User's Manual

AUTOSAR MCAL R4.0.3

User's Manual

MCU Driver Component Ver.1.0.2
Embedded User's Manual

Target Device:
RH850\P1x-C

Renesas Electronics
www.renesas.com

Rev.1.00 Jan 2017

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.
5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (space and undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.
6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures by your own responsibility as warranty for your products/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stockpiling, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.
10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Abbreviations and Acronyms

Abbreviation / Acronym	Description
ADC	Analog to Digital Converter
ANSI	American National Standards Institute
API	Application Programming Interface
ATOM	ARU-connected Timer Output Module
AUTOSAR	AUTomotive Open System ARchitecture
CAN	Control Area Network
CLMA	Clock Monitor
CMU	Clock Management Unit
CVM	Core Voltage Monitor
DEM/Dem	Diagnostic Event Manager
DET/Det	Development Error Tracer
DIO	Digital Input Output
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
GTM	Generic Timer Module
ICU	Input Capture Unit
ID/Id	IDentifier
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
OS/Os	Operating System
PWM	Pulse Width Modulation
PLL	Phase Locked Loop
RAM/Ram	Random Access Memory
ROM	Read Only Memory
RESF	Reset Factor Register
RTE	Run Time Environment
SPI	Serial Peripheral Interface
SW	SoftWare
TIM	Timer Input Module
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-C 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:

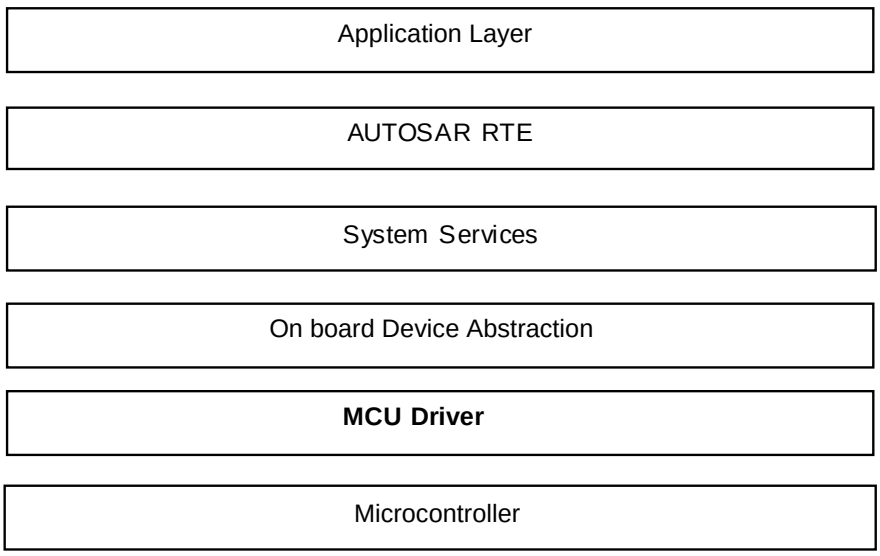

Figure 1-1 System Overview Of AUTOSAR Architecture

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:

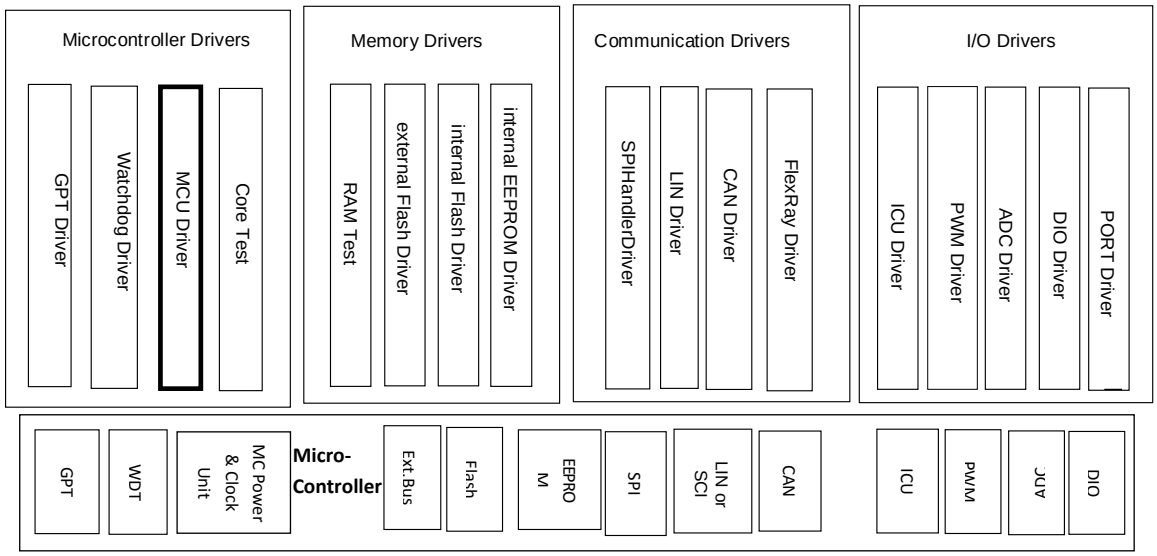

Figure 1-2 System Overview Of The MCU Driver In AUTOSAR MCAL Layer

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 (P1x-C Specific Information)	This section provides P1x-C 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.	Specification of MCU Driver (AUTOSAR_SWS_MCUDriver.pdf)	3.2.0
2.	RH850/P1x-C Group Document User's Manual: Hardware (r01uh0517ej0100_rh850p1x-c_Open)	1.00
3.	Specification of Memory Mapping (AUTOSAR_SWS_MemoryMapping.pdf)	1.4.0
4.	Specification of Platform Types (AUTOSAR_SWS_PlatformTypes.pdf)	2.5.0
5.	AUTOSAR BSW Makefile Interface (AUTOSAR_BSW_MakefileInterface.pdf)	0.3
6.	Specification of Compiler Abstraction (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 "R20UT3652EJ0100-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-C\modules\mcu\src
\Mcu.c
\Mcu_Ram.c
\Mcu_Irq.c
\Mcu_Version.c

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

X1X\P1x-C\modules\mcu\sample_application\<SubVariant>\make\ghs
\App_MCU_P1x-C_Sample.mak

X1X\P1x-C\modules\mcu\sample_application\<SubVariant>\make\ghs
\App_MCU_P1x-C_Sample.ld

X1X\P1x-C\modules\mcu\sample_application\<SubVariant>\obj

X1X\P1x-C\modules\mcu\generator
\R403_MCU_P1x-C_BSWMDT.arxml.

X1X\P1x-C\modules\mcu\user_manual

(User manuals will be available in this folder)

Note: 1. <AUTOSAR_version> should be 4.0.3.
2. <SubVariant> can be P1H-C or P1H-CE or P1M-C.

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.
- 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 state feature is enabled the API Mcu_InitRamSection follows this procedure:
  - Initializes all configured RAM sections according to user configuration.
  - Enables ECM interrupt generation for all configured RAM errors according to user configuration.

  The procedure requires that the complete RAM is initialized before the RAM state functionality is used.
- 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 achive the same functionality. These containers have multiplicity 1 - 1 and have fixed values depends on the reset type.
- The parameter 'McuLoopCount' represents the number of register write retries in MCU module. User has to take care to provide a proper value for this parameter to avoid stabilization issues. The default value used for this parameter is 28, to avoid unwanted reporting of DEM due to stabilization issues.
- Support for CLMA4 is available only for P1H-C (Dual core) devices.

4.2. Preconditions

Following preconditions have to be adhered by the user, for proper

functioning of the MCU Driver Component:

- The Mcu_Cfg.h file 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.
- The user shall configure the exact Module Short Name Mcu in configurations as specified in config.xml file and the same shall be given in command line.

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 following exclusive area along with scheduler services is used to provide data integrity for shared resources:
MCU_REGISTER_PROTECTION
MCU_PWR_MODE_PSC_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'.

If the 'McuCriticalSectionProtection' parameter is enabled then the critical section protection is applicable to all these API's in MCU Module:

Table 4-1 Critical Section Details

API Name	Exclusive Area Type	Protected Resources
Mcu_Init	MCU_REGISTER_PROTECTION	Registers: ECMmnESSTC0 DTMCTL
Mcu_InitRamSection	MCU_REGISTER_PROTECTION	Registers: ECMmnESSTC0 ECMmnESSTC1
Mcu_GetRamState	VARIABLE_PROTECTION	Shared Data: Global variable to store Ram state of MCU Driver
Mcu_SetMode	MCU_PWR_MODE_PSC_PROTECTION	Registers: MSR_LM3,MSR_LM4, MSR_LM5,MSR_LM6, MSR_LM7,MSR_LM8, MSR_LM10,MSR_L11, MSR_LM12

The highest measured duration of a critical section was 0.587 micro seconds measured for **Mcu_Init** API with a CPU frequency of 160 MHz.

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	Critical section protection cannot be enabled
2	Mcu_InitClock	x	x	-
3	Mcu_DistributePllClock	x	x	-

4	Mcu_GetPllStatus	x	x	-
5	Mcu_InitRamSection	-	x	Critical section protection cannot be enabled
6	Mcu_GetResetRawValue	x	x	-
7	Mcu_GetVersionInfo	x	x	-
8	Mcu_GetRamState	-	x	Critical section protection cannot be enabled
9	Mcu_SetMode	-	x	1.The execution of the assembly instruction for entering HALT mode will not be possible 2. Critical section protection cannot be enabled
10	Mcu_PerformReset	x	x	-

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 Lists

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. Register Write Verify

Register write-verify is a functional safety based implementation, where the control registers' write operation is verified straight away after the write operation. After writing to control registers, content of the registers are read back and verified against the expected content to make sure that register content has been written correctly.

The main use of this implementation is to detect random HW faults (transient/permanent). This can happen on the bus while writing to the configuration registers which will potentially lead to wrong configuration and potentially wrong operation. Also it could happen because of faulty registers which will potentially lead to incorrect operation.

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:

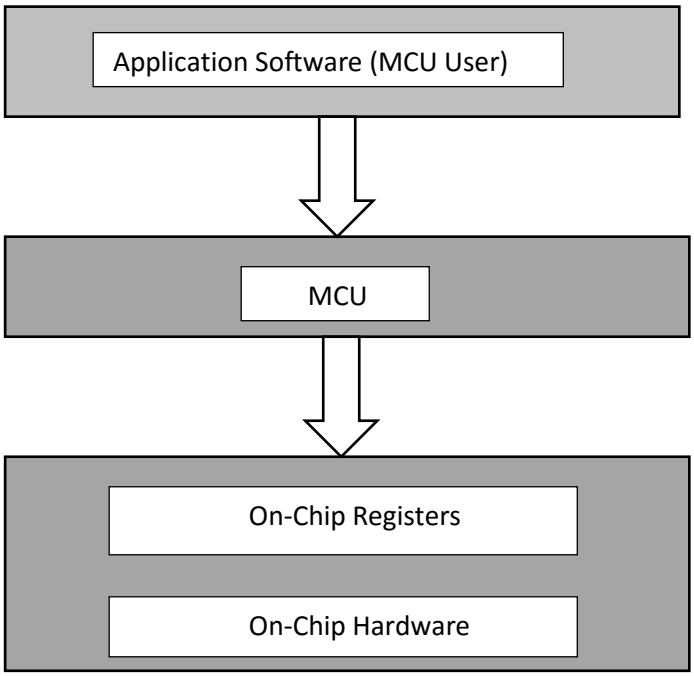

Figure 5-1 MCU Driver Architecture

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 Core Voltage Monitoring, Clock monitoring and Lock Step mechanism is possible in real scenario.
- 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 Core Voltage Monitoring, Clock monitoring and Lock Step mechanism 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.

Generic Timer Module

P1x-C controller uses GTM HW core for timer related drivers.

The Clock Management Unit is responsible for clock generation of the counters and of the GTM-IP. The CMU generate different clock sources for the whole GTM-IP.

The Configurable Clock Generation subunit provides eight dedicated clock sources for the GTM submodules: TIM and ATOM.

All the CMU clock initializations required for TIM and ATOM sub modules are done from the MCU module. The GTM CMU registers are provided in Chapter 6.

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 watchdog overflow. The limitation of this implementation is that this type of reset activation is possible only if the watchdog is configured in reset mode. If microcontroller reset is requested when the watchdog is configured in NMI mode, then an interrupt is generated which would not be handled in this driver component.

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

HW BIST is executed by Power-On-Reset, System Reset 1 and SystemReset2. In System Reset 2, HW BIST execution can be disabled depending on Field BIST control register (BSEQ0CTL).

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	Config Parameter	Macro/Variable
Mcu_Init	RESC	McuEcmRstConfigure	-
	ECMnEMK0	-	MCU_ECMEMK0_FULL_MASK
	ECMnEMK1	-	MCU_ECMEMK1_FULL_MASK
	ECMnEMK2	-	MCU_ECMEMK2_FULL_MASK
	ECMnPS	-	-
	ECMnPCMD1	-	-
	CVMDEW	McuCvmOutMaskFbist, McuCvmOutMaskDiag, McuCvmResetEnable	-
	ECMnEPCFG	McuEcmErrorOutputMode	MCU_ECM_ERROUT_MODE
	ECMnMICFG0	McuEcmErrorMaskableInterrupt	-
	ECMnMICFG1	McuEcmErrorMaskableInterrupt	-
	ECMnMICFG2	McuEcmErrorMaskableInterrupt	-
	ECMnNMICFG0	McuEcmErrorNonMaskableInterrupt	-
	ECMnNMICFG1	McuEcmErrorNonMaskableInterrupt	-
	ECMnNMICFG2	McuEcmErrorNonMaskableInterrupt	-
	ECMnIRCFG0	McuEcmErrorInternalReset	-
	ECMnIRCFG1	McuEcmErrorInternalReset	-
	ECMnIRCFG2	McuEcmErrorInternalReset	-
	ECMnDTMCTL	-	MCU_ECM_DELAY_TIMER_STOP
	ECMnDTMCMP	-	MCU_ECM_DLYTIMER_VALUE
	ECMnDTMCFG0	McuEcmErrorMIDelayTimer	-

API Name	Registers	Config Parameter	Macro/Variable
	ECMnDTMCFG1	McuEcmErrorMIDelayTimer	-
	ECMnDTMCFG2	McuEcmErrorMIDelayTimer	-
	ECMnDTMCFG3	McuEcmErrorNMIDelayTimer	-
	ECMnDTMCFG4	McuEcmErrorNMIDelayTimer	-
	ECMnDTMCFG5	McuEcmErrorNMIDelayTimer	-
	GTM0CMUCLKEN	-	MCU_CMUCLK_DISABLE
	GTM0CMUGCLKNUM	-	MCU_ZERO
	GTM0CMUGCLKDEN	-	MCU_ZERO
	GTM0CMUCLK0CTRL	-	MCU_ZERO
	GTM0CMUCLK1CTRL	-	MCU_ZERO
	GTM0CMUCLK2CTRL	-	MCU_ZERO
	GTM0CMUCLK3CTRL	-	MCU_ZERO
	GTM0CMUCLK4CTRL	-	MCU_ZERO
	GTM0CMUCLK5CTRL	-	MCU_ZERO
	GTM0CMUCLK6CTRL	-	MCU_ZERO
	GTM0CMUCLK7CTRL	-	MCU_ZERO
	MSR _LM5	-	MCU_ZERO
	RESF	McuEcmRstConfigure	-
	RESFC	-	-
	CVMFC	McuClma0SelfDiagnosticTest, McuClma1SelfDiagnosticTest, McuClma2SelfDiagnosticTest, McuClma3SelfDiagnosticTest, McuClma4SelfDiagnosticTest	-
	CVMF	McuClma0SelfDiagnosticTest, McuClma1SelfDiagnosticTest, McuClma2SelfDiagnosticTest, McuClma3SelfDiagnosticTest, McuClma4SelfDiagnosticTest	-
	CVMDMASK	McuCvmOutMaskDiag	-
	CVMDIAG	McuClma0SelfDiagnosticTest, McuClma1SelfDiagnosticTest, McuClma2SelfDiagnosti	-

API Name	Registers	Config Parameter	Macro/Variable
	CVMMON	McuCvmOutMaskDiag	-
	CMPTST0	McuLockStepSelfDiagnosticTest	MCU_LOCKSTEP_DUMMY_VALUE
	CMPTST1	McuLockStepSelfDiagnosticTest	MCU_LOCKSTEP_DUMMY_VALUE
	ECMMnESSTR0	-	-
	ECMnESSTC0	-	-
	ECM0ESSTC1	-	-
	ECM0ESSTC2	-	-
	ECM0PS	-	-
	ECMMESSTR0	-	-
	ECMMESSTR1	-	-
	ECMMESSTR2	-	-
	ECMCESSTR0	-	-
	ECMCESSTR1	-	-
	ECMnPEM	-	-
	ECM0PCMD1	-	-
	ECMCESSTR2	-	-
	CVMDE	McuCvmDiagLockBit	-
	ECMnPE0	-	-
	ECMPCMD1	-	-
	ECMPE0	-	-
	ECMPS	-	-
	ECMESSTC0	-	-
Mcu_InitRamSection	ECMnMICFG0	McuEcmErrorMaskableInterrupt	-
	ECMnMICFG1	McuEcmErrorMaskableInterrupt	-
	ECMnNMICFG0	McuEcmErrorNonMaskableInterrupt	-
	ECMnNMICFG1	McuEcmErrorNonMaskableInterrupt	-
	ECMnIRCFG0	McuEcmErrorNonMaskableInterrupt	-
	ECMnIRCFG1	McuEcmErrorNonMaskableInterrupt	-
	ECMnEMK0	McuEcmErrorNonMaskableInterrupt	-
	ECMnPS	-	-
	ECMnPCMD1	-	-
	ECMnESSTC0	-	-
	ECMnESSTC1	-	-
	ECMnESSTR0	-	-
	ECMnESSTR1	-	-
	ECMnEMK1	-	-

API Name	Registers	Config Parameter	Macro/Variable
Mcu_InitClock	CKSC0C	-	ucSysClk0SelectedSrcCloc k LucClkSrcClk
	CKSC0S	-	-
	CLKD0STAT	-	-
	CLKD0DIV	-	usSysClk0Divider LusClkDivider
	CLKD1STAT	-	-
	CLKD1DIV	-	usSysClk1Divider LusClkDivider
	CKSC2C	-	ucExtClk0SelectedSrcCloc k LucClkSrcClk
	CKSC2S	-	-
	CLKD2STAT	-	-
	CLKD2DIV	-	usExtClk0Divider LusClkDivider
	CKSC3C	-	ucExtClk1SelectedSrcCloc k LucClkSrcClk
	CKSC3S	-	-
	CLKD3STAT	-	-
	CLKD3DIV	-	usExtClk1Divider LusClkDivider
	CLMA0CMPH	McuClm0MonitoringCloc kAccuracy, McuClm0SamplingClock Accuracy	-
	CLMA0CMPL	McuClm0MonitoringCloc kAccuracy, McuClm0SamplingClock Accuracy	-
	CLMA0CTL0	-	MCU_ONE
	CLMA0PCMD	-	-
	CLMA0PS	-	-
	CLMA1CMPH	McuClm1MonitoringCloc kAccuracy, McuClm1SamplingClock Accuracy	-
	CLMA1CMPL	McuClm1MonitoringCloc kAccuracy, McuClm1SamplingClock Accuracy	-
	CLMA1CTL0	-	MCU_ONE
	CLMA1PS	-	-
	CLMA1PCMD	-	-
	CLMA2CMPH	McuClm2MonitoringCloc kAccuracy, McuClm2SamplingClock Accuracy	-
	CLMA2CMPL	McuClm2MonitoringCloc kAccuracy, McuClm2SamplingClock Accuracy	-
	CLMA2CTL0	-	MCU_ONE
	CLMA2PCMD	-	-

API Name	Registers	Config Parameter	Macro/Variable
	CLMA2PS	-	-
	CLMA3CMPH	McuClm3MonitoringClockAccuracy, McuClm0SamplingClockAccuracy	-
	CLMA3CMPL	McuClm3MonitoringClockAccuracy, McuClm0SamplingClockAccuracy	-
	CLMA3CTL0	-	MCU_ONE
	CLMA3PCMD	-	-
	CLMA3PS	-	-
	CLMA4CMPH	McuClm3MonitoringClockAccuracy, McuClm0SamplingClockAccuracy	CLMA4CMPH
	CLMA4CMPL	McuClm3MonitoringClockAccuracy, McuClm0SamplingClockAccuracy	CLMA4CMPL
	CLMA4CTL0	-	CLMA4CTL0
	CLMA4PCMD	-	CLMA4PCMD
	CLMA4PS	-	CLMA4PS
	CLMATESTS	-	-
	CLMATEST	-	-
	GTM0CMUGCLKNUM	McuGTMCMUGCLKNumerator	-
	GTM0CMUGCLKDEN	McuGTMCMUGCLKDenominator	-
	GTM0CMUCLKxCTRL	McuGTMChannelClkSrcDivider	-
	GTM0CMUCLKEN	-	MCU_CMUCLK_ENABLE
	GTM0GTMIRQMODE		MCU_ZERO
Mcu_DistributePllClock	-	-	-
Mcu_GetPllStatus	-	-	-
Mcu_GetResetReason	-	-	-
Mcu_GetResetRawValue	-	-	-
Mcu_PerformReset	SWSRESA0	-	MCU_ONE
	SWARESA0	-	MCU_ONE
	MSR_LM3	McuMcanStopTrigger, McuMcanWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER

API Name	Registers	Config Parameter	Macro/Variable
Mcu_SetMode	MSR_LM4	McuFlexrayStopTrigger, McuFlexrayWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	MSR_LM5	McuGtmStopTrigger, McuGtmWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	MSR_LM6	McuEthernetStopTrigger ,McuEthernetWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	MSR_LM7	McuRsentStopTrigger, McuRsentWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	MSR _LM8	McuHsUsrtStopTrigger, McuHsUsrtWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	MSR _LM10	McuCsihStopTrigger, McuCsihWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	MSR _LM11	McuRlin3StopTrigger, McuRlin3WakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	MSR _LM12	McuAdcStopTrigger, McuAdcWakeupTrigger	MCU_TARGET_STOP_TRIGGER, MCU_TARGET_WAKEUP_TRIGGER
	SWLRESS3	-	-
	SWLRESS4	-	-
	SWLRESS5	-	-
	SWLRESS6	-	-
	SWLRESS7	-	-
	SWLRESS8	-	-
	SWLRESS10	-	-
	SWLRESS11	-	-
	SWLRESS12	-	-

API Name	Registers	Config Parameter	Macro/Variable
	EIC0, EIC1, EIC2, EIC3, EIC8, EIC9, EIC32, EIC33, EIC34, EIC35, EIC36, EIC38, EIC39, EIC41, EIC42, EIC53, EIC54, EIC61, EIC62, EIC83, EIC87, EIC91, EIC111, EIC114, EIC128, EIC129, EIC130, EIC131, EIC132, EIC141, EIC142, EIC174, EIC177, EIC184, EIC186, EIC197, EIC209, EIC211, EIC240, EIC241, EIC242, EIC243, EIC244, EIC245	-	-
	IMR0	McuWakeUpFactorName	-
	IMR1	McuWakeUpFactorName	-
	IMR2	McuWakeUpFactorName	-
	IMR3	McuWakeUpFactorName	-
	IMR4	McuWakeUpFactorName	-
	IMR5	McuWakeUpFactorName	-
	IMR6	McuWakeUpFactorName	-
	IMR7	McuWakeUpFactorName	-
Mcu_ResetReasonStore	RESF	-	-
	RESFC	-	MCU_RESF_CLEAR
	ECMMESSTR0	-	-
	ECMCESSTR0	-	-
	ECM0ESSTC0	-	-
	ECM0PCMD1	-	-
	ECM0PS	-	-
	ECMMESSTR1	-	-
	ECMCESSTR1	-	-
	ECM0ESSTC1	-	-
	ECMMESSTR2	-	-
	ECMCESSTR2	-	-
	ECM0ESSTC2	-	-

API Name	Registers	Config Parameter	Macro/Variable
Mcu_WakeupConfigure	EIC0, EIC1, EIC2, EIC3, EIC8, EIC9, EIC32, EIC33, EIC34, EIC35, EIC36, EIC38, EIC39, EIC41, EIC42, EIC53, EIC54, EIC61, EIC62, EIC83, EIC87, EIC91, EIC111, EIC114, EIC128, EIC129, EIC130, EIC131, EIC132, EIC141, EIC142, EIC174, EIC177, EIC184, EIC186, EIC197, EIC209, EIC211, EIC240, EIC241, EIC242, EIC243, EIC244, EIC245	-	MCU_WAKEUP_INTP_MASK
	IMR0	McuWakeUpFactorName	-
	IMR1	McuWakeUpFactorName	-
	IMR2	McuWakeUpFactorName	-
	IMR3	McuWakeUpFactorName	-
	IMR4	McuWakeUpFactorName	
	IMR5	McuWakeUpFactorName	-
	IMR6	McuWakeUpFactorName	-
	IMR7	McuWakeUpFactorName	-
Mcu_GetRamState	-	-	-

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 Initialize the ECM, EVM, 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.

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_Hardware.h
- Mcu_Cbk.h

The C source file generated by MCU Driver Generation Tool:

- Mcu_PBcfg.c
- Mcu_Hardware.c

The MCU Driver Component C header files:

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

The MCU Driver Component source files:

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

The Stub C header files:

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

The Stub C source files:

- Dem.c
- Det.c
- Os.c
- SchM_Mcu.c

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 Generation Tool for various MCU Driver Component pre-compile time parameters. The macros and the parameters generated will vary with respect to the configuration in the input ARXML file.
Mcu_Hardware.h	This file contains the #define macros for the hardware registers to be used by the driver.
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_Hardware.c	This file contains the reference objects for the hardware register structure which is defined in device header file.
Mcu.h	This file provides extern declarations for all the MCU Driver Component 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 Component.
Mcu_Types.h	This file provides data structure and type definitions for initialization of MCU Driver.
Mcu_Irq.h	This file contains the extern declaration of ISR routines.
Mcu_Debug.h	This file provides Provision of global variables for debugging purpose.
Mcu_PBTypes.h	This file contains the data structure definitions of clock setting and Mode setting.
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_RegWrite.h	This file contains macro for register write verify check
Mcu.c	This file contains the implementation of all APIs.
Mcu_Ram.c	This file contains the global variables used by MCU Driver Component.
Mcu_Irq.c	This file contains the definition of ISR routines
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 to map 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 contains platform dependent types declaration.
Os.h	This file contains macro definitions of OS component.
Std_Types.h	This file contains macro definitions of Standard Types.
Rte.h	This file contains macro definitions of RTE component.
SchM_Mcu.h	This file contains the external declaration of scheduler services of MCU module.
Dem.h	This file contains the external declaration of DEM Error Status function
Dem_Cfg.h	This file contains macro definitions of DemEventParameters.

Det.h	This file contains the external declaration of DET Report Error function and structure definition of DET Error.

Det.c	This file contains the definition of DET Report Error function and structure definition of DET Error.
Dem.c	This file contains the definition of DEM Error Status function
Os.c	This file is a stub for OS component and contains the definition of the OS category interrupts subroutines.
SchM_Mcu.c	This file is a stub for SchM Component and contains the definition of the exclusive areas for the scheduler services, which are used to provide data integrity for shared resources.

Chapter 9 Generation Tool Guide

For more information on the MCU Driver Code Generation Tool, please refer “R20UT3652EJ0100-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_types.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:	0 to 255
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_ModeType

Name:	Mcu_ModeType
Type:	uint8
Range:	0 to 255
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-C MCU module implementation.

10.2.4. Mcu_RamSectionType

Name:	Mcu_RamSectionType
Type:	Uint32
Range:	0 to 4294967295
Description:	Type definition for Mcu_RamSectionType used by the API Mcu_InitRamSection.

10.2.5. 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 does not support the PLL clock implementation. Hence Mcu_GetPllStatus always returns MCU_PLL_LOCKED Status.

10.2.6. 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	

10.2.7. Mcu_ResetType

Name:	Mcu_ResetType
Type:	Enumeration
Range:	MCU_POWER_ON_RESET
	MCU_TERMINAL_RESET
	MCU_CVM_RESET
	MCU_SW_SYS_RESET
	MCU_WATCHDOG_RESET
	MCU_LOCK_STEP_CORE_RST
	MCU_PBUS_FSS_RST
	MCU_BUS_BRIDGE_ERROR_RST
	MCU_SAFETY_MECH_COMP_RST
	MCU_TEMPERATURE_SENSOR_RST
	MCU_CLMA0_RST

	MCU_CLMA2_RST
	MCU_CLMA3_RST
	MCU_CLMA5_RST
	MCU_CLMA1_RST
	MCU_LRAM_ECC_DED_RST
	MCU_GRAM_ECC_DED_RST
	MCU_CACHE_RAM_EDC_RST
	MCU_CODE_FLS_ECC_DED_RST
	MCU_DATA_FLS_ECC_DED_RST
	MCU_CSIH_RAM_ECC_DED_RST
	MCU_CAN_RAM_ECC_DED_RST
	MCU_ETH_RAM_ECC_DED_RST
	MCU_FR_RAM_ECC_DED_RST
	MCU_GTM_RAM_ECC_DED_RST
	MCU_BUS_ECC_DED_RST
	MCU_BUS_ECC_SED_RST
	MCU_LRAM_ADDR_OVF_RST
	MCU_GRAM_ADDR_OVF_RST
	MCU_CODE_FLS_ADDR_OVF_RST
	MCU_DATA_FLS_ADDR_OVF_RST
	MCU_PERI_RAM_ECC_ADDR_OVF_RST
	MCU_DTS_RAM_ECC_DED_RST
	MCU_DTS_RAM_ECC_SED_RST
	MCU_LRAM_ECC_SED_RST
	MCU_GRAM_ECC_SED_RST
	MCU_CODE_FLS_ECC_SED_RST
	MCU_DATA_FLS_ECC_SED_RST
	MCU_CSIH_RAM_ECC_SED_RST
	MCU_CAN_RAM_ECC_SED_RST
	MCU_ETH_RAM_ECC_SED_RST
	MCU_FR_RAM_ECC_SED_RST
	MCU_GTM_RAM_ECC_SED_RST
	MCU_PE_GUARD_RST
	MCU_GRAM_GUARD_RST
	MCU_MEMC_GUARD_RST
	MCU_SLAVE_GUARD_RST
	MCU_CODE_FLS_PE_UNMAP_ACCESS_RST
	MCU_GRAM_PE_UNMAP_ACCESS_RST
	MCU_LPB_PE_UNMAP_ACCESS_RST
	MCU_PBUS_UNMAP_ACCESS_RST
	MCU_HBUS_UNMAP_ACCESS_RST
	MCU_CODE_FLS_GVCI_UNMAP_ACCESS_RST
	MCU_GRAM_FLS_GVCI_UNMAP_ACCESS_RST
	MCU_RES_HBUS_UNMAP_ACCESS_RST
	MCU_DMA_TRANSFER_RST

	MCU_DMA_UNMAPPED_RST
	MCU_FLS_SEQUENCE_RST
	MCU_FLS_FACI_RST
	MCU_ADC_PARITY_RST
	MCU_PE_UNINTEN_EN_DIS_RST
	MCU_UNINTEN_DEACT_USR_RST
	MCU_UNINTEN_ACT_CFP_MODE_RST
	MCU_UNINTEN_DEBUG_EN_DET_RST
	MCU_UNINTEN_ACT_TESTMODE_RST
	MCU_ECM_COMP_RST
	MCU_DEBUGGER_RESET
	MCU_SW_APPL_RESET
	MCU_BIST_RESET
	MCU_RESET_UNDEFINED
	MCU_RESET_UNKNOWN
Description:	Type of reset supported by the hardware

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_GetVersionInfo
9.	Mcu_PerformReset
10.	Mcu_SetMode
11	Mcu_GetRamState

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(Mcu_PllStatusType, 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 (Mcu_ModeType McuMode)		
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 (P2VAR(Std_VersionInfoType, AUTOMATIC, MCU_APPL_CONST) 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_INVALID = 0, MCU_RAMSTATE_VALID	
Description:	This service provides the actual status of the microcontroller RAM area		
Configuration Dependency:	None		
Preconditions:	None		

Chapter 11 Development And Production Errors

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 Mcu_Init is called with NULL PTR.
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_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.	**5**
Error Code	MCU_E_PARAM_POINTER
Related API(s)	Mcu_GetVersionInfo
Source of Error	When Mcu_GetVersionInfo is called with NULL PTR.
Sl. No.	**6**
Error Code	MCU_E_PARAM_MODE
Related API(s)	Mcu_SetMode
Source of Error	When McuMode is not within the settings defined in the configuration data structure.
Sl. No.	**7**
Error Code	MCU_E_INVALID_DATABASE
Related API(s)	Mcu_Init
Source of Error	When the API is invoked with no database.

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_ProtectedWrite
Source of Error	When writing to a write-protected register fails
Sl. No.	**3**
Error Code	MCU_E_POWER_DOWN_MODE_FAILURE
Related API(s)	Mcu_SetMode
Source of Error	When there is failure in low power mode transition.
Sl. No.	**4**
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.	**5**
Error Code	MCU_E_REG_WRITE_VERIFY
Related API(s)	Mcu_Init, Mcu_InitRamSection, Mcu_InitClock, Mcu_SetMode, MCU_ECM_EIC_ISR,
Source of Error	When there is a failure in Register write.
Sl. No.	**6**
Error Code	MCU_E_CLM_SELFDIAG_FAILURE
Related API(s)	Mcu_InitClock
Source of Error	When there is failure in Clock Monitor Self Diagnosis
Sl. No.	**7**
Error Code	MCU_E_CVM_SELFDIAG_FAILURE
Related API(s)	Mcu_Init
Source of Error	When there is failure in CVM Self Diagnosis
Sl. No.	**8**
Error Code	MCU_E_ECM_SELFDIAG_FAILURE
Related API(s)	Mcu_Init
Source of Error	When there is failure in ECM Self Diagnosis
Sl. No.	**9**
Error Code	MCU_E_LOCKSTEP_SELFDIAG_FAILURE
Related API(s)	Mcu_Init
Source of Error	When there is failure in Lockstep Self Diagnosis

Chapter 12 Memory Organization

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

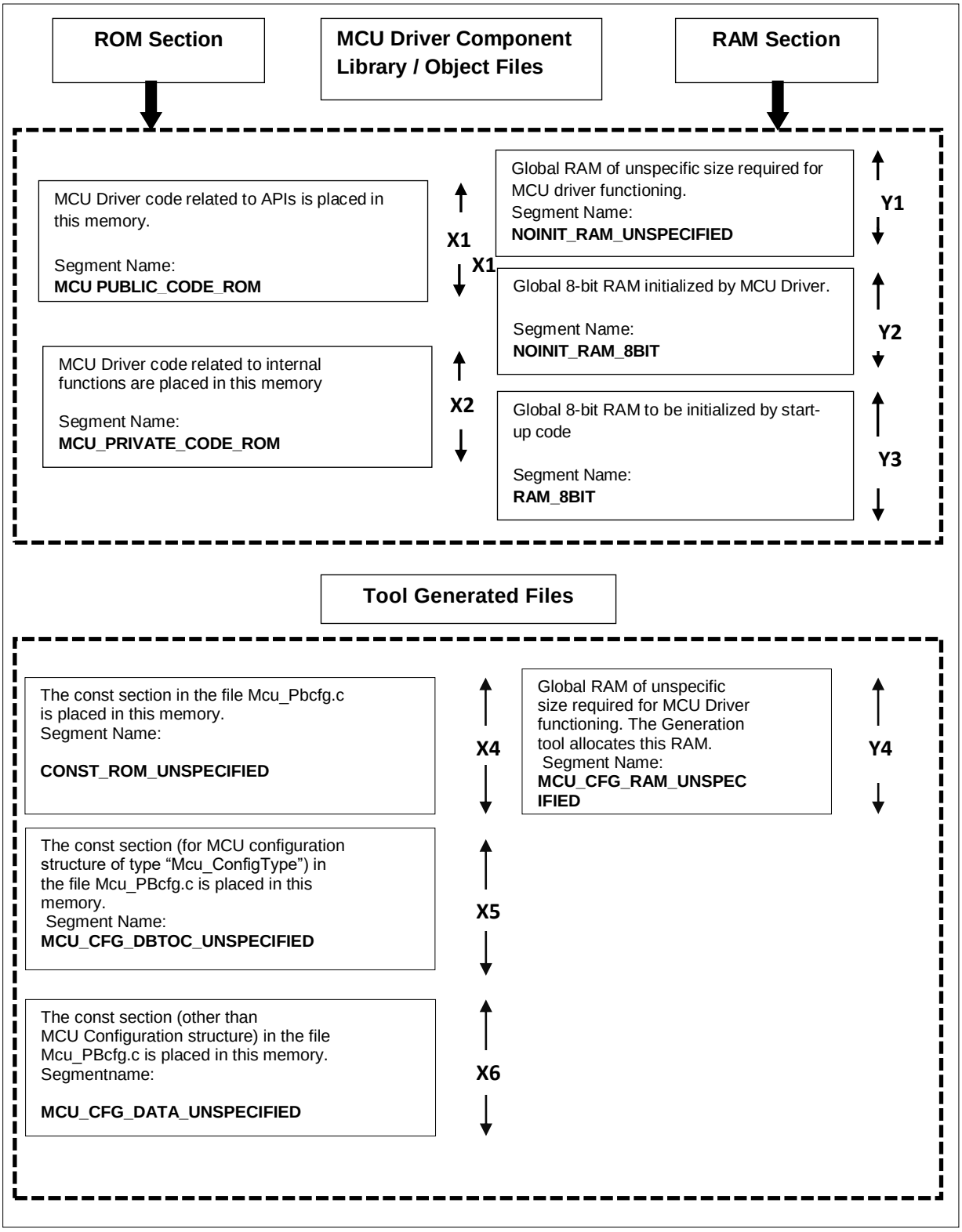

Figure 12-1 MCU Driver Component Memory Organization

ROM Section (X1, X2, X3, X4 and X5):

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_CFG_DBTOC_UNSPECIFIED (X4): This section consists of MCU Driver Component database table of contents generated by the MCU Driver Component Generation Tool. This can be located in code memory.

MCU_CFG_DATA_UNSPECIFIED (X5): This section consists of MCU Driver Component constant configuration structures. This can be located in code memory.

CONST_ROM_UNSPECIFIED (X6): This section consists of MCU Driver Component constant structures used for function pointers in MCU Driver Component. This can be located in code memory.

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

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

NOINIT_RAM_8BIT (Y2): This section consists of the global RAM variables of 8-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.

MCU_CFG_RAM_UNSPECIFIED (Y4): This section consists of the global RAM variables that are generated by MCU Driver Component Generation Tool. This can be located in data memory.

Remark

- X1, X2, Y1, Y2 and Y3 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 P1x-C Specific Information

P1x-C supports following devices:

- R7F701370A(CPU1(PE1)), R7F701371(CPU1(PE1)), R7F701372(CPU1(PE1)), R7F701373, R7F701374

13.1. 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

13.1.1. 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

_INTERRUPT_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.

13.2. 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.

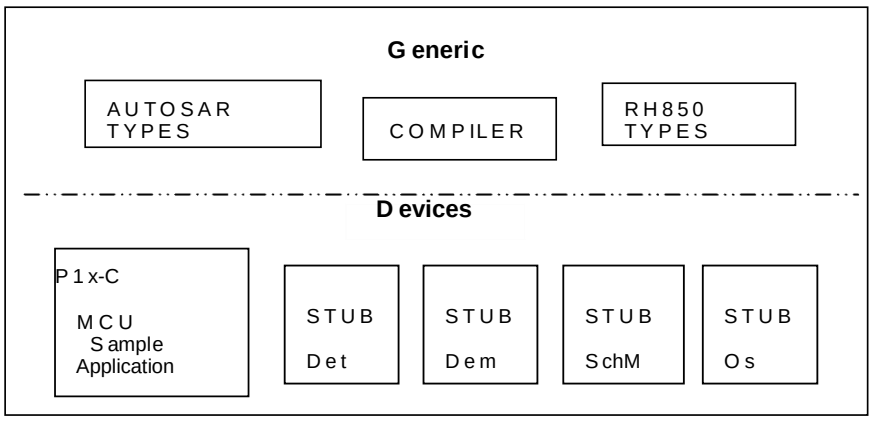

Figure 13-1 Overview of MCU Driver Sample Application

The Sample Application of the P1x-C is available in the path

X1X\P1x-C\modules\mcu\sample_application

The Sample Application consists of the following folder structure:

X1X\P1x-C\modules\mcu\definition\<AUTOSAR_version>\common
R403_MCU_P1X-C.arxml

X1X\P1x-C\modules\mcu\sample_application\< SubVariant>

\<AUTOSAR_version>
\src\Mcu_PBcfg.c
\src\Mcu_Hardware.c
\include\Mcu_Cfg.h
\include\Mcu_Hardware.h

\config\ App_MCU_P1x-C_701370A_Sample.arxml
\config\App_MCU_P1x-C_701370A_Sample.html
\config\App_MCU_P1x-C_701370A_Sample.one

\config\ App_MCU_P1x-C_701371_Sample.arxml
\config\ App_MCU_P1x-C_701371_Sample.html
\config\ App_MCU_P1x-C_701371_Sample.one

\config\ App_MCU_P1x-C_701372_Sample.arxml
\config\ App_MCU_P1x-C_701372_Sample.html
\config\ App_MCU_P1x-C_701372_Sample.one

\config\ App_MCU_P1x-C_701373_Sample.arxml
\config\ App_MCU_P1x-C_701373_Sample.html
\config\ App_MCU_P1x-C_701373_Sample.one

\config\ App_MCU_P1x-C_701374_Sample.arxml
\config\ App_MCU_P1x-C_701374_Sample.html
\config\ App_MCU_P1x-C_701374_Sample.one

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.
- The API Mcu_GetPllStatus is invoked to provide the lock status of the PLL. This API will return the PLL status 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 is invoked to reset the microcontroller 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 ’ 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.

- For Autosar Version R4.0.3

Configuration Details:

App_MCU_<SubVariant>_<Device_Name>_Sample.html

Note For P1x-C <Device_name> can be 701370A, 701371, 701372, 701373, 701374.

13.2.2.2 Debugging The Sample Application

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-C\common_family\Sample_Application\<Complier>”

Now execute batch file SampleApp.bat with following parameters:

SampleApp.bat Mcu <Device_name>

Note For P1x-C <Device_name> can be 701370A, 701371, 701372, 701373, 701374.

After this, the tool output files will be generated with the configuration as mentioned in the path:

- For Autosar Version R4.0.3

“X1X\P1x-C\modules\Mcu\sample_application\<SubVariant>\<

AUTOSAR_version> \config”

- After this, all the object files, map file and the executable file App_MCU_P1x-C_Sample.out will be available in the output folder (“X1X\P1x-C\modules\Mcu\sample_application\<SubVariant>\obj\<complier>” in this case).

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

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 file.

"X1X\P1x-C\modules\Mcu\sample_application\<SubVariant>\<Autosar_version>\config\App_MCU_<SubVariant>_<Device_name>_Sample.arxml" the database alone can be generated by using the following commands

make –f App_MCU_<SubVariant>_Sample.mak generate_Mcu_config

make –f App_MCU_<SubVariant>_Sample.mak App_MCU_<SubVariant>_Sample.out

- After this, a flash able Motorola S-Record file App_MCU_<SubVariant>_Sample.run is available in the output folder.

Note
1.For P1x-C <Device_name> can be 701370A, 701371,701372, 701373, 701374.
2. <compiler> for example can be "ghs".
3. <SubVariant> can be P1H-C, P1H-CE, P1M-C.
4. <AUTOSAR_version> can be 4.0.3.

13.3. Memory and Throughput

Typical Configuration

- DET ON
- All other Pre-Compile Settings ON
- RAM Sector Configuration0
  - Default Value 0xFF
  - RAM Section Base Address 0xFEDE0000
  - RAM Section Size 0x40

13.3.1. ROM/RAM Usage

The details of memory usage for the typical configuration, with DET disabled are provided in this section.

Table 13-2 ROM/RAM Details without DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes in GHS
1.	ROM	DEFAULT_CODE_ROM	11394
		CONST_ROM_UNSPECIFIED	316
		CONST_ROM_32BIT	48
2.	RAM	RAM_1BIT	1
		RAM_8BIT	1
		RAM_16BIT	4
		RAM_32BIT	4
		RAM_UNSPECIFIED	24

The details of memory usage for the typical configuration, with DET enabled are provided in this section.

Table 13-3 ROM/RAM Details with DET

Sl. No.	ROM/RAM	Segment Name	Size in bytes in GHS
1.	ROM	DEFAULT_CODE_ROM	9790
		CONST_ROM_UNSPECIFIED	316
		CONST_ROM_32BIT	48
2.	RAM	RAM_1BIT	2
		RAM_8BIT	1
		RAM_16BIT	4
		RAM_32BIT	4
		RAM_UNSPECIFIED	24

13.3.2. Stack Depth

The worst-case stack depth for MCU Driver Component for the typical configuration is 148 bytes.

13.3.3. Throughput Details

The throughput details of the APIs at 160 MHz clock frequency are mentioned below.

Table 13-4 Throughput Details of the APIs

Sl. No.	API Name	Throughput in microseconds in GHS	Remarks
1.	Mcu_Init	84.425	-
2.	Mcu_InitRamSection	14.125	-
3.	Mcu_InitClock	117.562	-
4.	Mcu_DistributePllClock	0.87	-
5.	Mcu_GetPllStatus	0.87	-
6.	Mcu_GetResetReason	0.100	-
7.	Mcu_GetResetRawValue	0.87	-
8.	Mcu_GetVersionInfo	0.137	-
9.	Mcu_GetRamstate	0.662	-
10.	Mcu_PerformReset	0. 150	_
11.	Mcu_EcmReleaseErrorOutPin	8.900	_

Chapter 14 Release Details

MCU Driver Software

Version: 1.1.0

Revision History

Sl. No.	Description	Version	Date
1.	Initial Version	1.0.0	14-Aug-2015
2.	Following changes are made 1. Chapter 2 “Reference Documents” is updated. 2. Chapter 3 and Chapter 9 is updated for the name of the Tool User Manual. 3. Chapter 4 “Forethoughts” is updated. 4. Section 4.3 is updated for adding the information on Critical Section Protection. 5. Chapter 5 is updated for the information on GTM and the HW BIST. 6. Section 10.3 “Function Definitions” are updated. 7. Chapter 6 “Register Details” is updated. 8. Section 13.2 “ISR Function” is added. 9. Section 13.4 “Memory and Throughput” is updated. 10. Chapter 14 “Release Details” is updated. 11. Added R number for the document.	1.0.1	15-Apr-2016
3.	Following changes are made 1. Removed Section 13.1 “Compiler Linker and Assembler”. 2. Updated Section 4.4 to add note on User Mode. 3. Chapter 6 “Register Details” is updated. 4. Added critical section details table in section 4.3 5. Chapter 14 “Release Details” is updated. 6. Chapter 8 is updated for Stub C Header files and added the description of the stub files in Table 8-1. 7. Updated the Table 4-1 Supervisor Mode and User Mode Details. 8. Updated Table 6-1 and Table 11-2 9. Section 4.6 register write verify has added. 10. Chapter 5 Architecture Details is updated. 11. Section 7.1 Services Provided by MCU driver component to user is updated. 12. Section MCU driver generation tool has updated with Mcu_Cbk.h header file in chapter 8. 13. Section 13.2.1 is updated with 701371 series. 14. Device name R7F701370A, R7F701371 and R7F701372, updated in chapter13. 15. Section 4.1 updated with forethought on ‘McuLoopCount’ parameter. 16. Os.c and SchM_Mcu.c are added in the stub files and their descriptions are included in Table 8-1 17. Updated Table 4-1 Supervisor Mode and User Mode Details. 18. Section 13.2.2 is updated with other device options. 19. Section 4.1 updated with general thought regarding CLMA4 support.	1.0.2	27-Jan-2017

AUTOSAR MCAL R4.0.3 User's Manual
MCU Driver Component Ver.1.0.2
Embedded User’s Manual

Publication Date: Rev.1.00, January 27, 2017

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.
2801 Scott Boulevard Santa Clara, CA 95050-2549, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

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: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100191, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, P. R. China 200333
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, 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 India Pvt. Ltd.
No.777C, 100 Feet Road, HAL II Stage, Indiranagar, Bangalore, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
12F., 234 Teheran-ro, Gangnam-Gu, Seoul, 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 4.1

AUTOSAR MCAL R4.0.3

User's Manual

Renesas Electronics Corporation

R20UT3651EJ0100