Z20K14xM MCU Driver User Manual

Table of contents

1. Revision history

Revision	Date	Author	Description
1.0	06.04.2023	ZhiXin MCAL Team	New creation

2. About this manual

2.1. Scope and purpose

This document describes MCU driver for Z20K14xM. This document provide deviations from the specification and limitations of MCU driver. AUTOSAR MCU driver requirements and APIs are described in the AUTOSAR MCU driver software specification document.

The purpose of this document is to enable users to integrate the MCU driver with basic software (BSW) stack.

2.2. Intended audience

This document is intended for anyone using the MCU driver of the Z20K14xM MCAL software.

2.3. Glossary

Term	Description
uC	Micro Controller
API	Application Interface
AoU	Assumption of Use
AUTOSAR	Automotive Open System Architecture
BSW	Basic software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
ECU	Electronic Control Unit
LVD	Low Voltage Detect
LVW	Low Voltage Warning
MCU	Micro Controller Unit
PARCC	Peripheral Access & Reset & Clock Controller
PMU	Power Management Unit
SCC	System Clock Controller
SCM	System Control Module

Term	Description
SRMC	System Reset and Mode Control

2.4. Reference documents

This manual should be read in conjunction with the following documents:

1. Requirements on MCU Driver, AUTOSAR_SRS_MCUDriver.pdf, AUTOSAR release R20-11
2. Specification of MCU Driver, AUTOSAR_SWS_MCUDriver.pdf, AUTOSAR release R20-11
3. Specification of Diagnostic Event Manager, AUTOSAR_SWS_DiagnosticEventManager.pdf, AUTOSAR release R20-11
4. Specification of Default Error Tracer, AUTOSAR_SWS_DefaultErrorTracer.pdf, AUTOSAR release R20-11
5. Specification of RTE Software, AUTOSAR_SWS_RTE.pdf, AUTOSAR release R20-11
6. Z20K14xM Series Reference Manual

3. Introduction

The MCU driver accesses the microcontroller hardware directly and is located in the Microcontroller Abstraction Layer(MCAL).

MCU driver Features:

1. Initialization of MCU clock, PLL, clock prescalers and MCU clock distribution
2. Initialization of RAM section
3. Activation of uC reduced power modes
4. Activation of a uC reset
5. Provides a service to get the reset reason from hardware

4. Deviations and limitations

4.1. Deviations

The following are the the deviations from software specification.

Requirement	Description	Status	Comments
SWS_Mcu_00051	The MCU driver follows the standardized AUTOSAR concept to report production errors. The provided callback routines are specified in the Diagnostic Event Manager (DEM) specification.	Not implemented	
SWS_Mcu_00053	If clock failure notification is enabled in the configuration set and a clock source failure error occurs, the error code MCU_E_CLOCK_FAILURE shall be reported.	Not implemented	
SWS_Mcu_00257	Fail criteria for MCU_E_CLOCK_FAILURE: a clock source failure occurs.	Not implemented	
SWS_Mcu_00258	Pass criteria for MCU_E_CLOCK_FAILURE: no clock source failure occurs.	Not implemented	
SWS_Mcu_00054	The structure Mcu_ConfigType shall provide a configurable (enable/disable) clock failure notification if the MCU provides an interrupt for such detection.	Not implemented	
SWS_Mcu_00207	API Mcu_GetRamState()	Not implemented	Hardware does not support this functionality.
SWS_Mcu_00209	The function Mcu_GetRamState shall be available to the user if the pre-compile parameter McuGetRamStateApi is set to TRUE. Instead, if the former parameter is set to FALSE, this function shall be disabled(e.g. the hardware does not support this functionality).	Not implemented	Hardware does not support this functionality.
SWS_Mcu_00259	The MCU Driver module shall reject configurations with partition mappings which are not supported by the implementation.	Not implemented	
SWS_Mcu_00116	If the hardware allows for only one usage of the register, the driver module implementing that functionality is responsible for initializing the register.	Out of scope	

Requirement	Description	Status	Comments
SWS_Mcu_00244	If the register can affect several hardware modules and if it is an I/O register, it shall be initialised by the PORT driver.	Out of scope	
SWS_Mcu_00245	If the register can affect several hardware modules and if it is not an I/O register, it shall be initialised by this MCU driver.	Out of scope	
SWS_Mcu_00246	One-time writable registers that require initialisation directly after reset shall be initialised by the startup code.	Out of scope	
SWS_Mcu_00247	All other registers not mentioned before shall be initialised by the start-up code.	Out of scope	
SWS_Mcu_00136	The MCU module’s environment shall call the function Mcu_InitRamSection only after the MCU module has been initialized using the function Mcu_Init.	Out of scope	
SWS_Mcu_00139	The MCU module’s environment shall only call the function Mcu_InitClock after the MCU module has been initialized using the function Mcu_Init.	Out of scope	
SWS_Mcu_00145	The MCU module’s environment shall only call the function Mcu_PerformReset after the MCU module has been initialized by the function Mcu_Init.	Out of scope	
SWS_Mcu_00148	The MCU module’s environment shall only call the function Mcu_SetMode after the MCU module has been initialized by the function Mcu_Init.	Out of scope	
SWS_Mcu_00208	The MCU module’s environment shall call this function only if the MCU module has been already initialized using the function MCU_Init.	Out of scope	

Requirement	Description	Status	Comments
SWS_Mcu_00166	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value. This API will be available only if ({Dem/DemConfigSet/DemEventParameter/DemEventReportingType} == STANDARD_REPORTING)	Out of scope	
SWS_Mcu_CONSTR_00001	The module will operate as an independent instance in each of the partitions, means the called API will only target the partition it is called in.	Out of scope	

4.2. Limitations

In general the activation and configuration of MCU reduced power mode is not mandatory within AUTOSAR standardization.

Enabling/disabling of the ECU or uC power supply is not the task of the MCU driver. This is to be handled by the upper layer.

5. Hardware software mapping

This section describes the system view of the driver and hardware peripherals.

Figure 1. Hardware software mapping

5.1. PARCC: primary hardware peripheral

The MCU driver uses the PARCC IP for configure functional clock source and divide, supervisor mode access control, register write protection and module level clock behavior control in stop mode of every peripheral.

The key hardware functional features used by the driver are:

¥ Peripheral register write protection.

¥ Optional peripheral register access protection in supervisor mode.

¥ Reset peripheral.

¥ Functional clock source switch.

¥ Functional clock source divide by specified ratio.

¥ Three kinds of clock mode:

    ! Always off mode.

    ! Always on mode.

    ! Stop off mode.

5.2. PMU: primary hardware peripheral

The MCU driver uses the PMU IP for configure system modes, configure LVD and LVW for VDD.

The key hardware functional features used by the driver are:

¥ The Power Management Unit (PMU) provides multiple power modes allowing users to optimize power consumption to level the required functionality. The primary modes of operation are:

  ! RUN

  ! WAIT

  ! STOP

  ! STANDBY

¥ Low voltage warning detect circuit (LVW) for VDD.

¥ Low voltage detect circuit (LVD) for VDD.

5.3. SCC: primary hardware peripheral

The MCU driver uses the SCC IP for configure the MCU system clock and produce reference clocks to other modules.

The key hardware functional features used by the driver are:

¥ Crystal oscillator(OSC):

  ! Optional frequency mode for a high speed mode and low speed mode.

  ! Optional Loss of clock(LOC) detector to request system reset or interrupt.

  ! Optional crystal bypass mode.

  ! Optional enable or disable in wait, stop and debug mode.

¥ Internal RC oscillator (FIRC):

  ! Optional Loss of clock(LOC) detector to request system reset or interrupt.

  ! Optional enable or disable in stop mode.

¥ System Phase Locked Loop(SPLL):

  ! Voltage controlled oscillator(VCO).

  ! Optional Loss of lock(LOL) detector to request system reset or interrupt.

  ! Programmable post divider for VCO clock output.

  ! Programmable pre divider for VCO reference clock input.

  ! OSC or FIRC can be selected as reference clock.

  ! Optional enable or disable in stop mode.

¥ Either SPLL, OSC or FIRC can be selected as system clock.

¥ Up to 16 frequency divisions for generating core/bus/slow clock from system clock.

5.4. SCM: primary hardware peripheral

The MCU driver uses the SCM IP for configure chip configuration registers.

The key hardware functional features used by the driver are:

¥ Clock output configuration

¥ FPU Error Detect

¥ Cache Parity Error Detect

¥ Flash and system RAM size configuration

¥ Timer clock and channel select on and configuration

5.5. SRMC: primary hardware peripheral

The MCU driver uses the SRMC IP for get microcontoller reset reason, configure system interrupts, configure reset pin filter in different mode and switch mode.

The key hardware functional features used by the driver are:

¥ Power On Reset (POR)

¥ System resets:

! External pin reset (PIN)

! Low voltage detect (LVD)

! Watchdog reset

! Loss of clock reset

! Stop mode acknowledge error (SACKERR)

! Software reset (SW)

! Core Lockup reset (LOCKUP)

! SERU system reset

! SERU cold reset

¥ Chip Mode of operation:

! RUN mode: The MCU can run at full speed and the internal supply is fully regulated

! WAIT mode: CPU is under sleep mode, NVIC is sensitive to interrupt. Peripherals continued to be clocked

! STOP mode: In STOP mode system clock and core clock, and the bus clocks are gated. CPU is under deep sleep mode, NVIC is disabled. Peripherals clocks are off. SOG and SRAM are powered on.

! STANDBY mode: Most peripherals off except AO domain like RTC, REGFILE, etc.

6. File structure

This section provides details of the C files of the MCU driver.

Figure 2. C File structure

File name	Description
Mcu.c	AUTOSAR level source file containing implementations of APIs
Mcu.h	AUTOSAR level header file containing declarations of APIs
Mcu_Types.h	AUTOSAR level header file containing definition of data types
Mcu_Cfg.h	AUTOSAR level configuration header file (Generated) containing pre-processor macros
Mcu_PBcfg.c	AUTOSAR level post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Mcu_PBcfg.h	AUTOSAR level post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Mcu_Drvw.c	Mcu driver wrapper source file containing implementations of APIs
Mcu_Drvw.h	Mcu driver wrapper header file containing declarations of APIs
Mcu_Drvw_Types.h	Mcu driver wrapper header file containing definition of data types
Mcu_Drvw_Cfg.h	Mcu driver wrapper configuration header file (Generated) containing pre-processor macros

File name	Description
Mcu_Drvw_PBcfg.c	Mcu driver wrapper post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Mcu_Drvw_PBcfg.h	Mcu driver wrapper post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Parcc_Drv.c	PARCC Low-level driver source file containing implementations of APIs
Parcc_Drv.h	PARCC Low-level driver header file containing declarations of APIs
Parcc_Drv_Types.h	PARCC Low-level driver header file containing definition of data types
Parcc_Drv_Cfg.h	PARCC Low-level driver configuration header file (Generated) containing pre-processor macros
Parcc_Drv_PBcfg.c	PARCC Low-level driver post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Parcc_Drv_PBcfg.h	PARCC Low-level driver post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Pmu_Drv.c	PMU Low-level driver source file containing implementations of APIs
Pmu_Drv.h	PMU Low-level driver header file containing declarations of APIs
Pmu_Drv_Types.h	PMU Low-level driver header file containing definition of data types
Pmu_Drv_Cfg.h	PMU Low-level driver configuration header file (Generated) containing pre-processor macros
Pmu_Drv_PBcfg.c	PMU Low-level driver post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Pmu_Drv_PBcfg.h	PMU Low-level driver post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Ram_Drv.c	RAM Low-level driver source file containing implementations of APIs
Ram_Drv.h	RAM Low-level driver header file containing declarations of APIs
Ram_Drv_Types.h	RAM Low-level driver header file containing definition of data types
Ram_Drv_Cfg.h	RAM Low-level driver configuration header file (Generated) containing pre-processor macros
Ram_Drv_PBcfg.c	RAM Low-level driver post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Ram_Drv_PBcfg.h	RAM Low-level driver post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Scc_Drv.c	SCC Low-level driver source file containing implementations of APIs
Scc_Drv.h	SCC Low-level driver header file containing declarations of APIs

File name	Description
Scc_Drv_Types.h	SCC Low-level driver header file containing definition of data types
Scc_Drv_Cfg.h	SCC Low-level driver configuration header file (Generated) containing pre-processor macros
Scc_Drv_PBcfg.c	SCC Low-level driver post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Scc_Drv_PBcfg.h	SCC Low-level driver post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Scm_Mcu_Drv.c	SCM Low-level driver source file containing implementations of APIs
Scm_Mcu_Drv.h	SCM Low-level driver header file containing declarations of APIs
Scm_Mcu_Drv_Types.h	SCM Low-level driver header file containing definition of data types
Scm_Mcu_Drv_Cfg.h	SCM Low-level driver configuration header file (Generated) containing pre-processor macros
Scm_Mcu_Drv_PBcfg.c	SCM Low-level driver post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Scm_Mcu_Drv_PBcfg.h	SCM Low-level driver post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Srmc_Drv.c	SRMC Low-level driver source file containing implementations of APIs
Srmc_Drv.h	SRMC Low-level driver header file containing declarations of APIs
Srmc_Drv_Types.h	SRMC Low-level driver header file containing definition of data types
Srmc_Drv_Cfg.h	SRMC Low-level driver configuration header file (Generated) containing pre-processor macros
Srmc_Drv_PBcfg.c	SRMC Low-level driver post-build configuration source file (Generated) containing definition of the post-build configuration data structures
Srmc_Drv_PBcfg.h	SRMC Low-level driver post-build configuration header file (Generated) containing declaration of the post-build configuration data structures
Device_Regs.h	Registers definition header file containing type definition of register data structure and IRQ number
McalLib.h	McalLib header file containing declarations of APIs and definitions of data types
SchM_Mcu.h	MCU SchM header file containing enter and exit exclusive areas function declarations of MCU driver
Mcu_MemMap.h	MCU driver memory mapping header file containing definitions of memory section
Det.h	header file of Det module containing declarations of APIs
Dem.h	header file of Dem module containing declarations of APIs

7. Configuration tips

This section lists the configuration tips that an integrator or user of the MCU driver may need.

7.1. Configure Crystal Oscillator (OSC) Clock

¥ Enable/Disable OSC clock

¥ Enable/Disable high frequency mode

    ! It is recommended to enable this when using high frequency crystal(typically >= 24Mhz), due to deviation of crystal.

¥ Set crystal frequency

    ! Crystal frequency shall be between 8Mhz and 40Mhz.

¥ Configure OSC clock source(internal crystal oscillator or external crystal oscillator)

¥ Enable/Disable FOSC clock in stop mode.

Figure 3. Configure Crystal OSC Clock

7.2. Configure FIRC Clock

¥ Enable/Disable FIRC clock

¥ Enable/Disable FIRC clock in stop mode.

Figure 4. Configure FIRC Clock

7.3. Configure PLL Clock

¥ Enable/Disable PLL clock

¥ Enable/Disable PLL clock in stop mode

¥ Select PLL clock reference clock source(OSC clock or FIRC clock)

Figure 5. Configure PLL Clock

7.4. Configure System Clock Source

Either SPLL, OSC or FIRC can be selected as system clock.

Figure 6. Select System Clock Source

7.5. Configure Peripheral Access Control and Functional Clock

Each peripheral has independent configuration of supervisor mode access control, register write protection and functional clock source and divide.

¥ Enable/Disable peripheral supervisor access control.

    ! If enabled, all of registers in this peripheral is writable only in supervisor mode and write operation request would leads to hard fault interrupt in user mode.

¥ Enable/Disable peripheral write lock

    ! If enabled, all registers in this peripheral is read only and write operation would be ignored.

¥ Configure functional clock and divide for this peripheral

Figure 7. Configure Peripheral Access Control and Functional Clock

7.6. Configure Reference Clock

The MCU driver provides reference clocks for other modules refer as an input value.

Figure 8. Configure Reference Clock

7.7. Configure Modules Initialized in MCU Driver

The initialization of PMU, SCM and SRMC module is configurable in the MCU driver.

If McuPmuInitDisabled/McuScmInitDisabled/McuSrmcInitDisabled is selected, module initialization will not be executed in Mcu_Init().

Figure 9. Configure Module Initialized in MCU Driver

7.8. Configure Standby Mode and Wakeup Source

¥ Enable/Disable allow the MCU to enter standby mode.

¥ Configure wakeup source and polarity of each wakeup source

Figure 10. Configure Standby Mode and Wakeup Source

7.9. Configure User Defined Callback Before Performing MCU Reset

The MCU driver supports invoking user defined function before performing MCU Reset.

If McuUserDefinedBeforeResetCalloutEn is enabled, user defined callback function can be configured in McuUserDefinedBeforeResetCallout.

Figure 11. Configure User Defined Callback Before Performing MCU Reset

7.10. Configure MCU Mode

Due to Mcu_SetMode API entertains only the configured modes, modes shall be configured before using.

Figure 12. Configure MCU Mode

7.11. Configure Chip

¥ FPU error interrupt configuration The interrupt of FPU errors(incl. inexact, overflow, underflow, invalid-op, div-zero and de-normal error) is configurable in the MCU driver.

¥ Cache configuration

¥ SRAM read buffer configuration

¥ Timer clock and channel selection and configuration

Figure 13. Configure Chip

7.12. Configure System Reset and Reset Interrupt

¥ Maximum reset delay time

¥ Enable/Disable system reset interrupts(incl. core lockup interrupt, External reset pin interrupt, Loss of clock interrupt, SERU system reset interrupt, Software reset interrupt, Stop acknowledge error interrupt, Watchdog interrupt)

Figure 14. Configure System Reset and Reset Interrupt

7.13. Configure LVD and LVW for VDD

¥ Enable/Disable VDD5v LVD(low voltage detect circuit) under low power mode or active mode

¥ Enable/Disable VDD5v LVD(low voltage detect circuit) interrupt

¥ Enable/Disable VDD5v LVD(low voltage detect circuit) reset

¥ Enable/Disable VDD5v LVW(low voltage warning detect circuit)

¥ Enable/Disable VDD5v LVW(low voltage warning detect circuit) interrupt

¥ Enable/Disable REF 1V buffer

Figure 15. Configure LVD and LVW for VDD

8. Integration hints

This section lists the key points that an integrator or user of the MCU driver must consider.

8.1. Integration with AUTOSAR stack

This section lists the modules, which are not part of the MCAL, but are required to integrate the MCU driver.

EcuM

The ECU Manager module is a BSW module that manages common aspects of ECU. Specifically, the ECU Manager module is used for initialization and de-initialization of the software drivers.

The EcuM module is implemented as stub code in the MCAL package, and it must be replaced with a complete EcuM module during the integration phase.

Det

The DET module is is a BSW module that handles all detected development and runtime errors reported by the BSW modules. The MCU driver reports all the development errors to the DET module through the Det_ReportError() API, and report runtime errors to the DET module through Det_ReportRuntimeError() API.

The Det module is implemented as stub code in the MCAL package, and it must be replaced with a complete Det module during the integration phase.

SchM

The SchM is a part of the RTE that apply data consistency mechanisms for BSW modules. ExclusiveAreas mechanism is used to guarantee data consistency. The ExclusiveArea concept is to block potential concurrent accesses to get data consistency. ExclusiveAreas implement critical section.

The MCU driver uses the exclusive areas defined in SchM_Mcu.c file to protect the registers and variables access.

The SchM_Mcu.c and SchM_Mcu.h files are provided in the MCAL package as an example code and needs to be updated by the integrator.

Memory mapping

AUTOSAR specifies mechanisms for the mapping of code and data to specific memory sections via memory mapping files. For many ECUs and microcontroller platforms it is of utmost necessity to be able to map code, variables and constants to specific memory sections.

Memory mapping macros are provided to select specific memory sections. These macros are defined in the Mcu_MemMap.h file.

The Mcu_MemMap.h file is provided as an example code in the MCAL package, and it must be replaced with appropriate compiler pragmas during the integration phase.

Callback and Notification

The MCU driver does not provide any callbacks or notifications.

8.2. Interrupt connections

The interrupt connection of the MCU driver is described in this section.

SCC Interrupt

If corresponding interrupt of SCC is enabled in platform module and FIRC clock monitor or OSC clock monitor is enabled with action of generate interrupt when loss of corresponding clock is detected, an interrupt request will be generated.

Figure 16. Configure SCC interrupt in Platform module

An example code of SCC ISR is provided in Scc_Drv_Irq.c as below:

```
ISR(Scc_Drv_IrqHandler)
{
Ê   Scc_Drv_IntHandler();
Ê   EXIT_INTERRUPT();
}
```

9. Assumptions of Use

The following are the AoUs for the driver:

Correct pointer of configuration structure

User of the MCU driver shall ensure that correct pointer to the configuration structure is passed for initializing the driver.

RAM section base address

User shall provide the base address for the RAM section as per the natural memory alignment of the memory type.

Software reset configuration

User shall ensure that when the Mcu_PerformReset API is called to perform software reset, the McuPerformResetApi parameter shall not be configured as disabled.

Get power mode state configuration

User shall ensure that when the Mcu_GetPowerModeState API is called to get current power mode, the McuGetPowerModeStateApi parameter shall not be configured as disabled.

Loss-Of-Clock(LOC) Reset

In order to prevent unexpected loss of clock reset events, all clock monitors should be disabled prior to entering into STOP modes.

How to Reach Us:

Home Page:

zhixin-semi.com/

Document Number: Z20K14xM-MCAL-MCU-UM

Revision 1.0.0, April, 2023

[redacted]