximately 36 bytes.

13.3.3. Throughput Details

The throughput details of the APIs for the configuration mentioned in the Section 13.2.2.1 *Configuration Example* are listed here. The clock frequency used to measure the throughput is 160MHz for all APIs.

Table 13-5 Throughput Details of the APIs

Sl. No.	API Name	Throughput in microseconds in GHS	Remarks
1.	Mcu_Init	674.150	-
2.	Mcu_InitClock	266.650	-
3.	Mcu_GetPllStatus	0.87	-
4.	Mcu_InitRamSection	10.462	-
5.	Mcu_GetResetReason	0.125	-
6.	Mcu_GetResetRawValue	0.125	-
7.	Mcu_GetVersionInfo	0.150	-
8.	Mcu_GetRamState	0.512	-
10.	Mcu_SetMode	0.87	-
11.	Mcu_EcmReleaseErrorOutPin	5.987	
12.	Mcu_PerformReset	0.150	-
13.	Mcu_ProtectedWriteRetNone	3.487	-

Chapter 14 Release Details

MCU Driver Software

Version: 1.0.7

Revision History

Sl. No.	Description	Version	Date
1.	Initial Version	1.0.0	18-Oct-2013
2.	Following changes are made: 1. Chapter 2 is updated for reference documents. 2. Section 4.3 is updated for Exclusive Area name change. 3. Section 4.4 is updated for adding user and supervisor mode details for new APIs. 4. Chapter 6 is updated for Register details used in APIs. 5. Chapter 8 is updated for adding file description for tool generated file 'Mcu_Cbk.h'. 6. Section 10.3 is updated for adding new APIs in function definition. 7. Section 11.1 is updated for adding DET error. 8. Chapter 13 and section 13.1.1 is updated for adding new supported devices. 9. Section 13.2 is updated for change in compiler and linker version details. 10. Section 13.3 is updated for adding latest configuration details for supported devices. 11. Section 13.3.2 is updated for change in configuration example for sample application testing. 12. Section 13.4 is updated for updating ROM/RAM, Stack and Throughput details. 13. Chapter 14 is updated for increment in MCU Driver software version.	1.0.1	30-Apr-2014
3.	Following changes are made: 1. Chapter 4 is updated for adding a new section regarding RAM Initialization. 2. Chapter 13 and section 13.3.1 is updated for removal of unsupported devices.	1.0.2	09-May-2014
4.	Following changes are made: 1. Chapter 2 is updated for referenced documents version. 2. Section 13.1.1 is updated for adding the device names. 3. Section 13.2 is updated for compiler, assembler and linker details. 4. Section 13.3 is updated to add parameter definition file and sample application configuration files for all P1M devices. 5. Chapter 14 is updated for MCU driver component version information. 6. Deviation list is updated to add MCU_E_PARAM_POINTER error for Mcu_GetVersioInfo API and AUTOSAR requirement.	1.0.3	20-Oct-2014
5	Following changes are made: 1. Chapters 7 to 11 updated to start at odd page. 2. Date in Revision History updated. 3. Document page numbers are corrected. 4. Chapter 11 updated to add MCU_E_PARAM_MODE DET. 5. Page number is removed from publication info page. 6. Section 13.2 is updated for compiler, assembler and linker details. 7. Chapter 8 is update to include rh850_types.h file.	1.0.4	12-Dec-2014

Sl. No.	Description	Version	Date
6	Following changes are made: 1. New section, “4.7 Callout API” added to chapter 4. 2. Information regarding Interrupt vector table is added to “Section 4.1 General”. 3. As part of device support activity for R7F701304, R7F701305, R7F701313, R7F701315, R7F701318 to R7F701323 updated sections 3.1.1, 13.1, 13.2. 4. Removed section Compiler,Linker and Assembler in Chapter13. 5. Updated section 13.3 for memory and throughput 6. Copyright information has been changed to 2015	1.0.5	30-Apr-2015
7	Following changes are made: 1. Added details of stubs in section 3.1.1. 2. Section 4.1 General is updated. 3. Chapter 6 Register Details is updated by adding the missing registers and APIs. 4. Section 7.1 is updated for new functionality of clearing error out pin. 5. Alignment of Headings and Tables are corrected in Section 10.2 Type definition. 6. Section10.3 is updated for new API and 11.2 for new production errors. 7. Alignment of bullet points of remarks are corrected in Chapter 12 Memory Organization. 8. Alignment of the text "P1x Mcu Sample Application" is corrected in Figure 13-1 Overview of MCU Driver Sample Application. 9. Section 13.1.2.1 is added for OS interrupt routines. 10. Section 13.3 Memory and throughput updated. 11. Clock frequency used to measure the throughput is updated in Section 13.3.3 Throughput Details. 12. Exclusive area for register protection renamed to MCU_REGISTER_PROTECTION. Also added exclusive area VARIABLE_PROTECTION. 13. Chapter14 is updated.	1.0.6	08-Jan-2016
8	Following changes are made: 1. Chapter 6 Register details are updated. 2. Section 4.3 Data consistency is updated. 3. Section 13.4 is updated for ROM/RAM, Stack and Throughput details 4. Section 6.1 General is updated for the description of Mcu_InitRamSection and McuRamSectorSetting. 5. R-number and Version are updated.	1.0.7	25-Mar-2016

Sl. No.	Description	Version	Date
9	Following changes are made: 1. Chapter 4 section 4.1 General 5 points added. 2. Section 4.4 a Note is added below the table 'Supervisor mode and User mode details'. 3. Table 4-1 MCU Driver Protected Resources List added in section 4.3 Data consistency. 4. VARIABLE_PROTECTION changed to MCU_VARIABLE_PROTECTION in section 4.3 Data consistency. 5. Chapter 6 Register details are updated. 6. Section 3.1.1 Folder structure, chapter 8 C header file section, Table 8-1 updated by adding Mcu_RegWrite.h. 7. DEM error for register write verify added in Table 11-2 DEM Errors of MCU Driver Component. 8. Chapter 12 Memory Organization updated to follow initialization policy. 9. Section 13.2 reference to .one and .html files are removed. 10. Note added in section 13.2.1 ISR function. 11. Added new section 13.3.4 Critical section details. 12. R Number updated. 13. Section 13.3 updated with throughput values. 14. Chapter 4 forethoughts section updated by adding points for STATIC tag and for multiple invoke of Mcu_Init multiple times. 15. Chapter 14 release details updated.	1.0.8	24-Oct-2016

AUTOSAR MCAL R4.0.3 User's Manual
MCU Driver Component Ver.1.0.8
Embedded User’s Manual

Publication Date: Rev.1.00, October 24, 2016

Published by: Renesas Electronics Corporation

SALES OFFICES

Renesas Electronics Corporation

http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

AUTOSAR MCAL R4.0.3

User’s Manual

Renesas Electronics Corporation

R20UT3720EJ0100