RZ/T1 Group

CMSIS-RTOS RTX for Cortex-R4
RTX Sample Programs
(for use with the following combinations of environments and compilers:
EWARM and ICCARM, e2studio and Renesas GCC, DS-5 and ARMCC)

R01AN3538EJ0100
Rev.1.00
Mar. 13, 2017

Introduction

This application note describes the CMSIS-RTOS RTX sample applications for RZ/T1 devices.

The features of the CMSIS-RTOS RTX sample programs are described below.

The programs have boot modules for each operating mode of the device: a 16-bit bus boot version, an SPI boot version and a RAM execution version, which only uses the internal RAM (tightly coupled memory, TCM) and does not require flash memory.

Each sample program has two sections, the loader program and the actual CMSIS-RTOS RTX sample programs. After booting, initial settings of the RZ/T1 are made, the CMSIS-RTOS is initialized, and the sample application is run, in that sequence.

The loader program, after booting the RZ/T1, initializes the clock generator and bus state controller, and copies the sample program.

The following are provided as the sample programs (see Table 5.1 for details).

FPU_ex1

mail

message

RTX_CMT_ex1

RTX_ex1

RTX_ex2

RTX_MTU3_ex1

RTX_Traffic

Semaphore

Target Devices

RZ/T1

When applying the program covered in this application note to another microcontroller, modify the program according to the specifications for the target microcontroller and extensively evaluate the modified program.

Table of Contents

1. Specifications

Table 1.1 lists the peripheral modules used and their applications.

Table 1.1 Peripheral Modules and Their Applications

Peripheral Module	Application
CPU (R7X910017)	For program operation
RZ/T1 internal bus state controller	Reading the program itself when booting up in 16-bit bus boot*1 mode (from NOR flash memory) and access to the SDRAM
RZ/T1 internal SPI multi-I/O bus controller	Reading the program itself when booting up in SPI boot*2 mode
NOR flash memory (MX29GL512FLT2I-10Q)	Storage area for the program when booting up in 16-bit bus boot*1 mode (from NOR flash memory)
Serial flash memory (MX25L51245GMI-10G)	Storage area for the program when booting up in SPI boot mode
RZ/T1 internal RAM (ATCM, BTCM)	Storage area for the program in RAM boot*3 mode Program data storage area
RZ/T1 internal clock pulse generator	Controlling the clock supply to various internal macros of the RZ/T1. Generating frequencies for various clock sources (specifying the multiplication factors).
RZ/T1 internal general I/O port	RZ/T1 multiplexed pin
RZ/T1 internal interrupt controller (ICUA)	Interrupt control

Note 1. 16-bit bus boot: Booting up from the NOR flash memory (hereafter, "NOR boot")
Note 2. SPI boot: Booting up from the serial flash memory (hereafter, "SPI boot")
Note 3. RAM boot: Booting by using JTAG-ICE for direct downloading to the internal RAM (hereafter, "RAM boot")

2. Operating Environment

Operation of the sample code covered in this application note has been confirmed under the conditions below.

Table 2.1 Operating Environment

Item	Description
MCU used	RZ/T1 Group
Operating frequency	CPUCLK = 450 MHz
Operating voltage	3.3 V
Integrated development environment	• Embedded Workbench for ARM ((EWARM), version 7.80.2, from IAR Systems • e2studio, version 5.2.0.020, from Renesas • DS-5, version: 5.25.0, from ARM
Operating mode	SPI boot mode SW4: ON/ON/ON
	RAM boot mode or NOR boot mode SW4: ON/OFF/ON
Board used	RZ/T1 evaluation board (RTK7910018C00000BE)
Devices used	• NOR flash memory (connected to the CS0 and CS1 spaces) Manufacturer: Macronix International Co., Ltd. Model: MX29GL512FLT2I-10Q • SDRAM (connected to the CS2 and CS3 spaces) Manufacturer: Integrated Silicon Solution Inc. Model: IS42S16320D-7TL • Serial flash memory Manufacturer: Macronix International Co., Ltd. Model: MX25L51245GMI-10G
ICE	• I-jet JTAG emulator from IAR Systems • J-Link JTAG emulator from SEGGER • KEIL ULINK2 emulator from ARM

3. Documents

3.1 Related Documents

The documents related to descriptions in this application note are listed below for reference.

- CMSIS-RTOS compliant Kernel Version 4.74
  This is the specification for RTOS for use in this system. The sample application uses the functions of the CMSIS-RTOS RTX. Each sample application initializes the CMSIS-RTOS RTX.

- RZ/T1 Group User's Manual: Hardware (R01UH0483)
  This document describes the hardware specifications of RZ/T1 devices.
  Download the latest version from the Renesas Electronics website.

- Application Note: RZ/T1 Group Initial Settings (R01AN2554)
  This document describes the initial settings for RZ/T1 devices. For those not covered in this application note, the values set in *Application Note: RZ/T1 Group Initial Settings* are used.
  Download the latest version from the Renesas Electronics website.

- Application Note: RZ/T1 Group CMSIS-RTOS RTX for Cortex-R4 CMT(W) & ELC & ADC Sample Programs (R01AN3539EJ)
  This document describes a sample program for RZ/T1 devices, which handles A/D conversion triggered by a timer. A 32-bit timer (compare-match timer W, CMTW) is interlinked with the ADC through the event link controller (ELC).

- Application Note: RZ/T1 Group CMSIS-RTOS RTX for Cortex-R4 MTU3a Sample Program (R01AN3540EJ)
  This document describes a sample program that sets up the output of positive and negative PWM waveforms in three phases (six signals in total) that include dead time, by using the complementary PWM mode of MTU3 and MTU4 of the multi-function timer pulse unit (MTU3a) of RZ/T1 devices.

- Technical Update and Technical News
  Download the latest version from the Renesas Electronics website.

- User's manuals related to the development environment
  The latest version of the IAR integrated development environment (IAR Embedded Workbench for ARM) is available from the IAR Systems website.

  The latest version of the DS-5 integrated development environment (ARM Development Studio 5) is available from the ARM website.

  The latest version of the Renesas Electronics software development tools (e2studio, etc.) is available from the Renesas Electronics website.

4. Hardware

4.1 Example of the Hardware Configuration

The figure below shows an example of connection.

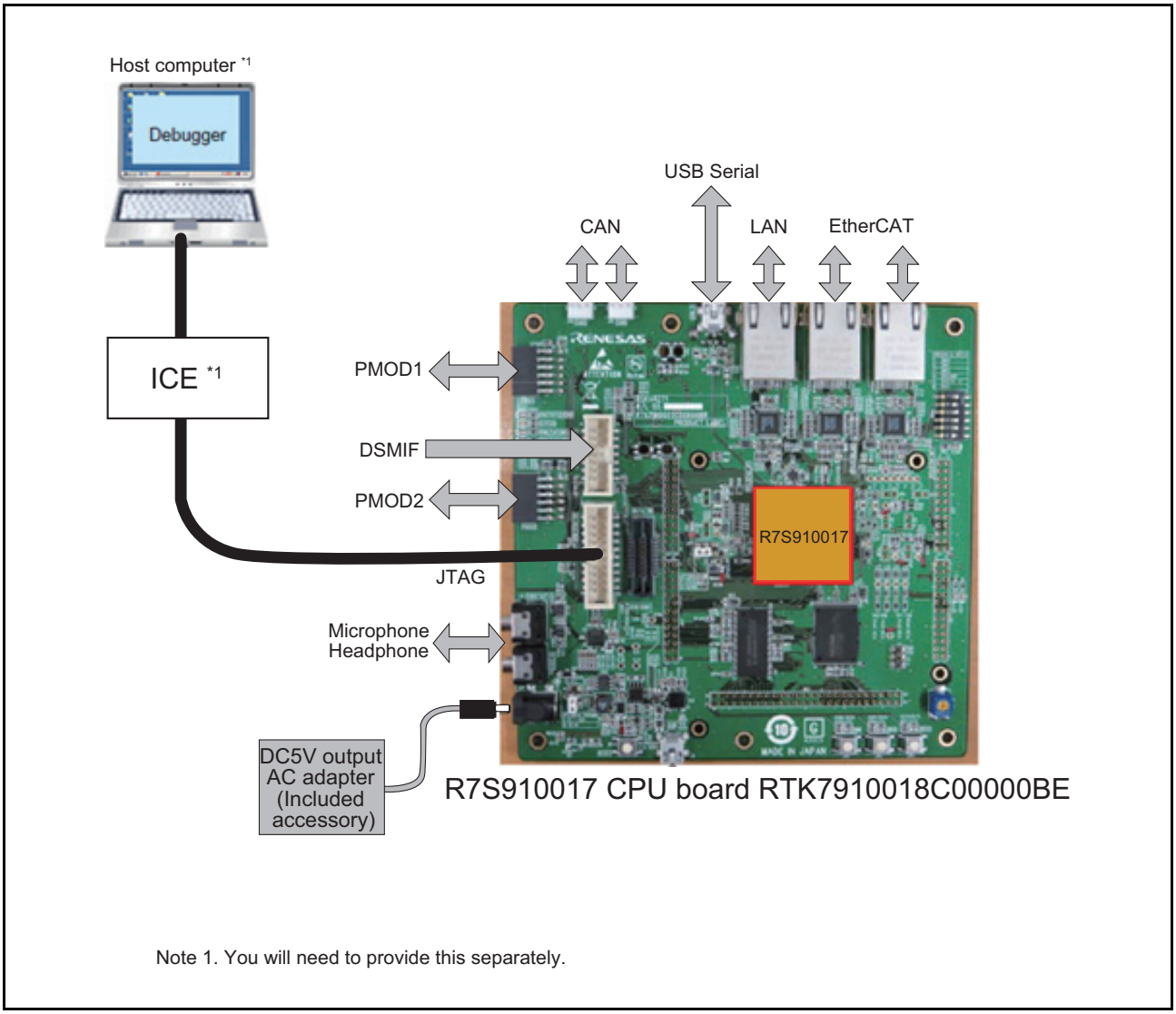


Figure 4.1 Example of Connection

4.2 Pins

Table 4.1 lists the pins used and their functions.

Table 4.1 Pins Used and Their Functions (1 / 3)

Pin Name	I/O	Description	RZ/T1 Pin Setting	Board Connection
MD0	Input	Selection of the operating mode MD0 = "L", MD1 = "L", MD2 = "L" (SPI boot mode) MD0 = "L", MD1 = "H", MD2 = "L" (NOR boot mode)	MD0 dedicated pin	SW4-1 Note: The combination of MD0 to MD2 selects the operating mode.
MD1	Input		MD1 dedicated pin	SW4-2 Note: The combination of MD0 to MD2 selects the operating mode.
MD2	Input		MD2 dedicated pin	SW4-3 Note: The combination of MD0 to MD2 selects the operating mode.
A21 to A25	Output	Address specification in the CSn spaces	PT6, PT7 PK2, PK3 P97 Pin function select bit = 35 Operating mode = SPI boot mode	SDRAM
A16 to A20	Output	Address specification in the CSn spaces	PH7 P20, P25 to P27 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode (automatically set for NOR boot mode)	NOR flash memory SDRAM
A1 to A15	Output	Address specification in the CSn spaces	PG0 to PD7 PH0 to PH6 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode (automatically set for NOR boot mode)	NOR flash memory SDRAM
A0	Output	Address specification in the CSn spaces	P23 Pin function select bit = 34 Operating mode = SPI boot mode	NOR flash memory SDRAM
D0 to D15	I/O	Transferring I/O data to and from the CSn spaces	P00 to P07 PE0 to PE7 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode (automatically set for NOR boot mode)	NOR flash memory SDRAM
CS0#	Output	Chip select signal for the CS0 space	P21 Pin function select bit = 34 Operating mode = NOR boot mode (automatically set for NOR boot mode)	NOR flash memory
CS1#	Output	Chip select signal for the CS1 space	PD1 Pin function select bit = 35 Operating mode = SPI boot mode, NOR boot mode	SDRAM
RD/WR#	Output	Enabling output for reading data from the CS0 and CS1 spaces	P24 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode (automatically set for NOR boot mode)	NOR flash memory SDRAM
CS2#	Output	Chip select signal for the CS2 space	P45 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode	SDRAM

Table 4.1 Pins Used and Their Functions (2 / 3)

Pin Name	I/O	Description	RZ/T1 Pin Setting	Board Connection
CS3#	Output	Chip select signal for the CS3 space	PT4 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode	SDRAM
RAS#	Output	Connection to the RAS# pin of the SDRAM	P90 Pin function select bit = 35 Operating mode = SPI boot mode, NOR boot mode	SDRAM
CAS#	Output	Connection to the CAS# pin of the SDRAM	PK0 Pin function select bit = 35 Operating mode = SPI boot mode, NOR boot mode	SDRAM
CKE	Output	Connection to the CKE pin of the SDRAM	P46 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode	SDRAM
WE0/DQMLL	Output	Byte specification	P36 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode	SDRAM
WE1/DQMLU	Output	Byte specification	P37 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode	SDRAM
CKIO	Output	System clock output	P10 Pin function select bit = 34 Operating mode = SPI boot mode, NOR boot mode	NOR flash memory SDRAM
SPBCLK	Output	SPI multi-I/O bus controller clock output	P62 Pin function select bit = 27 Operating mode = SPI boot mode (automatically set for SPI boot mode)	Serial flash memory
SPBSSL	Output	SPI multi-I/O bus controller slave select	P60 Pin function select bit = 27 Operating mode = SPI boot mode (automatically set for SPI boot mode)	Serial flash memory
SPBMO/ SPBIO0	I/O	Data input to and output from channel 0 of the SPI multi-I/O bus controller	P63 Pin function select bit = 27 Operating mode = SPI boot mode (automatically set for SPI boot mode)	Serial flash memory
SPBMI/ SPBIO1	I/O	Data input to and output from channel 0 of the SPI multi-I/O bus controller	P64 Pin function select bit = 27 Operating mode = SPI boot mode (automatically set for SPI boot mode)	Serial flash memory
SPBIO2	I/O	Data input to and output from channel 0 of the SPI multi-I/O bus controller	P65 Pin function select bit = 27 Operating mode = SPI boot mode (automatically set for SPI boot mode)	Serial flash memory
SPBIO3	I/O	Data input to and output from channel 0 of the SPI multi-I/O bus controller	P61 Pin function select bit = 27 Operating mode = SPI boot mode (automatically set for SPI boot mode)	Serial flash memory
RxD2	Input	Input for data received through SCIF channel 2	P91 Pin function select bit = 11	RZ/T1 evaluation board USB/serial port

Table 4.1 Pins Used and Their Functions (3 / 3)

Pin Name	I/O	Description	RZ/T1 Pin Setting	Board Connection
TxD2	Output	Input for data received through SCIF channel 2	P92 Pin function select bit = 11	RZ/T1 evaluation board USB/serial port

Note 1. The pins to be set by hardware at the time of a reset are not re-set by initialization.
Note 2. The CMSIS-RTOS RTX sample programs do not use the functions of the SCIF driver for the output of debugging messages through the SCIF. Instead, it directly outputs them through that interface.

5. Software

5.1 Operation in Outline

The operation of the CMSIS-RTOS RTX sample programs is described in outline below.

For booting after release from the reset state, the RZ/T1 copies the loader program for the given operating mode (NOR boot or SPI boot) stored in the external flash memory (NOR flash or serial flash memory) to the internal RAM (BTCM).

After booting up the device, the loader program checks whether the device has been reset, sets up the clock and bus, and then copies the user application program stored in the external flash memory (NOR or serial) to the internal RAM (ATCM). It then makes settings for the MPU and caches and branches to the start of the CMSIS-RTOS RTX user application program to handle the operations of the sample program.

- Target hardware
  RZ/T1 evaluation board on which the RZ/T1 is mounted

- For IAR Embedded Workbench for ARM Version 7.80.2
  The sample application environment is provided in the project file format of IAR Embedded Workbench for ARM, Version 7.80.2.

- For RENESAS e2studio 5.2.0.020
  The sample application environment is provided in the project file format of RENESAS e2studio 5.2.0.020.

- For ARM DS-5 Version: 5.25.0
  The sample application environment is provided in the project file format of ARM DS-5 Version: 5.25.0.

- Endian
  Only for little endian

- Booting
  Either of the following boot modes is selectable.
  – Booting up from the serial flash memory (SPI boot mode)
  – Booting up from the NOR flash memory (NOR boot mode)
  – Booting by using JTAG-ICE for direct downloading to the internal RAM (ATCM or BTCM)

- Memory area
  – The internal RAMs (ATCM and BTCM) are used for the code and data areas.

- Memory management
  – Memory management using the MPU

- Initialization of hardware
  Initialization of the following modules by initializing the platform (in order of initialization)
  – Clock pulse oscillator (system clock control)
  – General I/O port (pin function settings)
  – Bus state controller (CS0 CS1 NOR flash, CS2 CS3 SDRAM)

– CMT4 (CMSIS-RTOS RTX system timer)

- Configuration of the CMSIS-RTOS RTX
  The following settings are made for the items for configuration of the CMSIS-RTOS RTX.
  – CMT4 is used as the OS timer (the interrupt cycle is 1 ms).
  – CMT5 is used as a PendSV interrupt.
  – The idle state is WFI sleep mode.
  – Default stack size for threads: 1024 bytes
  – Stack overflow checking: Enabled
  – Thread operating mode: Privilege mode
  – Round-robin operation: Disabled
  – Number of FIFO queues for ISR events: 16
  – Default value of the maximum number of threads generated: 50
  – User timer: Enabled
  – Message queue size for timer threads: 5

The table below lists and gives outlines of the sample applications.

Table 5.1 Outlines of the Sample Applications

Sample Application	Outline
FPU_ex1	Sample application for floating-point operations
Mail	Sample application for RTX Kernel Mail Queue Management
Message	Sample application for RTX Kernel Message Management
RTX_CMT_ex1 Note: For details, see the description of the document *R01AN3539EJ in section 3.1*.	Sample application for RZ/T1 devices, which handles A/D conversion triggered by a timer. A 32-bit timer (compare-match timer W, CMTW) is interlinked with the ADC through the event link controller (ELC).
RTX_ex1	Sample application for controlling two threads by using RTX Kernel Signal Management
RTX_ex2	Sample application for controlling four threads by using RTX Kernel Signal Management
Note: For details, see the description of the document *R01AN3540EJ in section 3.1*.	Sample application that sets up the output of positive and negative PWM waveforms in three phases (six signals in total) that include dead time, by using the complementary PWM mode of MTU3 and MTU4 of the multi-function timer pulse unit (MTU3a) of RZ/T1 devices.
RTX_Traffic	Sample application for RTX Kernel Timer Management and Signal Flag Management
Semaphore	Sample application for RTX Kernel Semaphore Management

5.2 Initialization Sequence

5.2.1 EWARM Environment

The figure below shows the EWARM initialization sequence.

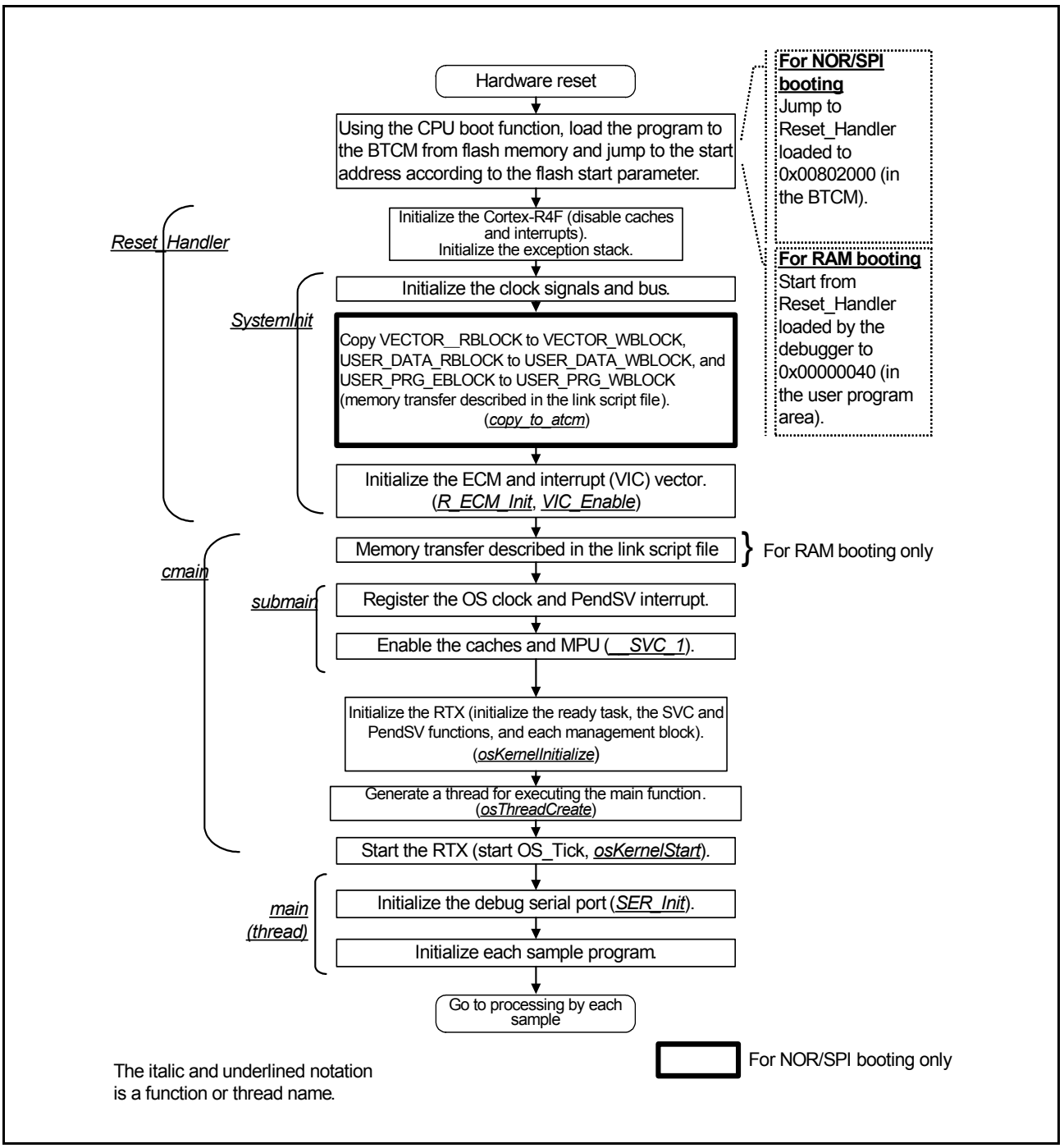


Figure 5.1 EWARM Initialization Sequence

5.2.2 e2Studio Environment

The figure below shows the e2studio initialization sequence.

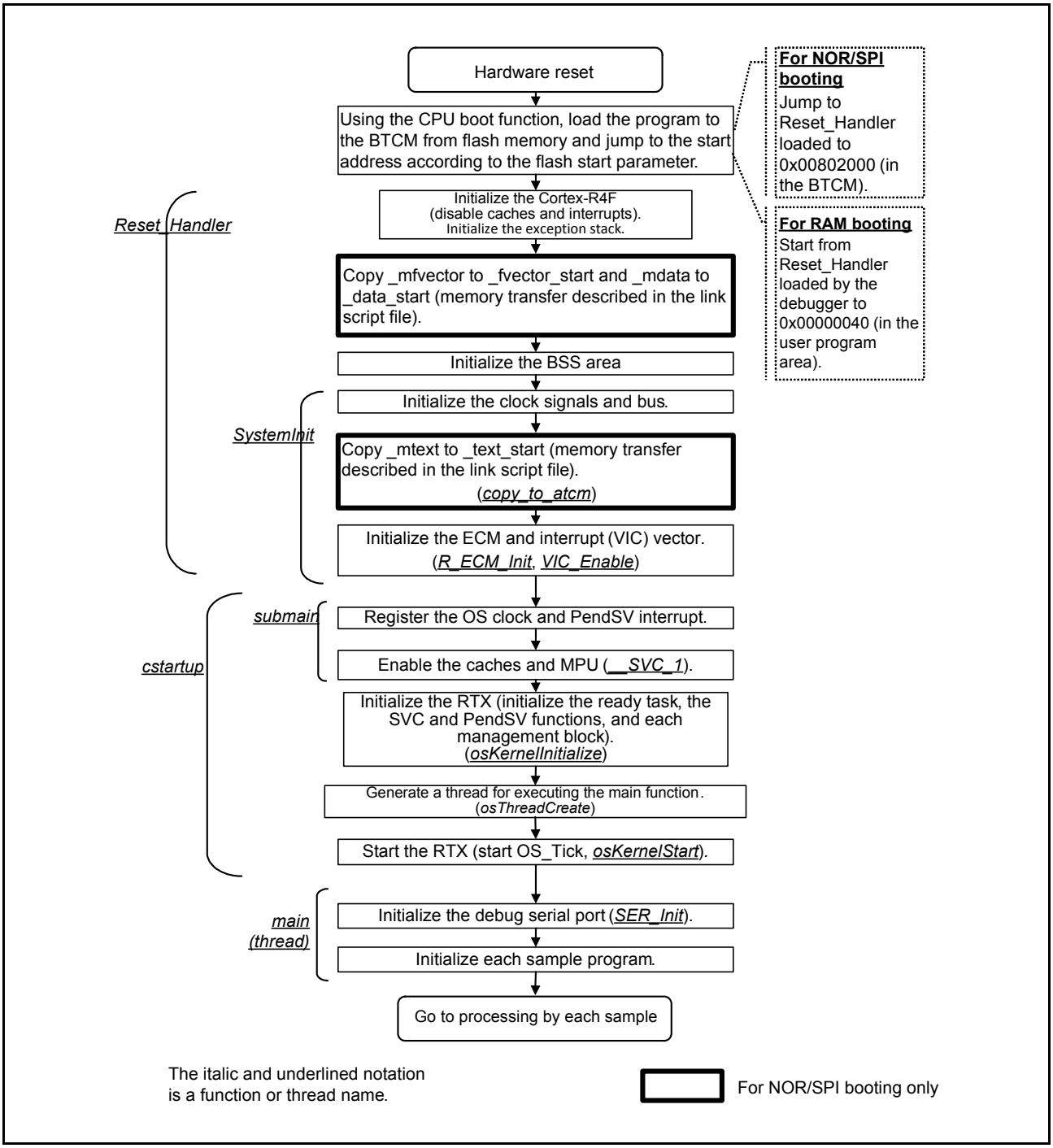


Figure 5.2 e2studio Initialization Sequence

5.2.3 DS-5 Environment

The figure below shows the DS-5 initialization sequence.

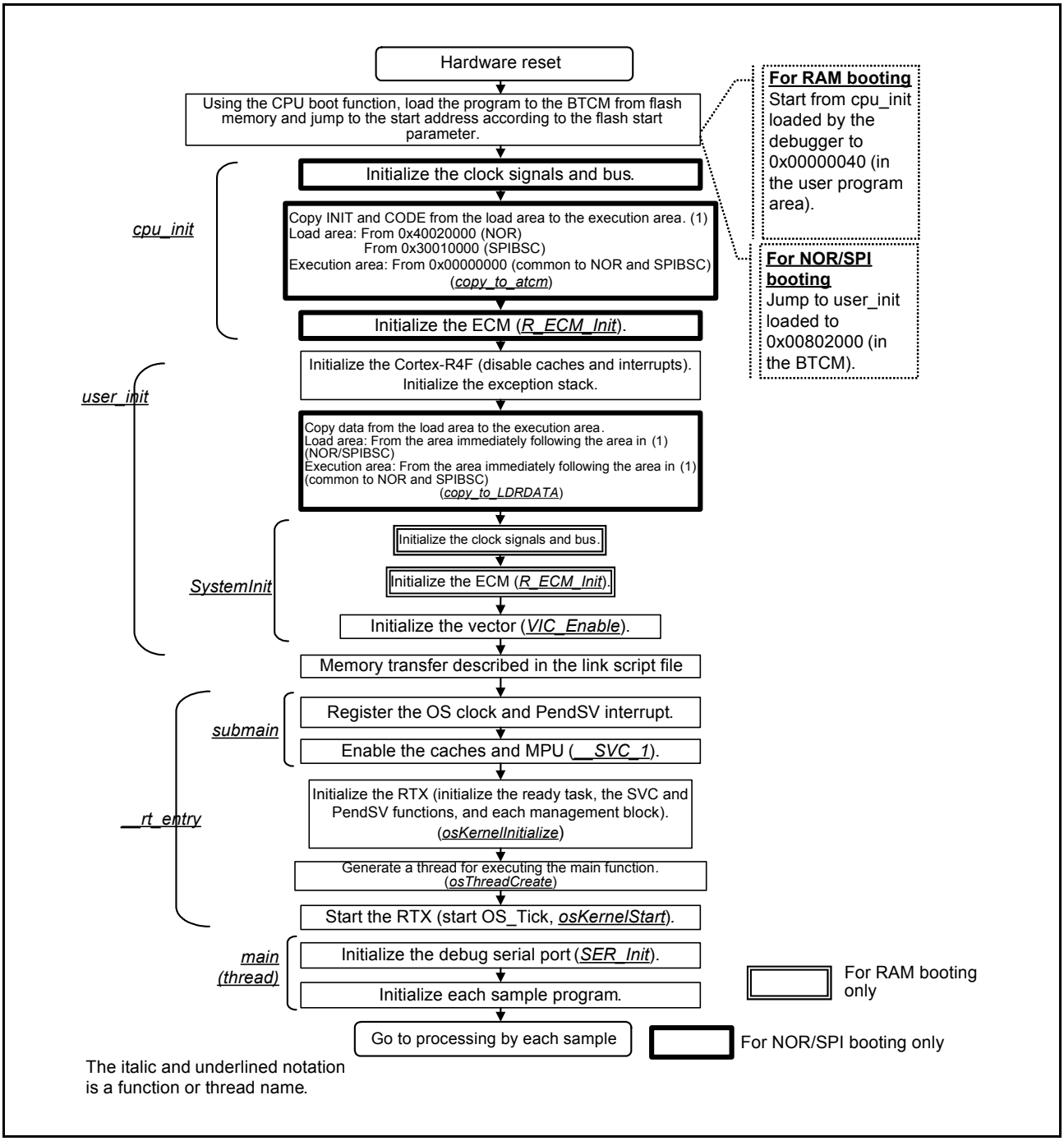


Figure 5.3 DS-5 Initialization Sequence

5.3 Configuration of Projects

The table below lists the projects included in the working sets of the CMSIS-RTOS RTX sample programs. Check the settings for each project (EWARM, e2studio, or DS-5) against the actual projects.

Table 5.2 Configuration of the Working Sets for the RZ/T1

Project	Project Outline	Boot Mode
Bootloader NorFlash	NOR boot mode (NOR flash) boot loader library (Bootloader NorFlash.a)	NOR
Bootloader SerialFlash	SPI boot mode (serial flash) boot loader library (Bootloader SerialFlash.a)	SPI
RTX_CMT_ex1_NOR	Sample application for RZ/T1 devices, which handles A/D conversion triggered by a timer. A 32-bit timer (compare-match timer W, CMTW) is interlinked with the ADC through the event link controller (ELC).	NOR
RTX_CMT_ex1_RAM		RAM
RTX_CMT_ex1_SPIBSC		SPI
RTX_MTU3_ex1_NOR	Sample application that sets up the output of positive and negative PWM waveforms in three phases (six signals in total) that include dead time, by using the complementary PWM mode of MTU3 and MTU4 of the multi-function timer pulse unit (MTU3a) of RZ/T1 devices.	NOR
RTX_MTU3_ex1_RAM		RAM
RTX_MTU3_ex1_SPIBSC		SPI
FPU_ex1_NOR	Sample application for floating-point operations	NOR
FPU_ex1_RAM		RAM
FPU_ex1_SPIBSC		SPI
Mail_NOR	Sample application for RTX Kernel Mail Queue Management	NOR
Mail_RAM		RAM
Mail_SPIBSC		SPI
Message_NOR	Sample application for RTX Kernel Message Management	NOR
Message_RAM		RAM
Message_SPIBSC		SPI
RTX_ex1_NOR	Sample application for controlling two threads by using RTX Kernel Signal Management	NOR
RTX_ex1_RAM		RAM
RTX_ex1_SPIBSC		SPI
RTX_ex2_NOR	Sample application for controlling four threads by using RTX Kernel Signal Management	NOR
RTX_ex2_RAM		RAM
RTX_ex2_SPIBSC		SPI
RTX_Traffic_NOR	Sample application for RTX Kernel Timer Management and Signal Flag Management	NOR
RTX_Traffic_RAM		RAM
RTX_Traffic_ SPIBSC		SPI
Semaphore_NOR	Sample application for RTX Kernel Semaphore Management	NOR
Semaphore_RAM		RAM
Semaphore_SPIBSC		SPI

5.4 Memory Map

The address space of the RZ/T1 group and a memory map of the RZ/T1 evaluation board are described in *Application Note: RZ/T1 Group Initial Settings*.

5.4.1 Assignment of the Sample Programs to Sections

The assignment of the CMSIS-RTOS RTX sample programs to sections is described below.

(1) Assignment of the Sample Programs for EWARM to Sections

Table 5.3 Assignment to Sections for EWARM

Area	Description	Type	Load Area	Execution Area
VECTOR_WBLOCK	Reset and exception vector table	Code	—	ATCM
USER_PRG_WBLOCK	Sample application program area (for execution)	Code	—	ATCM
USER_DATA_WBLOCK	Sample application program variable area (for execution)	Data	—	ATCM
LDR_DATA_WBLOCK*3	Loader program variable area (for execution)	Data	—	BTCM
LDR_PRG_WBLOCK*3	Loader program area (for execution)	Code	—	BTCM
ldr_param	Loader parameters*1	Data	Flash*2	—
LDR_PRG_RBLOCK*3	Loader program area (for storage)*1	Code	Flash*2	—
LDR_DATA_RBLOCK*3	Loader program variable area (for storage)*1	Data	Flash*2	—
VECTOR_RBLOCK	Reset and exception vector table (for storage)*1	Code	Flash*2	—
USER_PRG_RBLOCK	Sample application program area (for storage)*1	Code	Flash*2	—
USER_DATA_RBLOCK	Sample application program variable area (for storage)*1	Data	Flash*2	—

Note 1. The RAM execution version device does not have this area.
Note 2. This is assigned to the NOR flash memory for the NOR boot mode version and to the serial flash memory for the SPI boot mode version.
Note 3. The CMSIS-RTOS RTX sample programs require specification of functions (objects) used for the loader program in the loader program sections (LDR_PRG_xxx and LDR_DATA_xxx). To use an additional function used by the loader program, specify the object in the linker configuration file (RZ_T1_init_xxx.icf).

Example: When you add function r_func() (in r_func.c) used in the loader program, add them in appropriate sections as shown the following page.

```
define block LDR_PRG_RBLOCK with fixed order
                            {
                             ro code object startup_Renesas_RZ_T1.o,
                             ro code object system_Renesas_RZ_T1.o,
                             ro code object RZ_T1_RSK_Init.o,
                             ro code object vic.o,
                             ro code object r_atcm_init.o,
                             ro code object r_cpg.o,
                             ro code object r_ram_init.o,
                             ro code object r_mpc.o,
                             ro code object bus_init_nor_boot.o,
                             ro code object r_reset.o,
                                         // A comma is added to the right end of the line at left.
                              ro code object r_func.o  // Added line
 };
```

(2) Assignment of the Sample Programs for e2studio to Sections

Table 5.4 to Table 5.6 list the assignment of the linker script files for each boot mode to sections.

By specifying the following section names with "__attribute__" from the program, the locations of the files can be specified.

Table 5.4 Section Map (SPI Boot Setting) (1 / 2)

Address Range	Area	Description	Load Area	Execution Area
From 0x3000004C	.flash_contents	Label: _mloader_text Label: .=.+ (_loader_text_end - _loader_text_start) • For storage of loader images <Extension> KEEP (the group is not deleted even if there is no reference to it): Check	FLASH	—
From 0x3000604C	.flash_contents_user	Label: _mfvector Label: .=. + (_fvector_end - _fvector_start) • For storage of vector images Label: _mtext Label: .=. + (_text_end - _text_start) • For storage of user code images Label: _mdata Label; .=. + (_data_end - _data_start) • For storage of user data images	FLASH	—
From 0x00000000	.fvectors	Label: _fvectors_start Format: .fvectors Label; _fvectors_end <Extension> Reserve memory for section mapping: Check Start address: Label-_mfvector KEEP (the group is not deleted even if there is no reference to it): Check	—	ATCM
Contiguous following .fvectors	.text	Label: _text_start Format: .text Format: .text.* Label; _text_end <Extension> Reserve memory for section mapping: Check Start address: Label-_mtext	—	ATCM
Contiguous following .text	.init	Format: .init Key word: PROVIDE_HIDDEN (__exidx_start = .) Key word: PROVIDE_HIDDEN (__exidx_end = .)	—	ATCM
Contiguous following .init	.fini	Format: .fini	—	ATCM
Contiguous following .fini	.got	Format: .got Format: .got.plt	—	ATCM
Contiguous following .got	.rodata	Format; .rodata Format: .rodata.* Label: _erodata	—	ATCM
Contiguous following .rodata	.eh_frame_hdr	Format; .eh_fram_hdr	—	ATCM
Contiguous following .eh_frame_hdr	.eh_frame	Format: .eh_frame	—	ATCM
Contiguous following .eh_frame	.jcr	Format: .jcr	—	ATCM

Table 5.4 Section Map (SPI Boot Setting) (2 / 2)

Address Range	Area	Description	Load Area	Execution Area
Contiguous following .jcr	.tors	Label; __CTOR_LIST__ Key word: . = ALIGN(2) Label: __ctors Format: .ctors Label: __ctors_end Label: __CTOR_END__ Label: __DTOR_LIST__ Label: ___dtors Format: .dtors Label: ___dtors_end Label: __DTOR_END__ Key word: . = ALIGN(2)	—	ATCM
From 0x00800000	.loader_param	Label: .loader_param <Extension> Reserve memory for section mapping: Check Start address: Fixed address-0x40000000	—	BTCM
From 0x30000000	For storage of .loader_param		FLASH	—
From 0x00802000	.loader_text	Label: _loader_text_start Format: .loader_text <Extension> Reserve memory for section mapping: Check Start address: Label-_mloader_text	—	BTCM
Contiguous following .loader_text	.loader_text2	Format: .loader_text2 Label: . = . + (512 - ((. - _loader_text_start) % 512)) Label: _loader_text_end	—	BTCM
From 0x00700000	.data	Label: _data_start Format: .data Label: _data_end <Extension> Reserve memory for section mapping: Check Start address: Label-_mdata	—	ACTM
Contiguous following .data	.bss	Key word: PROVIDE(__bss_start__ = .) Label: _bss Format: .bss Format: .bss.** Format: COMMON Key word: PROVIDE(__bss_end__ = .) Label: _ebss Label: _end Key word: PROVIDE(end = .)	—	ACTM
From 0x00807000	.STACK		—	BTCM

Table 5.5 Section Map (NOR Boot Setting) (1 / 2)

Address Range	Area	Description	Load Area	Execution Area
From 0x4000004C	.flash_contents	Label: _mloader_text Label: .=.+ (_loader_text_end - _loader_text_start) • For storage of loader images <Extension> KEEP (the group is not deleted even if there is no reference to it): Check	FLASH	—
From 0x4000604C	.flash_contents_user	Label: _mfvector Label: .=. + (_fvector_end - _fvector_start) • For storage of vector images Label: _mtext Label: .=. + (_text_end - _text_start) • For storage of user code images Label: _mdata Label; .=. + (_data_end - _data_start) • For storage of user data images	FLASH	—
From 0x00000000	.fvectors	Label: _fvectors_start Format: .fvectors Label; _fvectors_end <Extension> Reserve memory for section mapping: Check Start address: Label-_mfvector KEEP (the group is not deleted even if there is no reference to it): Check	—	ATCM
Contiguous following .fvectors	.text	Label: _text_start Format: .text Format: .text.* Label; _text_end <Extension> Reserve memory for section mapping: Check Start address: Label-_mtext	—	ATCM
Contiguous following .text	.init	Format: .init Key word: PROVIDE_HIDDEN (__exidx_start = .) Key word: PROVIDE_HIDDEN (__exidx_end = .)	—	ATCM
Contiguous following .init	.fini	Format: .fini	—	ATCM
Contiguous following .fini	.got	Format: .got Format: .got.plt	—	ATCM
Contiguous following .got	.rodata	Format; .rodata Format: .rodata.* Label: _erodata	—	ATCM
Contiguous following .rodata	.eh_frame_hdr	Format; .eh_fram_hdr	—	ATCM
Contiguous following .eh_frame_hdr	.eh_frame	Format: .eh_frame	—	ATCM
Contiguous following .eh_frame	.jcr	Format: .jcr	—	ATCM
Contiguous following .jcr	.tors	Label; __CTOR_LIST__ Key word: . = ALIGN(2) Label: __ctors Format: .ctors Label: __ctors_end Label: __CTOR_END__ Label: __DTOR_LIST__ Label: ___dtors Format: .dtors Label: ___dtors_end Label: __DTOR_END__ Key word: . = ALIGN(2)	—	ATCM

Table 5.5 Section Map (NOR Boot Setting) (2 / 2)

Address Range	Area	Description	Load Area	Execution Area
From 0x00800000	.loader_param	Label: .loader_param <Extension> Reserve memory for section mapping: Check Start address: Fixed address-0x40000000	—	BTCM
From 0x04000000	For storage of .loader_param		FLASH	—
From 0x00802000	.loader_text	Label: _loader_text_start Format: .loader_text <Extension> Reserve memory for section mapping: Check Start address: Label-_mloader_text	—	BTCM
Contiguous following .loader_text	.loader_text2	Format: .loader_text2 Label: . = . + (512 - ((. - _loader_text_start) % 512)) Label: _loader_text_end	—	BTCM
From 0x00700000	.data	Label: _data_start Format: .data Label: _data_end <Extension> Reserve memory for section mapping: Check Start address: Label-_mdata	—	ACTM
Contiguous following .data	.bss	Key word: PROVIDE(__bss_start__ = .) Label: _bss Format: .bss Format: .bss.** Format: COMMON Key word: PROVIDE(__bss_end__ = .) Label: _ebss Label: _end Key word: PROVIDE(end = .)	—	ACTM
From 0x00807000	.STACK		—	BTCM

Table 5.6 Section Map (RAM Boot Setting) (1 / 2)

Address Range	Area	Description	Load Area	Execution Area
From 0x00000000	.fvectors	Label: _text_start Format: .fvectors Label; _fvectors_end <Extension> KEEP (the group is not deleted even if there is no reference to it): Check	ATCM	Same as on the left
From 0x00000040	.text	Label: _text_start Format: .text Format: .text.*	ATCM	Same as on the left
Contiguous following .text	.init	Format: .init Key word: PROVIDE_HIDDEN (__exidx_start = .) Key word: PROVIDE_HIDDEN (__exidx_end = .)	ATCM	Same as on the left
Contiguous following .init	.fini	Format: .fini	ATCM	Same as on the left
Contiguous following .fini	.got	Format: .got Format: .got.plt	ATCM	Same as on the left
Contiguous following .got	.rodata	Format; .rodata Format: .rodata.* Label: _erodata	ATCM	Same as on the left
Contiguous following .rodata	.eh_frame_hdr	Format; .eh_fram_hdr	ATCM	Same as on the left
Contiguous following .eh_frame_hdr	.eh_frame	Format: .eh_frame	ATCM	Same as on the left
Contiguous following .eh_frame	.jcr	Format: .jcr	ATCM	Same as on the left
Contiguous following .jcr	.tors	Label; __CTOR_LIST__ Key word: . = ALIGN(2) Label: __ctors Format: .ctors Label: __ctors_end Label: __CTOR_END__ Label: __DTOR_LIST__ Label: ___dtors Format: .dtors Label: ___dtors_end Label: __DTOR_END__ Key word: . = ALIGN(2)	ATCM	Same as on the left
Contiguous following .tors	.loader_text	Label: _loader_text_start Format: .loader_text <Extension> Reserve memory for section mapping: Check Start address: Label-_mloader_text	ATCM	Same as on the left
Contiguous following .loader_text	.loader_text2	Format: .loader_text2 Label: . = . + (512 - ((. - _loader_text_start) % 512)) Label: _loader_text_end	ATCM	Same as on the left
Contiguous following .loader_text2	.data	Label: _data_start Format: .data Label: _data_end <Extension> Reserve memory for section mapping: Check Start address: Label-_mdata	ATCM	Same as on the left

Table 5.6 Section Map (RAM Boot Setting) (2 / 2)

Address Range	Area	Description	Load Area	Execution Area
Contiguous following .data	.bss	Key word: PROVIDE(__bss_start__ = .) Label: _bss Format: .bss Format: .bss.** Format: COMMON Key word: PROVIDE(__bss_end__ = .) Label: _ebss Label: _end Key word: PROVIDE(end = .)	—	ATCM
Contiguous following .bss	.SVC_TABLE		—	ATCM
From 0x00807000	.STACK		—	BTCM

(3) Assignment of the Sample Programs for DS-5 to Sections

Table 5.7 to Table 5.8 list the assignment of the linker script files for each boot mode to sections.

By specifying the following section names with "#pragma arm section code" from the program, the locations of the files can be specified.

Table 5.7 Assignment to Sections for DS-5 (SPI/NOR Boot Setting)

Area	Description	Type	Load Area	Execution Area
CONST_LOADER_TABLE	Loader parameters*1	Data	FLASH*1	FLASH
LOADER_RESET_HANDLER	Reset handler area	Code	FLASH*1	FLASH
LOADER_CODE*2	Loader code area	Code	FLASH*1	BTCM
LOADER_CONST*2	Read-only data area for the loader	Data	FLASH*1	BTCM
LOADER_DATA*2	Read/write data area for the loader	Data	FLASH*1	BTCM
LOADER_BSS*2	Loader BSS area	Zero	FLASH*1	BTCM
LOADER_IN_ROOT	Root area for the ARM library	Code Ven Data	FLASH*1	FLASH
INIT	Reset and exception vector table area (for storage)	Code	FLASH*1	ATCM
CODE	Program code area	Code	FLASH*1	ATCM
DATA	Read-only or read/write data area	Data	FLASH*1	ATCM

Note 1. This is assigned to the NOR flash memory for the NOR boot mode version and to the serial flash memory for the SPI boot mode version.

Note 2. The CMSIS-RTOS RTX sample programs require specification of functions (objects) used for the loader program in the loader program sections (LOADER_CODE, LOADER_CONST, LOADER_DATA, and LOADER_BSS). To use an additional function used by the loader program, add it to the program as shown in the example below.

Example: Add function R_ATCM_WaitSet() in appropriate sections.

```
#pragma arm section code = "LOADER_CODE"
#pragma arm section rodata = "LOADER_CONST"
#pragma arm section rwdata = "LOADER_DATA"
#pragma arm section zidata = "LOADER_BSS"
void R_ATCM_WaitSet()
{
}
```

Table 5.8 Assignment to Sections for DS-5 (RAM Boot Setting)

Area	Description	Type	Load Area	Execution Area
VECTORS	Reset and exception vector table area (for storage)	Code	ATCM	Same as on the left
RO_CODE	Program code area	Code	ATCM	Same as on the left
RO_DATA	Read-only data area	Data	ATCM	Same as on the left
LOADER_CODE*1	Program code area	Code	ATCM	Same as on the left
LOADER_CONST*1	Read-only data area	Data	ATCM	Same as on the left
LOADER_DATA*1	Read/write data area	Data	ATCM	Same as on the left
LOADER_BSS*1	BSS area	Zero	ATCM	Same as on the left
RW_DATA	Read/write data area	Data	ATCM	Same as on the left
ZI_DATA	BSS area	Zero	—	ATCM
STACK	STACK area	Zero	—	BTCM
INIT	Dummy area (workaround for build errors)	—	No area as it is only a workaround for build errors (zero size)	

Note 1. Although "LOADER_xxxx" sections are not required in RAM boot mode, these sections are defined since they are common to the source code for SPI boot mode and NOR boot mode.

5.4.2 MPU Settings

The settings for the MPU are described in *Application Note: RZ/T1 Group Initial Settings*.

5.4.3 Exception Vector Table

The RZ/T1 has 7 types of exception processing (reset, undefined instruction, software interrupt, prefetch abort, data abort, IRQ and FIQ exceptions) that are allocated to the 32-byte area starting from address 0000 0000H (address 0000 0000H to 0000 001F). Specify a branch instruction to each exception processing in the exception processing vector table.

The table below lists the contents of exceptional processing vector table for the CMSIS-RTOS RTX sample programs.

Table 5.9 Exception Processing Vector Table for the RZ/T1

Exception	Handler Address	Remark
RESET exception	0000 0000H	Branches to the start of the loader program.
Undefined instruction exception	0000 0004H	Undefined instruction exception handler for CMSIS-RTOS RTX
Software interrupt exception	0000 0008H	SVC exception handler for CMSIS-RTOS RTX
Prefetch abort exception	0000 000CH	Prefetch abort handler for CMSIS-RTOS RTX
Data abort exception	0000 0010H	Data abort handler for CMSIS-RTOS RTX
Reserved	0000 0014H	NOP
IRQ exception	0000 0018H	IRQ exception handler for CMSIS-RTOS RTX (not used)
FIQ exception	0000 001CH	Dummy routine (endless loop)

The RZ/T1 adopts the vector interrupt controller (VIC) to control interrupts for Cortex-R4F and the VIC driver manages interrupts.

For the interrupts to be used, call function InterruptHandlerRegister with the vector number and interrupt service routine specified to register the interrupt service routine for each generated interrupt.

5.4.4 Memory Size

(1) Memory Used by the Sample Program for EWARM

For reference, the table below lists the amounts of memory used by the RTX_ex1 sample program in the case of the EWARM IDE.

Table 5.10 Amounts of Memory Used in the Case of EWARM (RTX_ex1_NOR, RTX_ex1_SPIBSC)

		Memory Size (bytes) (approx.)	
Label	**Outline**	**NOR**	**SPIBSC**
VECTOR_WBLOCK	Reset and exception vector table area (for execution)	60	60
USER_PRG_WBLOCK	Sample application program area (for execution)	16 K	11 K
.noinit	Stack area	1280	1280
USER_DATA_WBLOCK	Sample application program variable area (for execution)	16	16
USER_DATA_ZBLOCK	Uninitialized data area	59 K	59 K
LDR_DATA_WBLOCK	Loader program variable area (for execution	16	16
LDR_PRG_WBLOCK	Loader program area (for execution)	4 K	16 K
ldr_param	Loader parameters	76	76
LDR_PRG_RBLOCK	Loader program area (for storage)	4 K	16 K
LDR_DATA_RBLOCK	Loader program variable area (for storage)	16	16
VECTOR_RBLOCK	Reset and exception vector table area (for storage)	60	60
USER_PRG_RBLOCK	Sample application program area (for storage)	16 K	11 K
USER_DATA_RBLOCK	Sample application program variable area (for storage)	16	16

Note: The value depends on the sample.

Table 5.11 Amounts of Memory Used in the Case of EWARM (RTX_ex1_RAM)

Label	Outline	Memory Size (bytes) (approx.)
intvec	Reset and exception vector table area	60
ROM_region	Program, constant area	19 K
RAM_region	Uninitialized data area, data area	60 K

Note: The value depends on the sample.

(2) Memory Used by the Sample Program for e2studio

For reference, the table below lists the amounts of memory used by the RTX_ex1 sample program in the case of the e2studio IDE.

Table 5.12 Amounts of Memory Used in the Case of e2studio (RTX_ex1_RAM, RTX_ex1_NOR, RTX_ex1_SPIBSC)

		Memory Size (bytes) (approx.)		
Label	**Outline**	**NOR**	**SPIBSC**	**RAM**
.flash_contents	Loader preprocessor code area (for storage)	6 K	19 K	—
.flash_content_user	Sample application program area, sample application data area, sample application constant area, reset and exception vector table (for storage)	48 K	48 K	—
.fvectors	Reset and exception vector table (for execution)	60	60	60
.text	Sample application program area (for execution)	48 K	48 K	49 K
.rodata	Sample application constant area (for execution)	168	168	168
.tors		0	0	0
.loader_param	Loader parameters	76	76	—
.loader_text	Loader preprocessor code area (for execution)	1104	1104	—
.loader_text2	Loader postprocessor code area (for execution)	5 K	17 K	2 K
.data	Sample application data area (for execution)	32	32	32
.bss	Uninitialized data area	59 K	59 K	59 K
.SVC_TABLE	SVC table	8	8	8
.STACK	Stack area for each mode	1280	1280	1280

Note: The value depends on the sample.

(3) Memory Used by the Sample Program for DS-5

For reference, the table below lists the amounts of memory used by the RTX_ex1 sample program in the case of the DS-5 IDE.

Table 5.13 Amounts of Memory Used in the Case of DS-5 (RTX_ex1_NOR, RTX_ex1_SPIBSC)

		Memory Size (bytes) (approx.)	
Label	**Outline**	**NOR**	**SPIBSC**
CONST_LOADER_TABLE	Loader parameters	76	76
LOADER_RESET_HANDLER	Reset handler area	28	28
LOADER_CODE	Loader code area	236	236
LOADER_CONST	Read-only data area for the loader	0	0
LOADER_DATA	Read/write data area for the loader	20	20
LOADER_BSS	Loader BSS area	1204	1308
LOADER_IN_ROOT	Root area for the ARM library	236	236
INIT	Reset and exception vector table area (for storage)	60	60
CODE	Program code area	25 K	25 K
DATA	Read-only or read/write data area	68 K	68 K
SVC_TABLE	SVC table	8	8
STACK	Stack area for each mode	1280	1280

Note: The value depends on the sample.

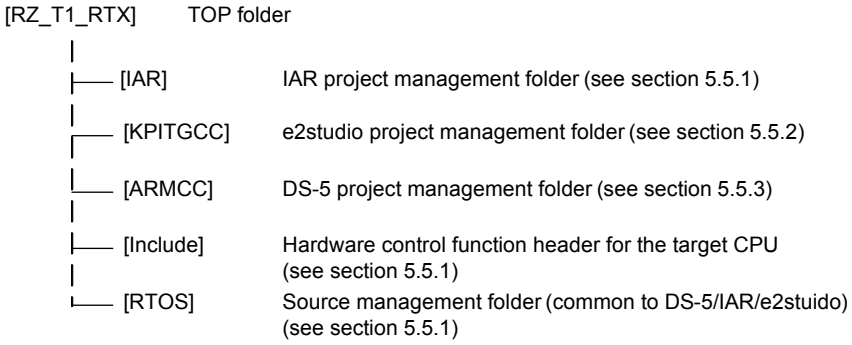
Table 5.14 Amounts of Memory Used in the Case of DS-5 (RTX_ex1_RAM)

		Memory Size (bytes) (approx.)
Label	**Outline**	**RAM**
LOADER_CODE	Loader code area	1260
LOADER_CONST	Read-only data area for the loader	0
LOADER_DATA	Read/write data area for the loader	12
LOADER_BSS	Loader BSS area	0
VECTORS	Reset and exception vector table area (for storage)	60
RO_CODE	Program code area	22 K
RO_DATA	Read-only data area	108
RW_DATA	Read/write data area with initial values	128
ZI_DATA	Read/write data area without initial values	69 K
SVC_TABLE	SVC table	8
STACK	Stack area for each mode	1280

Note: The value depends on the sample.

5.5 Folder and File Structure

The top folder structure is as shown below. Strings enclosed by [ ] represent folder names. The details are described in each section.



5.5.1 Folder and File Structure of the Sample Applications for EWARM

(1) Folder Structure

The folders of the sample applications for EWARM are in three folders below the top folder, [RZ_T1_RTX]: [IAR], [Include], and [RTOS].

Figure 5.4 shows the directory tree for the [IAR] folder and its subfolders. The only files under the [IAR] folder are project management files and these all become project management folders for EWARM (source code is not stored).

The folder names with the suffix […_***] below indicate three folders, […_RAM] for RAM booting, […_NOR] for NOR booting, and […_SPIBSC] for SPI booting.

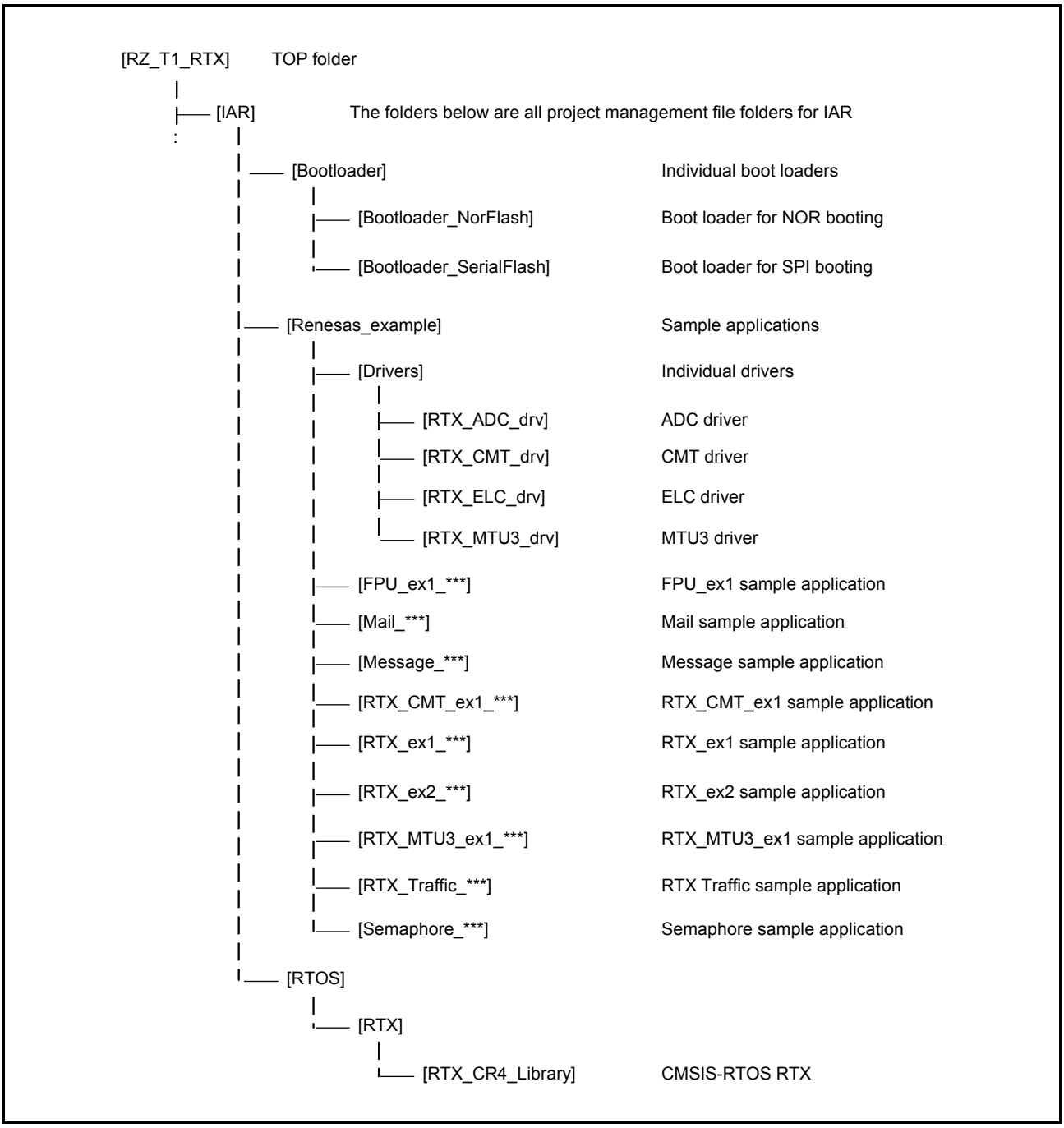


Figure 5.4 [IAR] Folder Structure

Next, Figure 5.5 describes the [include] and [RTOS] folders. These two folders hold the programs. The contents are described next to the names.

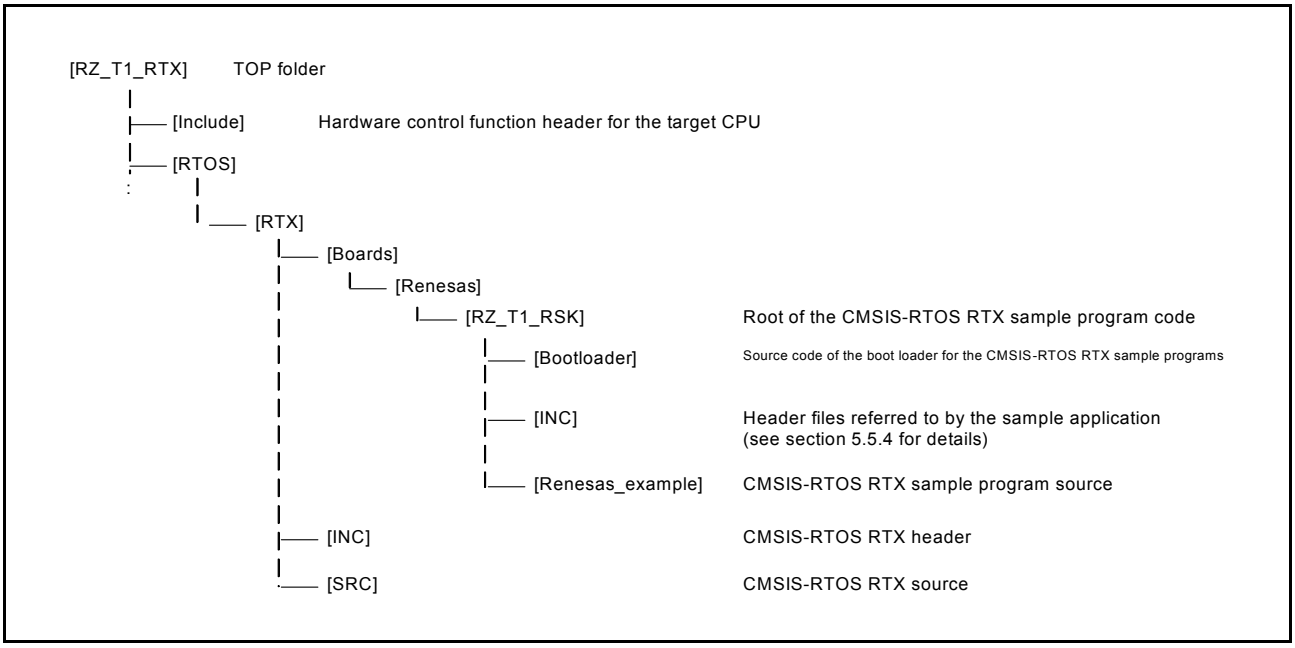


Figure 5.5 [Include] and [RTOS] Folders

(2) File Structure

The table below lists the libraries to which the CMSIS-RTOS RTX sample programs are linked.

Table 5.15 Libraries to which the Sample Applications are Linked

File	Outline	Remark
RTX_CR4_Library.a	CMSIS-RTOS RTX library	A library for little endian for Cortex-R4F
Bootloader_NorFlash.a	Library for NOR booting	A library to start the sample applications in NOR boot mode (from NOR flash memory). It is linked by the _NOR project.
Bootloader_SerialFlash.a	Library for SPI booting	A library to start the sample applications in SPI boot mode (from serial flash memory). It is linked by the _SPIBSC project.
RTX_ADC_drv.a	RTX_ADC driver library	A library for the RTX_ADC driver. It is linked by the RTX_CMT_ex1 sample project.
RTX_CMT_drv.a	RTX_CMT driver library	A library for the RTX_CMT driver. It is linked by the RTX_CMT_ex1 sample project.
RTX_ELC_drv.a	RTX_ELC driver library	A library for the RTX_ELC driver. It is linked by the RTX_CMT_ex1 sample project.
RTX_MTU3_drv.a	RTX_MTU3 driver library	A library for the RTX_MTU3 driver. It is linked by the RTX_MTU3_ex1 sample project.

The table below lists the common source files for the CMSIS-RTOS RTX sample programs.

Table 5.16 Common Source Files for the Sample Applications

File	Outline	Remark
close.c	Closes a file.	IAR C/C++ compiler file I/O
lseek.c	Sets the file position indicator.	IAR C/C++ compiler file I/O
read.c	Reads a character buffer.	IAR C/C++ compiler file I/O
write.c	Writes to a character buffer.	IAR C/C++ compiler file I/O
fpuenable.s	Floating-point unit (FPU) setting source	
icu_vic.s	Cortex-R4F vector interrupt controller (VIC) control source	
startup_Renesas_RZ_T1.s	Exception handling vectors (reset handlers)	
vector.s	Vector addresses	
RAM\system_Renesas_RZ_T1.c	Hardware initialization source	

The above files are included in any of the following paths.

RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example\(*Dir1)\IAR
RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example\Drivers\drv_example\(*Dir2)\IAR
*Dir1: FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore
*Dir2: RTX_CMT_ex1 / RTX_MTU3_ex1

5.5.2 Folder and File Structure of the Sample Applications for e2studio

(1) Folder Structure

The folders of the sample applications for e2studio are in three folders below the top folder, [RZ_T1_RTX]: [KPITGCC], [Include], and [RTOS].

The structure of the folder is the same as that in Figure 5.4 and Figure 5.5 in Section 5.5.1. When you view it, simply read IAR as KPITGCC and EWARM as e2studio.

(2) File Structure

The libraries to which the CMSIS-RTOS RTX sample programs are linked are the same as those in Table 5.15. See this table for reference.

The table below lists the common source files for the CMSIS-RTOS RTX sample programs.

Table 5.17 Common Source Files for the Sample Applications

File	Outline	Remark
syscalls.c	_read, _write, and _exit functions	GCC compiler file I/O
exclusion_func.c	Exclusive control function	GCC compiler file I/O
siorw.c	Serial port I/O function	GCC compiler file I/O
newheap.c	Heap end address acquisition function	GCC compiler file I/O
fpuenable.S	Floating-point unit (FPU) setting source	
icu_vic.S	Cortex-R4F vector interrupt controller (VIC) control source	
startup_Renesas_RZ_T1.S	Exception handling vectors (reset handlers)	
vector.S	Vector addresses	
RAM\system_Renesas_RZ_T1.c	Hardware initialization source	

The above files are included in any of the following paths.

RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example\(*Dir1)\GCC
RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example\Drivers\drv_example\(*Dir2)\GCC
*Dir1: FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore
*Dir2: RTX_CMT_ex1 / RTX_MTU3_ex1

5.5.3 Folder and File Structure of the Sample Applications for DS-5

(1) Folder Structure

The folders of the sample applications for DS-5 are in three folders below the top folder, [RZ_T1_RTX]: [ARMCC], [Include], and [RTOS].

The structure of the folder is the same as that in Figure 5.4 and Figure 5.5 in Section 5.5.1. When you view it, simply read IAR as ARMCC and EWARM as DS-5.

Caution: The directory trees of the following two folders should not be changed because the included fmtool only runs with this tree structure.

\ARMCC\Bootloader\Bootloader_NorFlash

\ ARMCC\Bootloader\Bootloader_SerialFlash

Supplementary information on the files for use in flash programming is given below.

Table 5.18 Files for Use in Flash Programming in NOR Boot Mode

File	Outline	Remark
\ARMCC\Bootloader\Bootloader_NorFlash\ fmtool	fmtool of the sample NOR flash programming tool	This folder and files should not be changed because the included fmtool only runs with this folder structure.
\ARMCC\Bootloader\Bootloader_NorFlash\ script_nor	Script file for sample downloading	

Table 5.19 Files for Use in Flash Programming in SPI Boot Mode

File	Outline	Remark
\ ARMCC\Bootloader\Bootloader_ SerialFlash\fmtool	fmtool of the sample Serial flash programming tool	This folder and files should not be changed because the included fmtool only runs with this folder structure.
\ ARMCC\Bootloader\Bootloader_ SerialFlash\script_nor	Script file for sample downloading	

(2) File Structure

The libraries to which the CMSIS-RTOS RTX sample programs are linked are the same as those in Table 5.15. See this table for reference.

The table below lists the common source files for the CMSIS-RTOS RTX sample programs.

Table 5.20 Common Source Files for the Sample Applications

File	Outline	Remark
fpuenable.s	Floating-point unit (FPU) setting source	ARM C/C++ compiler file I/O
read.c	Reads a character buffer.	ARM C/C++ compiler
write.c	Writes to a character buffer.	ARM C/C++ compiler
retarget.c	ARM C library target dependent I/O support function	ARM C/C++ compiler
sio_char.h	Character function related header	ARM C/C++ compiler
icu_vic.s	Cortex-R4F vector interrupt controller (VIC) control source	
startup_Renesas_RZ_T1.s	Exception handling vectors (reset handlers)	
vector.s	Vector addresses	
RAM\system_Renesas_RZ_T1.c	Hardware initialization source	

The above files are included in any of the following paths.

RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example\(*Dir1)\ARM
RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example\Drivers\drv_example\(*Dir2)\ARM
*Dir1: FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore
*Dir2: RTX_CMT_ex1 / RTX_MTU3_ex1

5.5.4 Common Folder Structure of the Sample Applications

The table below lists the header files referred to by the CMSIS-RTOS RTX sample programs.

Table 5.21 Header Files Referred to by the Sample Applications

File	Outline	Remark
iodefine.h	RZ/T1 definitions header (used by all other include files)	
r_typedefs.h	Base data type header	
Renesas_RZ_T1.h	RZ_T1 CPU-dependent definition	
RZ_T1_RSK_Init.h	RZ/T1 evaluation board initialization header	
system_Renesas_RZ_T1.h	CPU-dependent part initialization function header	
vic.h	Interrupt controller setting API header	
r_atcm_init.h	ATCM wait setting API header	
r_bsc.h	Bus controller setting API header	
r_cpg.h	Clock pulse generator (CPG) setting API header	
r_ecm.h	Error control module (ECM) API header	
r_icu_init.h	Interrupt controller initialization and API header	
r_mpc.h	Multi-function pin controller (MPC) setting API	
r_port.h	I/O port setting API header	
r_ram_init.h	Internal extended SRAM (area 1, area 2) setting API header	
r_reset.h	Reset API header	
r_spibsc_flash_api.h	Serial flash setting API header	
r_spibsc_ioset_api.h	Serial flash IO setting API header	
r_system.h	Hardware initialization header	

The above files are located in the following directory. They are used in common by all projects.

RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\INC

The table below lists the common source files for the CMSIS-RTOS RTX sample programs.

Table 5.22 Common Source Files for the Sample Applications

File	Outline	Remark
low_level_init.c	Low-level initialization function source	
loader_reset_handler.s	NOR/SPI booting reset handler	RZ/T1, RZ/T1 evaluation board
r_atcm_init.c	ATCM wait setting source	RZ/T1, RZ/T1 evaluation board
r_cpg.c	Clock pulse generator (CPG) setting source	RZ/T1, RZ/T1 evaluation board
r_ecm.c	Error control module setting source	RZ/T1, RZ/T1 evaluation board
r_mpc.c	Multi-pin function setting source	RZ/T1, RZ/T1 evaluation board
r_reset.c	Reset control source	RZ/T1, RZ/T1 evaluation board
RTX_Conf_CM.c	CMSIS-RTOS RTX configuration	RZ/T1, RZ/T1 evaluation board
RZ_T1_RSK_Init.c	RZ/T1 evaluation board initialization source	RZ/T1, RZ/T1 evaluation board
Serial.h	Debug serial I/O header	RZ/T1, RZ/T1 evaluation board
Serial.c	Debug serial I/O source	RZ/T1, RZ/T1 evaluation board
vic.c	Cortex-R4F vector interrupt controller (VIC) setting API source	RZ/T1 (Cortex-R4F), RZ/T1 evaluation board

The above files are included in any of the following paths.

RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example\(*Dir1)
*Dir1: FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore

The table below lists the common files for the actual CMSIS-RTOS RTX sample programs.

Table 5.23 Files for the Actual Sample Applications

File	Outline
FPU_ex1\FPU_ex1.c	Actual FPU ex1 sample application
Mail\Mail.c	Actual Mail sample application
Message\Message.c	Actual Message sample application
RTX_ex1\RTX_ex1.c	Actual RTX ex1 sample application
RTX_ex2\RTX_ex2.c	Actual RTX ex2 sample application
Semaphore\Semaphore.c	Actual Semaphore sample application
Drivers\drv_example\RTX_CMT_ex1\CMT_ex1.c	Actual RTX_CMT ex1 sample application
Drivers\drv_example\RTX_MTU3_ex1 \MTU_ex1.c	Actual RTX_MTU3 ex1 sample application

The path to the above directory is as follows.

RZ_T1_RTX\RTOS\RTX\Boards\Renesas\RZ_T1_RSK\Renesas_example

5.6 Base Data Types

The table below lists the base data types used in the CMSIS-RTOS RTX sample programs.

These CMSIS-RTOS RTX sample programs do not use C-language types directly as base data types but use the types listed below.

Table 5.24 Base Data Types

typedef char	**char_t**
typedef int	int_t
typedef unsigned int	bool_t
typedef signed char	int8_t
typedef unsigned char	uint8_t
typedef signed short	int16_t
typedef unsigned short	uint16_t
typedef signed int	int32_t
typedef unsigned int	uint32_t
typedef signed long long	int64_t
typedef unsigned long long	uint64_t

5.7 Macro Definitions for Use in Configuration

The table below lists the constant macros for use in configuration.

Table 5.25 Constant Macros for Use in Configuration

Macro	Value	Description	Definition File
__FPU_PRESENT	Defined without values	Use of the FPU When defined: To be used; when not defined: Not to be used	Project settings
OS_TASKCNT	50	Maximum number of threads generated by the CMSIS-RTOS RTX Note: The upper limit for the system is 250.	RTX_Conf_CM.c
OS_STKSIZE	256	Default stack size for the CMSIS-RTOS RTX threads (unit: 4 bytes)	RTX_Conf_CM.c
OS_MAINSTKSIZE	256	Stack size for the first CMSIS-RTOS RTX thread (unit: 4 bytes)	RTX_Conf_CM.c
OS_PRIVCNT	0	Number of threads with user-defined stacks for CMSIS-RTOS RTX	RTX_Conf_CM.c
OS_PRIVSTKSIZE	500	Total size of user-defined stacks for CMSIS-RTOS RTX	RTX_Conf_CM.c
OS_STKCHECK	1	CMSIS-RTOS RTX stack overflow checking 1: Enabled; 0: Disabled	RTX_Conf_CM.c
OS_RUNPRIV	1	Operating mode of the CPU when a thread of the CMSIS-RTOS RTX is run 1: Privileged; 0: Non-privileged	RTX_Conf_CM.c
OS_SYSTICK	0	Selection of the system timer for CMSIS-RTOS RTX 0: Peripheral timer; 1: Internal timer of the CPU core Note: The value is fixed to 0 as the RZ/T1 does not have the CPU core internal timer.	RTX_Conf_CM.c
OS_CLOCK	2343750 (75000000u/32u)	CMSIS-RTOS RTX system timer input clock (unit: Hz) Note: There is no reference to the value when OS_SYSTICK is 1.	RTX_Conf_CM.c
OS_TICK	1000	CMSIS-RTOS RTX system timer cycle (unit: µsec) Note: The example setting is 1 ms.	RTX_Conf_CM.c
OS_ROBIN	0	CMSIS-RTOS RTX round-robin function 1: Enabled; 0: Disabled	RTX_Conf_CM.c
OS_ROBINTOUT	5	CMSIS-RTOS RTX round-robin timeout (unit: system timer tick)	RTX_Conf_CM.c
OS_TIMERS	1	CMSIS-RTOS RTX user timer function 1: Enabled; 0: Disabled Note: When the setting is 1, a "user timer callback function execution thread" is generated.	RTX_Conf_CM.c
OS_TIMERPRIO	5	Priority level of the user timer callback function execution thread of the CMSIS-RTOS RTX (from minimum 1 to maximum 6)	RTX_Conf_CM.c
OS_TIMERSTKSZ	256	Stack size for the user timer callback function execution thread of the CMSIS-RTOS RTX (unit: 4 bytes)	RTX_Conf_CM.c
OS_TIMERCBQS	5	Allowable number of timeout notification message queues for the user timer callback function execution thread of the CMSIS-RTOS RTX	RTX_Conf_CM.c
OS_FIFOSZ	16	Number of FIFO queue elements for isr_* type API events of the CMSIS-RTOS RTX	RTX_Conf_CM.c
OS_MUTEXCNT	8	Number of definitions of exclusive mutexes for multiple threads for standard C library functions	RTX_Conf_CM.c
OS_TRV	The value is shown on the right.	Compare register setting for the CPU core internal timer Note: There is no reference to the value when OS_SYSTICK is 1. This is defined below. ((uint32_t)(((double)OS_CLOCK*(double)OS_TICK)/1E6)-1)	RTX_Conf_CM.c

5.8 Error Codes

The CMSIS-RTOS RTX returns the following values as error codes of the API.

Table 5.26 CMSIS-RTOS RTX Error Codes

Enumerator	Value	Description	Definition File
osOK	0x00	Normal termination of the API	cmsis_os.h
osEventSignal	0x08	Signal event occurred.	cmsis_os.h
osEventMessage	0x10	Message event occurred.	cmsis_os.h
osEventMail	0x20	Mail event occurred.	cmsis_os.h
osEventTimeout	0x40	Timeout occurred.	cmsis_os.h
osErrorParameter	0x80	Parameter error	cmsis_os.h
osErrorResource	0x81	Resource not available	cmsis_os.h
osErrorTimeoutResource	0xC1	Resource not available within the timeout period	cmsis_os.h
osErrorISR	0x82	The API cannot be called from interrupt service routines (ISR).	cmsis_os.h
osErrorISRRecursive	0x83	The API was called multiple times from ISR with the same object.	cmsis_os.h
osErrorPriority	0x84	The system cannot determine priority or specified an illegal priority.	cmsis_os.h
osErrorNoMemory	0x85	Memory allocation failed	cmsis_os.h
osErrorValue	0x86	The value of a parameter is out of range.	cmsis_os.h
osErrorOS	0xFF	Unspecified CMSIS-RTOS RTX error: run-time error but no other message fits.	cmsis_os.h

5.9 Functions

The table below lists the internal functions which include processing that is dependent on the RZ/T1 evaluation board.

Table 5.27 RZ/T1 Evaluation Board Dependent Internal Functions (1 / 2)

Function	Outline	Implementation File
Reset_Handler	This is an assembler function for reset vector allocation for ICCARM and RenesasGCC. After SystemInit() has been executed, execution jumps to either of the following functions. • **GCC compiler: cstartup()** • **IAR compiler: __cmain()**	startup_Renesas_RZ_T1.s
user_init	This is an assembler function for reset vector allocation for ARMCC. After SystemInit() has been executed, execution jumps to function __main.	startup_Renesas_RZ_T1.s
cpu_init	This function is for ARMCC and NOR or SPI booting. It handles initialization of the clock signals and bus, etc., and then causes a jump to function user_init.	loader_reset_handler.s
Undef_Handler	Undefined instruction exception handler	startup_Renesas_RZ_T1.s
PAbt_Handler	Software exception handler	startup_Renesas_RZ_T1.s
DAbt_Handler	Data-abort exception handler	startup_Renesas_RZ_T1.s
CUndefHandler	Data-abort exception handler	system_Renesas_RZ_T1.c
CPAbtHandler	Prefetch-abort exception handler	system_Renesas_RZ_T1.c
CDAbtHandler	C handler for the data-abort exception	system_Renesas_RZ_T1.c
__SVC_1	Initializing the caches	system_Renesas_RZ_T1.c
__cmain **Note: For the ICCARM compiler only**	This is a C-language function to be called from Reset_Handler(). Initialization for the C library and startup of the OS	RTX_CM_lib.h
cstartup **Note: For the RenesasGCC compiler only**	This is a C-language function to be called from Reset_Handler(). Initialization for the C library and startup of the OS	RTX_CM_lib.h
SystemInit	This is a C-language function to be called from Reset_Handler(). Board-dependent hardware initialization	system_Renesas_RZ_T1.c
submain	Registering the interrupts to be used and initializing the caches	system_Renesas_RZ_T1.c
__low_level_init **Note: For the ICCARM compiler only**	Low-level initialization function	low_level_init.c
R_ATCM_WaitSet	ATCM wait setting function	r_atcm_init.c
R_CPG_WriteEnable	Enabling writing to the clock generator related registers	r_cpg.c
R_CPG_WriteDisable	Disabling writing to the clock generator related registers.	r_cpg.c
R_CPG_PLL_Wait	Processing to wait for PLL stabilization	r_cpg.c
R_ECM_Init	Initializing the error control module	r_ecm.c
R_ECM_CompareError_Wait	Processing to cause a 15-µs wait for generation of the compare-match timer interrupt.	r_ecm.c
R_ECM_Write_Reg8	8-bit writing to the error control module register	r_ecm.c
R_ECM_Write_Reg32	32-bit writing to the error control module register	r_ecm.c
R_MPC_WriteEnable	Enabling writing to the multi-pin function related registers	r_mpc.c
R_MPC_WriteDisable	Disabling writing to the multi-pin function related registers	r_mpc.c
R_RST_WriteEnable	Enabling writing to the reset related registers	r_reset.c
R_RST_WriteDisable	Disabling writing to the reset related registers	r_reset.c
reset_check	Processing for judging the reset flag	RZ_T1_RSK_Init.c
cpg_init	Initializing the clock pulse generator (CPG)	RZ_T1_RSK_Init.c
copy_to_atcm	Copying the AP from the external flash memory to the ATCM	RZ_T1_RSK_Init.c
SER_Init	Initializing the debug serial port	Serial.c

Table 5.27 RZ/T1 Evaluation Board Dependent Internal Functions (2 / 2)

Function	Outline	Implementation File
SER_Enable	Enabling the debug serial port	Serial.c
SER_Disable	Disabling the debug serial port	Serial.c
SER_Set_baud_rate	Setting the bit rate for the debug serial port	Serial.c
SER_PutChar	Output of a single character code to the debug serial port	Serial.c
SER_GetChar	Input of a single character code from the debug serial port	Serial.c
interrupt_SER	Debug serial interrupt handler (not used)	Serial.c
FPUEnable	Enabling the floating-point unit	fpuenable.s
vic_isr_001 to vic_isr_300	Vector interrupt (from VIC1 to VIC300)	icu_vic.s
vic_isr	Processing to judge vector interrupts	icu_vic.s
act_irq	Actual interrupt processing	icu_vic.s
ret_irq	Post-processing for interrupts	icu_vic.s

The table below lists the IAR C/C++ compiler dependent internal functions.

Table 5.28 IAR C/C++ Compiler Dependent Internal Functions

Function	Outline	Implementation File
__read	Reading a character buffer	read.c
__write	Writing to a character buffer	write.c
__lseek	Seeking within a file	lseek.c
__close	Closing a file	close.c

The table below lists the RenesasGCC compiler dependent internal functions.

Table 5.29 GNU C/C++ Compiler Dependent Internal Functions

Function	Outline	Implementation File
_read	Reading a character buffer	syscalls.c
_write	Writing to a character buffer	syscalls.c
_exit	Ending the program	syscalls.c
SioRead	Output of characters to the SCIF	siorw.c
SioWrite	Input of characters from the SCIF	siorw.c
_top_of_heap	Heap end address acquisition function	newheap.c
__enable_irq	Enabling IRQ interrupts	exclusion_func.c
__disable_irq	Disabling IRQ interrupts	exclusion_func.c
__strex	Exclusive register store instruction	exclusion_func.c
__clrex	Instruction for removing exclusivity	exclusion_func.c
__ldrex	Exclusive register load instruction	exclusion_func.c

The table below lists the ARM C/C++ compiler dependent internal functions.

Table 5.30 ARM C/C++ Compiler Dependent Internal Functions

Function	Outline	Implementation File
__rt_entry	Initializing the ARM C library Note: This function is within the CMSIS-RTOS RTX header and does not require customization.	RTX_CM_lib.h
_user_perthread_libspace	Acquiring the local data area for the ARM C library threads Note: This function is within the CMSIS-RTOS RTX header and does not require customization.	RTX_CM_lib.h
_mutex_initialize	Generating resources for mutexes of the ARM C library Note: This function is within the CMSIS-RTOS RTX header and does not require customization.	RTX_CM_lib.h
_mutex_acquire	Acquiring locking by mutexes of the ARM C library Note: This function is within the CMSIS-RTOS RTX header and does not require customization.	RTX_CM_lib.h
_mutex_release	Releasing locking by mutexes of the ARM C library Note: This function is within the CMSIS-RTOS RTX header and does not require customization.	RTX_CM_lib.h
fgetc	Reading a single character from a file	retarget.c
fputc	Writing a single character to a file	retarget.c
ferror	Detecting the error state of the file	retarget.c
SioRead	Output of characters to the SCIF	read.c
SioWrite	Input of characters from the SCIF	write.c
_ttywrch	Output of a character to a terminal by using the ARM C library	retarget.c
_sys_exit	Ending the ARM C library	retarget.c

The table below lists the internal functions which include CMSIS-RTOS RTX dependent processing (these include processing that is dependent on the RZ/T1 evaluation board).

Table 5.31 CMSIS-RTOS RTX Dependent Internal Functions

Function	Outline	Implementation File
os_idle_demon	Idle thread for the CMSIS-RTOS RTX	RTX_Conf_CM.c
os_tick_init	Initializing the peripheral timer for CMSIS-RTOS RTX	RTX_Conf_CM.c
os_tick_val	Acquiring the current value of the hardware timer for CMSIS-RTOS RTX	RTX_Conf_CM.c
os_tick_ovf	Handles indication of whether the hardware timer for CMSIS-RTOS RTX has overflowed	RTX_Conf_CM.c
os_tick_irqack	Clearing the state of interrupts from the CMSIS-RTOS RTX peripheral timer	RTX_Conf_CM.c
os_error	Causing the system to hang when a runtime error occurs in the CMSIS-RTOS RTX	RTX_Conf_CM.c
OS_Tick_Handler	ISR handler for the CMSIS-RTOS RTX peripheral timer	HAL_CR4_asm.s
os_pendsv_init	Initializing the PendSV interrupt for the CMSIS-RTOS RTX	RTX_Conf_CM.c
os_pendsv_irqack	Clearing the state of the PendSV interrupt for the CMSIS-RTOS RTX	RTX_Conf_CM.c
os_pendsv	PendSV interrupt handler processing for the CMSIS-RTOS RTX	RTX_Conf_CM.c

The table below lists the functions which are available from the drivers and user application after initializing the system.

Table 5.32 Utility Functions

Function	Outline	Implementation File
RZ_T1_RSK_InitClock	Setting the internal clock multiplication factor	RZ_T1_RSK_Init.c
InterruptHandlerRegister	Registering an IRQ handler	system_Renesas_RZ_T1.c
InterruptHandlerUnregister	Deregistering an IRQ handler	system_Renesas_RZ_T1.c

The table below lists the functions for operations related to the Cortex-R4F vector interrupts.

Table 5.33 Functions for Operations Related to the Vector Interrupts

Function	Outline	Implementation File
VIC_EnableIRQ	Enabling the vector interrupt for Cortex-R4F	vic.c
VIC_DisableIRQ	Disabling the vector interrupt for Cortex-R4F	vic.c
VIC_SetDetectType	Setting the type of detection for interrupts for Cortex-R4F	vic.c
VIC_SetPriority	Setting the priority level of the vector interrupt for Cortex-R4F	vic.c
VIC_GetPriority	Acquiring the priority level of the vector interrupt for Cortex-R4F	vic.c
VIC_Init	Initializing the vector interrupt controller (VIC) for Cortex-R4F	vic.c
VIC_Enable	Enabling the vector interrupt controller (VIC) for Cortex-R4F	vic.c
VIC_ClearPIC	Deasserting interrupt signals	vic.c
VIC_EndInterrupt	Ending interrupts	vic.c

6. Specifications of the Functions

6.1 RZ/T1 Evaluation Board Dependent Internal Functions

6.1.1 Reset_Handler

Reset_Handler	RZ/T1 evaluation board dependent internal function
Reset entry (for ICCARM and Renesas GCC)	Handler

Header	None
Declaration	Reset_Handler .global Reset_Handler .func Reset_Handler
Arguments	None
Return values	None
Caller	The code starts from the interrupt vector address so is automatically executed on release from the reset state.
Description	Assembler code This code handles processing for initialization in the following order. • Initializing the operating mode for CP15 (disabling caches, virtual memory, and branch prediction) • Setting a stack for each type of exception (allocating individual stacks) • Copying images of the required ROM sections to RAM sections (processing depends on the compiler) • Executing function SystemInit (to initialize the platform-dependent peripherals) • Jumping to function __cmain (for ICCARM) or to function cstartup (for RenesasGCC)
Remarks	The code is for use with ICCARM and RenesasGCC (the details of processing differ slightly).

6.1.2 user_init

user_init	RZ/T1 evaluation board dependent internal function
Reset entry (for ARMCC)	Handler

Header	None
Declaration	user_init
Arguments	None
Return values	None
Caller	In RAM boot mode: cpu_init; In NOR or SPI boot mode: The code is automatically executed on release from the reset state.
Description	Assembler function This code handles processing for initialization in the following order. • Initializing the operating mode for CP15 (disabling caches, virtual memory, and branch prediction) • Setting a stack for each type of exception (allocating individual stacks) • Copying images of the required ROM sections to RAM sections • Executing function SystemInit (to initialize the platform-dependent peripherals) • Jumping to function __main
Remarks	This code is for ARMCC.

6.1.3 cpu_init

cpu_init	RZ/T1 evaluation board dependent internal function
Reset entry (for ARMCC)	Handler

Header	None
Declaration	user_init
Arguments	None
Return values	None
Caller	In NOR or SPI boot mode: The code is automatically executed on release from the reset state. It is not executed in RAM boot mode.
Description	Assembler function This code handles processing for initialization in the following order. • Initializing the clock signals and bus • Copying images of the required ROM sections to RAM sections • Initializing the error control module (ECM) • Jumping to function user_init
Remarks	This code is for ARMCC. It is only used in NOR or SPI boot mode.

6.1.4 Undef_Handler

Undef_Handler	RZ/T1 evaluation board dependent internal function
Handler for undefined instruction exceptions	Handler

Header	None
Declaration	Undef_Handler
Arguments	None
Return values	None
Caller	Undefined instruction generation code
Description	Handler for undefined instruction exceptions Assembler handler • Obtain the operating mode at the location where the undefined instruction exception occurred (ARM mode, Thumb mode) • Obtain the opcode at the location where the undefined instruction exception occurred. • Obtain the location of the return destination where the exception occurred (the value of LR (link register)). • Call function CUndefHandler with the above three values as arguments.
Remarks	This function is present for each compiler. See Section 6.1.7 for details of function CUndefHandler.

6.1.5 PAbt_Handler

PAbt_Handler	RZ/T1 evaluation board dependent internal function
Prefetch-abort exception handler	Handler

Header	None
Declaration	PAbt_Handler
Arguments	None
Return values	None
Caller	Prefetch-abort exception generation code
Description	Handler for the prefetch-abort exception Assembler handler • Obtain the instruction fault status register. • Obtain the instruction fault address register. • Obtain the location of the return destination where the exception occurred (the value of LR (link register)). • Call function CPAbtHandler with the above three values as arguments.
Remarks	This function is present for each compiler. See Section 6.1.8 for details of CPAbtHandler.

6.1.6 DAbt_Handler

DAbt_Handler	RZ/T1 evaluation board dependent internal function
Data-abort exception handler	Handler

Header	None
Declaration	DAbt_Handler
Arguments	None
Return values	None
Caller	Data-abort exception generation code
Description	Handler for the data-abort exception Assembler handler • Obtain the data fault status register. • Obtain the data fault address register. • Obtain the location of the return destination where the exception occurred (the value of LR (link register)). • Call function CDAbtHandler with the above three values as arguments.
Remarks	This function is present for each compiler. See Section 6.1.9 for details of function CDAbtHandler.

6.1.7 CUndefHandler

CUndefHandler	RZ/T1 evaluation board dependent internal function
C handler for undefined instruction exceptions	Synchronous function

Header	None		
Declaration	uint32_t CUndefHandler(uint32_t opcode, uint32_t state, uint32_t LR);		
Arguments	uint32_t opcode	I	Opcode where the undefined instruction occurred
	uint32_t state	I	State when the undefined instruction exception occurred (ARM or Thumb)
	uint32_t LR	I	Location of the return destination where the undefined exception occurred (the value of LR (link register)).
Return values	uint32_t		
Description	If an undefined instruction exception is generated by an FPU instruction, the FPU is enabled and the function returns execution to the caller. In other cases, it enters an endless loop.		
Remarks	This function is present for each compiler.		

6.1.8 CPAbtHandler

CPAbtHandler	RZ/T1 evaluation board dependent internal function
C handler for the prefetch-abort exception	Synchronous function

Header	None		
Declaration	void CPAbtHandler(uint32_t IFSR, uint32_t IFAR, uint32_t LR);		
Arguments	uint32_t IFSR	I	Instruction fault status register
	uint32_t IFAR	I	Instruction fault address register
	uint32_t LR	I	Location of the return destination where the undefined exception occurred (the value of LR (link register)).
Return values	None		
Description	If the function can be restored by using the value of the instruction fault status register, execution is returned to the caller. In other cases, it enters an endless loop.		
Remarks	This function is present for each compiler.		

6.1.9 CDAbtHandler

CDAbtHandler	RZ/T1 evaluation board dependent internal function
C handler for the data-abort exception	Synchronous function

Header	None		
Declaration	void CDAbtHandler(uint32_t DFSR, uint32_t DFAR, uint32_t LR);		
Arguments	uint32_t DFSR	I	Data fault status register
	uint32_t DFAR	I	Data fault address register
	uint32_t LR	I	Location of the return destination where the undefined exception occurred (the value of LR (link register)).
Return values	None		
Description	If the function can be restored by using the value of the data fault status register, execution is returned to the caller. In other cases, it enters an endless loop.		
Remarks	This function is present for each compiler.		

6.1.10 __SVC_1

__SVC_1	RZ/T1 evaluation board dependent internal function
Initializing the caches	Synchronous function

Header	None
Declaration	void __SVC_1(void);
Arguments	None
Return values	None
Caller	submain() (defined in system_Renesas_RZ_T1.c for each compiler.)
Description	This function initializes the caches and MPU.
Remarks	This function is present for each compiler.

6.1.11 __cmain

__cmain	RZ/T1 evaluation board dependent internal function
Startup function for ICCARM	Synchronous function

Header	RTX_CM_lib.h
Declaration	__noreturn __stackless void __cmain(void);
Arguments	None
Return values	None
Description	This function initializes the OS to start it. The OS is started in the following order. • Initialization proceeds if the data section must be initialized by the system startup code. • Function submain (see Section 6.1.14) is called to register the interrupts to be used and initialize the caches. • Call function osKernelInitialize to initialize the CMSIS-RTOS RTX. • Thread os_thread_def_main is started to execute the main function. • Function osKernelStart is called to start the RTOS kernel and start a thread switch.
Remarks	This code is for ICCARM.

6.1.12 cstartup

cstartup	RZ/T1 evaluation board dependent internal function
Startup function for Renesas GCC	Synchronous function

Header	RTX_CM_lib.h
Declaration	void cstartup(void);
Arguments	None
Return values	None
Description	This function initializes the OS to start it. The OS is started in the following order. • Function submain is called to register the interrupts to be used and initialize the caches. • Call function osKernelInitialize to initialize the CMSIS-RTOS RTX. • Thread os_thread_def_main is started to execute the main function. • Function osKernelStart is called to start the RTOS kernel and start a thread switch.
Remarks	This code is for RenesasGCC.

6.1.13 SystemInit

SystemInit	RZ/T1 evaluation board dependent internal function
RZ/T1 evaluation board dependent hardware initialization	Synchronous function

Header	system_Renesas_RZ_T1.h
Declaration	void SystemInit (void);
Arguments	None
Return values	None
Description	This function makes the following platform-dependent initialization settings. • Processing for judging the reset flag • Setting the clock multiplication factor • Initializing the error control module (ECM) • Initializing the interrupt controller.
Remarks	This function is present for each compiler. It implements platform-dependent processing of the CMSIS-RTOS RTX sample programs.

6.1.14 submain

submain	RZ/T1 evaluation board dependent internal function
Registering the interrupts to be used and initializing the caches	Synchronous function

Header	RTX_CM_lib.h
Declaration	void submain(void);
Arguments	None
Return values	None
Description	This function registers the interrupts to be used and initializes the caches. • The OS_Tick_Handler interrupt is registered and initialized (CMT4). • The PendSV_Handler interrupt is registered and initialized (CMT5). • Function __SVC_1 (see Section 6.1.10) is called to initialize the caches and set the MPU.
Remarks	This function is present for each compiler.

6.1.15 __low_level_init

__low_level_init	RZ/T1 evaluation board dependent internal function
Low-level initialization function	Handler

Header	RTX_CM_lib.h	
Declaration	int __low_level_init(void);	
Arguments	None	
Return values	Int	0: Low-level initialization does not proceed. 1: Low-level initialization proceeds.
Caller	void __cmain(void);	
Description	This function determines whether the data section should be initialized by the system startup code. When the function returns 0, the data section does not require initialization.	
Remarks	The function is called from function __cmain and is only for ICCARM (see Section 6.1.11), but its functionality is not implemented.	

6.1.16 R_ATCM_WaitSet

R_ATCM_WaitSet	RZ/T1 evaluation board dependent internal function
Setting wait cycles for access to the ATCM	Synchronous function

Header	r_atcm_init.h			
Declaration	void R_ATCM_WaitSet(uint32_t atcm_wait);			
Arguments		uint32_t atcm_wait	I	ATCM wait setting bits b1 b0 0 0: Optimization to waiting for one cycle 0 1: No optimization to waiting for one cycle 1 0: No wait 1 1: Setting prohibited
Return values	None			
Description	This function sets the number of wait cycles for access to the ATCM. If "optimization" is selected, speeding up memory access to virtually no wait is possible by reading the addresses in advance when the addresses for instruction fetching from the ATCM are consecutive.			
Remarks	To prevent access for fetching by the CPU, be sure to execute this setting function from program code allocated to a memory area other than the ATCM.			

6.1.17 R_CPG_WriteEnable

R_CPG_WriteEnable	RZ/T1 evaluation board dependent internal function
Enabling writing to the clock generator related registers	Synchronous function

Header	r_cpg.h
Declaration	void R_CPG_WriteEnable (void);
Arguments	None
Return values	None
Description	This function enables writing to the clock generator related registers.
Remarks	—

6.1.18 R_CPG_WriteDisable

R_CPG_WriteDisable	RZ/T1 evaluation board dependent internal function
Disabling writing to the clock generator related registers	Synchronous function

Header	r_cpg.h
Declaration	void R_CPG_WriteDisable(void);
Arguments	None
Return values	None
Description	This function disables writing to the clock generator related registers.
Remarks	—

6.1.19 R_CPG_PLL_Wait

R_CPG_PLL_Wait	RZ/T1 evaluation board dependent internal function
Processing to wait for PLL stabilization	Synchronous function

Header	r_cpg.h
Declaration	void R_CPG_PLL_Wait(void);
Arguments	None
Return values	None
Description	This function handles processing to wait for 100 µs as the oscillation settling time for PLL1 by using CMT0.
Remarks	The code within this function initializes CMT0 and stops its operation. Care must be taken when using CMT0.

6.1.20 R_ECM_Init

R_ECM_Init	RZ/T1 evaluation board dependent internal function
Initializing the error control module	Synchronous function

Header	r_ecm.h
Declaration	void R_ECM_Init(void);
Arguments	None
Return values	None
Description	This function initializes the error control module.
Remarks	—

6.1.21 R_ECM_CompareError_Wait

R_ECM_CompareError_Wait	RZ/T1 evaluation board dependent internal function
Processing to wait for generation of the compare-match timer interrupt	Synchronous function

Header	r_ecm.h
Declaration	void R_ECM_CompareError_Wait(void);
Arguments	None
Return values	None
Description	This function handles processing to use CMT0 to cause a 15-µs wait for generation of the compare-match timer interrupt.
Remarks	The code within this function initializes CMT0 and stops its operation. Care must be taken when using CMT0.

6.1.22 R_ECM_Write_Reg8

R_ECM_Write_Reg8	RZ/T1 evaluation board dependent internal function
8-bit writing to the error control module register	Synchronous function

Header	r_ecm.h		
Declaration	uint8_t R_ECM_Write_Reg8(uint8_t reg_type, volatile unsigned char *reg, uint8_t value);		
Arguments	uint8_t reg_type	I	Specifies the master and checker for the ECM.
	volatile unsigned char *reg	I	Specifies the address of a register in the ECM (access is only in 8-bit units).
	uint8_t value	I	Specifies the value to be written.
Return values			
Description	This function handles 8-bit writing to the error control module register.		
Remarks	—		

6.1.23 R_ECM_Write_Reg32

R_ECM_Write_Reg32	RZ/T1 evaluation board dependent internal function
32-bit writing to the error control module register	Synchronous function

Header	r_ecm.h		
Declaration	uint8_t R_ECM_Write_Reg32(uint8_t reg_type, volatile unsigned long *reg, uint32_t value);		
Arguments	uint8_t reg_type	I	Specifies the master and checker for the ECM.
	volatile unsigned long *reg	I	Specifies the address of a register in the ECM (access is only in 32-bit units).
	uint32_t value	I	Specifies the value to be written.
Return values			
Description	This function handles 32-bit writing to the error control module register.		
Remarks	—		

6.1.24 R_MPC_WriteEnable

R_MPC_WriteEnable	RZ/T1 evaluation board dependent internal function
Enabling writing to the multi-pin function related registers	Synchronous function

Header	r_mpc.h
Declaration	void R_MPC_WriteEnable(void);
Arguments	None
Return values	None
Description	This function enables writing to the multi-pin function related registers.
Remarks	—

6.1.25 R_MPC_WriteDisable

R_MPC_WriteDisable	RZ/T1 evaluation board dependent internal function
Disabling writing to the multi-pin function related registers	Synchronous function

Header	r_mpc.h
Declaration	void R_MPC_WriteDisable(void);
Arguments	None
Return values	None
Description	This function disables writing to the multi-pin function related registers.
Remarks	—

6.1.26 R_RST_WriteEnable

R_RST_WriteEnable	RZ/T1 evaluation board dependent internal function
Enabling writing to the reset related registers	Synchronous function

Header	r_reset.h
Declaration	void R_RST_WriteEnable(void);
Arguments	None
Return values	None
Description	This function enables writing to the reset related registers.
Remarks	—

6.1.27 R_RST_WriteDisable

R_RST_WriteDisable	RZ/T1 evaluation board dependent internal function
Disabling writing to the reset related registers	Synchronous function

Header	r_reset.h
Declaration	void R_RST_WriteDisable(void);
Arguments	None
Return values	None
Description	This function disables writing to the reset related registers.
Remarks	—

6.1.28 reset_check

reset_check	RZ/T1 evaluation board dependent internal function
Checking whether the device has been reset	Synchronous function

Header	None
Declaration	void reset_check(void);
Arguments	None
Return values	None
Description	This function judges the source of a reset and executes the appropriate sequence.
Remarks	—

6.1.29 cpg_init

cpg_init	RZ/T1 evaluation board dependent internal function
Initializing the clock pulse generator (CPG)	Synchronous function

Header	None
Declaration	void cpg_init(void)
Arguments	None
Return values	None
Description	This function initializes the clock pulse generator (CPG). It sets the CPU clock to 450 MHz by using PLL1. It also operates the low-speed on-chip oscillator.
Remarks	—

6.1.30 copy_to_atcm

copy_to_atcm	RZ/T1 evaluation board dependent internal function
Copying the user application program	Synchronous function

Header	None
Declaration	void copy_to_atcm(void)
Arguments	None
Return values	None
Description	This function copies the user application program from the external flash memory (NOR flash or serial flash memory) to the internal RAM (ATCM) in units of 4 bytes. It also copies exception vectors and variables with initial values for the user application.
Remarks	This function is present for each compiler. This is implemented in assembler and is only for RenesasGCC.

6.1.31 SER_Init

SER_Init	RZ/T1 evaluation board dependent internal function
Initializing the debug serial port	Synchronous function

Header	Serial.h
Declaration	void SER_Init(void)
Arguments	None
Return values	None
Description	This function makes the debug serial pin settings and calls the initialization function. The settings are as follows. Target: SCIF2 Serial port settings: Asynchronous mode, 8-bit character, stop bit length = 1 bit, no flow control Transfer rate: 115200 bps Output pin: P3_0 (TxD2) Input pin: P3_2 (RxD2)
Remarks	Note that SCIF2 is used as the debug serial port.

6.1.32 SER_Enable

SER_Enable	RZ/T1 evaluation board dependent internal function
Enabling the debug serial port	Synchronous function

Header	Serial.h
Declaration	void SER_Enable(void)
Arguments	None
Return values	None
Caller	SER_Init ()
Description	This function enables transmission and reception through the debug serial port.
Remarks	—

6.1.33 SER_Disable

SER_Disable	RZ/T1 evaluation board dependent internal function
Disabling the debug serial port	Synchronous function

Header	Serial.h
Declaration	void SER_Disable(void)
Arguments	None
Return values	None
Description	This function disables transmission and reception through the debug serial port.
Remarks	This function is not used in the sample programs.

6.1.34 SER_Set_baud_rate

SER_Set_baud_rate	RZ/T1 evaluation board dependent internal function
Setting the bit rate for the debug serial port	Synchronous function

Header	Serial.h		
Declaration	void SER_Set_baud_rate(uint32_t baud_rate)		
Arguments	uint32_t baud_rate	I	Not used
Return values	None		
Caller	None		
Description	This function sets the bit rate for the debug serial port.		
Remarks	This function does not have any functionality (it is empty). The bit rate setting is made by function SER_Init (see Section 6.1.31).		

6.1.35 SER_PutChar

SER_PutChar	RZ/T1 evaluation board dependent internal function
Output of a single character code to the debug serial port	Asynchronous function

Header	Serial.h		
Declaration	void SER_PutChar(uint8_t buffer)		
Arguments	uint8_t buffer	I	Output character
Return values	None		
Caller	__read(), __write()		
Description	This function transmits a single character code to the debug serial port. If the transmission FIFO buffer is not empty before transmission, polling continues until the transmission FIFO buffer becomes empty and transmission proceeds once the buffer has become empty.		
Remarks	Since this function incurs a wait over the period of polling if this is required, care should be taken with its usage. It is an output function for use in debugging.		

6.1.36 SER_GetChar

SER_GetChar	RZ/T1 evaluation board dependent internal function
Input of a single character code from the debug serial port	Synchronous function

Header	Serial.h	
Declaration	uint32_t SER_GetChar (void)	
Arguments	None	
Return values	0 to 255	Success: Input character code
	0	Failure: A hardware error occurred.
Caller	__read()	
Description	This function receives a single character code from the debug serial port. If the reception FIFO buffer holds no data, polling continues until the buffer has received data and return from the function does not proceed until then.	
Remarks	Since this function incurs a wait over the period of polling if this is required, care should be taken with its usage. It is an input function for use in debugging.	

6.1.37 interrupt_SER

interrupt_SER	RZ/T1 evaluation board dependent internal function
Debug serial interrupt handler	Synchronous function

Header	Serial.h
Declaration	void interrupt_SER(void);
Arguments	None
Return values	None
Caller	None
Description	This function handles processing by the interrupt handler for the debug serial port.
Remarks	This function does not have any functionality (it is empty).

6.1.38 FPUEnable

FPUEnable	RZ/T1 evaluation board dependent internal function
Enabling the floating-point unit (FPU)	Synchronous function

Header	None
Declaration	void FPUEnable(void);
Arguments	None
Return values	None
Description	This function initializes the floating-point unit (FPU) to enable it.
Remarks	This function is present for each compiler.

6.1.39 vic_isr_001 to vic_isr_300

vic_isr_001 to vic_isr_300	RZ/T1 evaluation board dependent internal function
Vector interrupt processing function	Asynchronous function

Header	None

Declaration	void vic_isr_001(void);
	:
	void vic_isr_299(void);
	void vic_isr_300(void);

Arguments	None
Return values	None
Description	Vector addresses of the interrupts with vector numbers 1 to 300. When an interrupt is generated, R0 and R1 are pushed onto the stack, the vector number is input to R1, and processing jumps to the corresponding vic_isr_<xxx> function.
Remarks	This function is present for each compiler.

6.1.40 vic_isr

vic_isr	RZ/T1 evaluation board dependent internal function
Vector interrupt processing	Asynchronous function

Header	None		
Declaration	void vic_isr(int);		
Arguments	Int	I	A vector number from 1 to 300
Return values	None		
Description	This function handles processing prior to the service routine for a vector interrupt. It shifts the processor mode to supervisor and increments the interrupt nesting level by one. It checks whether the argument is in the range of vector numbers, and if it is, processing jumps to function act_irq (see Section 6.1.41). If the vector number is outside the range, processing jumps to function ret_irq (see Section 6.1.42).		
Remarks	This function is present for each compiler.		

6.1.41 act_irq

act_irq	RZ/T1 evaluation board dependent internal function
Actual interrupt processing	Asynchronous function

Header	None		
Declaration	void act_irq(int);		
Arguments	Int	I	A vector number from 1 to 300
Return values	None		
Description	This function calls function VIC_ClearPIC for deasserting interrupt signals (see Section 6.7.8). It starts the service routine for actual processing from the address assigned to the corresponding vector number in the interrupt table.		
Remarks	This function is present for each compiler.		

6.1.42 ret_irq

ret_irq	RZ/T1 evaluation board dependent internal function
Post-processing for interrupts	Asynchronous function

Header	None
Declaration	void ret_irq(void);
Arguments	None
Return values	None
Description	This function sets the interrupt address register (HVA0) to 0 to end the interrupt processing. It decrements the interrupt nesting level by one. This function handles post-processing for interrupts.
Remarks	This function is present for each compiler.

6.2 IAR C/C++ Compiler Dependent Internal Functions

6.2.1 __read

__read				IAR C/C++ dependent internal function
Reading a character buffer				Synchronous function
Header	None			
Declaration	size_t __read(int handle, unsigned char * buffer, size_t size);			
Arguments		int handle	I	Only "_LLIO_STDIN" (standard in) is specifiable.
		unsigned char * buffer	I/O	Pointer to the read data storage buffer
		size_t size	I	Reception request size (bytes)
Return values	>= 0		Success: Number of received bytes	
	< 0		Failure: Error in reception	
Description	This function receives data from a serial port and stores it at the address specified by the argument buffer.			
Remarks	—			

6.2.2 __write

__write				IAR C/C++ dependent internal function
Writing to a character buffer				Synchronous function
Header	None			
Declaration	size_t __write(int handle, const unsigned char * buffer, size_t size);			
Arguments		int handle	I	Only "_LLIO_STDOUT" (standard out) or "_LLIO_STDERR" (standard err) is specifiable.
		unsigned char * buffer	I	Pointer to the write data storage buffer
		size_t size	I	Transmission request size (bytes)
Return values	>= 0		Success: Number of bytes for transmission	
	< 0		Failure: Error in transmission	
Description	This function transmits data from a serial port.			
Remarks	—			

6.2.3 __lseek

__lseek				IAR compiler dependent internal function
Seeking within a file				Synchronous function
Header	None			
Declaration	int __lseek(int handle, long offset, int whence);			
Arguments		int handle	I	Not used
		long offset	I	Not used
		int whence	I	Not used
Return values	-1		Fixed	
Caller	DLIB library (lseek function)			
Description	This function simply calls a platform-dependent function to handle the actual processing to seek within a file.			
Remarks	This function does not have any functionality (-1 is returned without any processing being executed).			

6.2.4 __close

__close	IAR compiler dependent internal function
Closing a file	Synchronous function

Header	None		
Declaration	int __close(int handle);		
Arguments	int handle	I	Not used
Return values	0		
Caller	DLIB library (close function)		
Description	This function simply calls a platform-dependent function to handle the actual processing to close a file.		
Remarks	This function does not have any functionality (0 is returned without any processing being executed).		

6.3 RenesasGCC Compiler Dependent Internal Functions

6.3.1 _read

_read				RenesasGCC dependent internal function
Reading a character buffer				Synchronous function
Header	None			
Declaration	int _read(int file, char * ptr, int len)			
Arguments		int file	I	Only "STDIN" (standard in) is specifiable.
		char * ptr	I/O	Pointer to the read data storage buffer
		Int len	I	Reception request size (bytes)
Return values	>= 0		Success: Number of received bytes	
	< 0		Failure: Error in reception	
Description	This function receives data from a serial port and stores it at the address specified by the argument buffer. It also outputs received data from the serial port.			
Remarks	—			

6.3.2 _write

_write				RenesasGCC dependent internal function
Writing to a character buffer				Synchronous function
Header	None			
Declaration	int _write(int file, char * ptr, int len)			
Arguments		int file	I	Only "STDOUT" (standard out) or "STDERR" (standard err) is specifiable.
		char * ptr	I	Pointer to the write data storage buffer
		Int len	I	Transmission request size (bytes)
Return values	>= 0		Success: Number of bytes for transmission	
	< 0		Failure: Error in transmission	
Description	This function transmits data from a serial port.			
Remarks	—			

6.3.3 _exit

_exit				RenesasGCC dependent internal function
Ending the program				Synchronous function
Header	None			
Declaration	void _exit(int status);			
Arguments		int status	I	End status
Return values	None			
Description	Return from this function does not proceed because it is an endless loop.			
Remarks	—			

6.3.4 SioRead

SioRead	RenesasGCC dependent internal function
Output of characters to the SCIF	Synchronous function

Header	None		
Declaration	int32_t SioRead(int32_t file_no, int_t * buffer, uint32_t reading_b);		
Arguments	int32_t file_no	I	Only "STDIN" (standard in) is specifiable.
	int_t * buffer	I/O	Character buffer for reading
	uint32_t reading_b	I	Character buffer size
Return values	>= 0		Success: Number of bytes for transmission
	-1		Failure: File number error, read character error
Description	This function receives data from a serial port and stores it at the address specified by the argument buffer. It also outputs received data from the serial port.		
Remarks	—		

6.3.5 SioWrite

SioWrite	RenesasGCC dependent internal function
Input of characters from the SCIF	Synchronous function

Header	None		
Declaration	int32_t SioWrite(int32_t file_no, const int_t * buffer, uint32_t writing_b);		
Arguments	int32_t file_no	I	File number
	const int_t * buffer	I	Character buffer for writing
	uint32_t writing_b	I	Character buffer size
Return values	>= 0		Success: Number of bytes for transmission
	-1		Failure: File number error
Description	This function transmits data from a serial port.		
Remarks	—		

6.3.6 _top_of_heap

_top_of_heap	RenesasGCC dependent internal function
Heap end address acquisition function	Synchronous function

Header	None	
Declaration	char* _top_of_heap(void);	
Arguments	None	
Return values	>= 0	Address where the stack starts
Description	This function returns the address where the stack starts. It is an overwrite function of optlib.	
Remarks	—	

6.3.7 __enable_irq

__enable_irq	RenesasGCC dependent internal function
Enabling IRQ interrupts	Synchronous function

Header	None
Declaration	void __enable_irq(void);
Arguments	None
Return values	None
Description	This function enables IRQ interrupts.
Remarks	—

6.3.8 __disable_irq

__disable_irq	RenesasGCC dependent internal function
Disabling IRQ interrupts	Synchronous function

Header	None	
Declaration	uint32_t __disable_irq(void);	
Arguments	None	
Return values	0	Enables interrupts.
	!0	Disables (inhibits) interrupts.
Description	This function disables IRQ interrupts.	
Remarks	—	

6.3.9 __strex

__strex	RenesasGCC dependent internal function
Exclusive register store instruction	Synchronous function

Header	None		
Declaration	int __strex(unsigned int val, volatile char *ptr);		
Arguments	unsigned int val	I	Value to be written to the register
	volatile char *ptr	I	Address of the register for writing
Return values	0	Writing succeeded because a lock was not in place.	
	1	Writing failed because a lock was in place.	
Description	This function executes the exclusive register store instruction.		
Remarks	—		

6.3.10 __clrex

__clrex	RenesasGCC dependent internal function
Instruction for removing exclusivity	Synchronous function

Header	None
Declaration	void __clrex(void);
Arguments	None
Return values	None
Description	This function executes the instruction for removing exclusivity.
Remarks	—

6.3.11 __ldrex

__ldrex	RenesasGCC dependent internal function
Exclusive register load instruction	Synchronous function

Header	None		
Declaration	unsigned int __ldrex(volatile char *ptr);		
Arguments	volatile char *ptr	I/O	Address of the register to be read
Return values	>= 0	Value read	
Description	This function executes the exclusive register load instruction.		
Remarks	—		

6.4 ARM C/C++ Compiler Dependent Internal Functions

6.4.1 __rt_entry

__rt_entry	ARM C/C++ dependent internal function
Entry to programs that utilize the ARM C library	Synchronous function

Header	RTX_CM_lib.h
Declaration	void __rt_entry (void);
Arguments	None
Return values	None
Caller	Function __main() is called from function Reset_Handler(), which in turn jumps to this function (__rt_entry), following release from the reset state. Function __main() of the ARM C library jumps to this function (__rt_entry) after loading the memory area specified in the scatter file.
Description	This function is the first to be executed first loading the memory to when the ARM C library is used. It makes the following initialization settings. • Initialization of the stack and heap for the ARM C library (by calling __user_setup_stackheap()) • Initialization of the ARM C library (by calling __rt_lib_init()) • Initialization of the CMSIS-RTOS RTX and generation of threads (by calling osKernelInitialize() and osThreadCreate) • Startup of the CMSIS-RTOS RTX (by calling osKernelStart())
Remarks	When osKernelStart() is operating normally, the CMSIS-RTOS RTX starts and processing does not return to this function. When osKernelStart() ends with an error, this function ends itself by calling exit(). This function is implemented in RTX_CM_lib.h of the CMSIS-RTOS RTX header as an inline function.

6.4.2 _user_perthread_libspace

_user_perthread_libspace	ARM C/C++ dependent internal function
Acquiring the local data area for the ARM C library threads	Synchronous function

Header	RTX_CM_lib.h	
Declaration	void *__user_perthread_libspace (void);	
Arguments	None	
Return values	Buffer address in BSP	When the function is called from a thread: Pointer to the 96-byte area for each thread for the CMSIS-RTOS RTX
	Address of __libspace_start	When the function is called from a non-thread: Pointer to the default 96-byte area when RTOS is not supported
Caller	ARM C library	
Description	This function returns the area of static data managed by the ARM C library according to threads. It acquires its own thread's ID to handle processing to return the 96-byte areas by thread. When called from a non-thread context, it returns __libspace_start as the common default area.	
Remarks	The static data to be managed according to threads specifically refers to errno, the state of the floating-point coprocessor, etc. This function is implemented in RTX_CM_lib.h of the CMSIS-RTOS RTX header as an inline function.	

6.4.3 _mutex_initialize

_mutex_initialize	ARM C/C++ dependent internal function
Generating resources for mutexes of the ARM C library	Synchronous function

Header	RTX_CM_lib.h		
Declaration	int _mutex_initialize(OS_ID *mutex);		
Arguments	OS_ID *mutex	I/O	Pointer storage area pointing to the mutex object
Return values	1		Success: A mutex has been generated.
Caller	ARM C library		
Description	This function generates CMSIS-RTOS RTX-dependent semaphore resources for safe operation of the ARM C library threads.		
Remarks	This function is implemented in RTX_CM_lib.h as an inline function.		

6.4.4 _mutex_acquire

_mutex_acquire	ARM C/C++ dependent internal function
Acquiring locking by mutexes of the ARM C library	Synchronous function

Header	RTX_CM_lib.h		
Declaration	void _mutex_acquire(OS_ID *mutex);		
Arguments	OS_ID *mutex	I	Pointer to the mutex resource Specifies the value of the first argument mutex when _mutex_initialize() ends normally.
Return values	None		
Caller	ARM C library		
Description	This function acquires locking by mutexes of the ARM C library.		
Remarks	This function is implemented in RTX_CM_lib.h as an inline function.		

6.4.5 _mutex_release

_mutex_release	ARM C/C++ dependent internal function
Releasing locking by mutexes of the ARM C library	Synchronous function

Header	RTX_CM_lib.h		
Declaration	void _mutex_release(OS_ID *mutex);		
Arguments	OS_ID *mutex	I	Pointer to the mutex resource Specifies the value of the first argument mutex when _mutex_initialize() ends normally.
Return values	None		
Caller	ARM C library		
Description	This function releases locking by mutexes of the ARM C library.		
Remarks	This function is implemented in RTX_CM_lib.h as an inline function.		

6.4.6 fgetc

fgetc				ARM C/C++ dependent internal function
Reading a single character				Synchronous function
Header	None			
Declaration	int fgetc(FILE * file_p);			
Arguments		FILE * file_p	I	Not used
Return values	get_data			Character to be read (a single character)
	EOF			No character (a character cannot be read)
Description	This function reads a single character code from a serial port and returns it.			
Remarks	—			

6.4.7 fputc

fputc				ARM C/C++ dependent internal function
Writing a single character				Synchronous function
Header	None			
Declaration	int fputc(int character, FILE * file_p);			
Arguments		int character	I	Character to be written (a single character)
		FILE * file_p	I	Not used
Return values	character			Character to be written (a single character) Note: The argument is returned as it is.
Description	This function transmits a single character code to a serial port.			
Remarks	—			

6.4.8 ferror

ferror				ARM C/C++ dependent internal function
Detecting the error state of the file				Synchronous function
Header	None			
Declaration	int ferror(FILE * file_p);			
Arguments		FILE * file_p	I	Not used
Return values	0			
Description	This function returns 0 without executing any processing.			
Remarks	This function does not have any functionality (0 is returned without any processing being executed).			

6.4.9 SioRead

SioRead	ARM C/C++ dependent internal function
Reading a character buffer	Synchronous function

Header	sio_char.h			
Declaration	size_t SioRead(int handle, unsigned char * buffer, size_t size);			
Arguments		int handle	I	Only "STDIN" (standard in) is specifiable.
		unsigned char * buffer	I/O	Pointer to the read data storage buffer
		size_t size	I	Reception request size (bytes)
Return values	>= 0			Success: Number of received bytes
	STDERR			Failure: Error in reception
Description	This function receives data from a serial port and stores it at the address specified by the argument buffer.			
Remarks	—			

6.4.10 SioWrite

SioWrite	ARM C/C++ dependent internal function
Writing to a character buffer	Synchronous function

Header	sio_char.h			
Declaration	size_t SioWrite(int handle, const unsigned char * buffer, size_t size);			
Arguments		int handle	I	Only "STDOUT" (standard out) or "STDERR" (standard err) is specifiable.
		const unsigned char * buffer	I	Pointer to the write data storage buffer
		size_t size	I	Transmission request size (bytes)
Return values	>= 0			Success: Number of bytes for transmission
	STDERR			Failure: Error in transmission
Description	This function transmits data from a serial port.			
Remarks	—			

6.4.11 _ttywrch

_ttywrch	ARM C/C++ dependent internal function
Output of a character to a terminal by using the ARM C library	Synchronous function

Header	rt_sys.h			
Declaration	void _ttywrch(int ch);			
Arguments		int ch	I	Not used (output character)
Return values	None			
Caller	ARM C library			
Description	This function outputs the character given as the argument through the debug serial port.			
Remarks	This function does not have any functionality (it is empty).			

6.4.12 _sys_exit

_sys_exit				ARM C/C++ dependent internal function
Ending the program				Synchronous function
Header	None			
Declaration	void _sys_exit(int returncode);			
Arguments	int returncode	I	Not used	
Return values	None			
Description	Return from this function does not proceed because it is an endless loop.			
Remarks	—			

6.5 CMSIS-RTOS RTX Dependent Internal Functions

6.5.1 os_idle_demon

os_idle_demon	CMSIS-RTOS RTX dependent internal function
Idle thread for the CMSIS-RTOS RTX	Thread

Header	RTX_Config.h
Declaration	void os_idle_demon (void);
Arguments	None
Return values	None
Caller	This function resides as the lowest-priority thread for the CMSIS-RTOS RTX. It runs when there are no threads to be executed by the CMSIS-RTOS RTX scheduler.
Description	This thread for the CMSIS-RTOS RTX is executed in the idle state.
Remarks	When the function is running, the Cortex-R4F core is placed in WFI mode. The CMSIS-RTOS RTX scheduler is restarted in response to the generation of an interrupt by the system timer, etc. You need to customize this function if you implement power-saving operation for peripheral module macros. If you want to implement power saving mode through customization, run os_suspend() and os_resume() before and after changing the operating mode.

6.5.2 os_tick_init

os_tick_init	CMSIS-RTOS RTX dependent internal function
Initializing the peripheral timer for the CMSIS-RTOS RTX	Synchronous function

Header	RTX_Config.h
Declaration	int os_tick_init (void);
Arguments	None
Return values	Positive number : Success: IRQ number of the initialized peripheral timer
Caller	Processing for initialization of the CMSIS-RTOS RTX to be executed via osKernelStart()
Description	This function initializes the timer when a timer from a peripheral module macro is used as the system timer for the CMSIS-RTOS RTX instead of the internal timer of the Cortex-R4F core. CMT4 of the RZ/T1 is initialized for this purpose.
Remarks	This function is used when the value of OS_TIMER is 1 in the configuration settings of the CMSIS-RTOS RTX. The default setting of the CMSIS-RTOS RTX sample programs is "OS_TIMER==1 (the peripheral timer is used)" and this function is used.

6.5.3 os_tick_val

os_tick_val	CMSIS-RTOS RTX dependent internal function
Peripheral timer for the CMSIS-RTOS RTX	Synchronous function

Header	RTX_Config.h	
Declaration	uint32_t os_tick_val(void);	
Arguments	None	
Return values		Elapsed clock counter value (within one tick)
Caller	This function is called by calling function osKernelSysTick.	
Description	This function is implemented when the timer from a peripheral module macro is used as the system timer for the CMSIS-RTOS RTX instead of the internal timer of the Cortex-R4F core. It returns the current CMSIS-RTOS RTX system timer value.	
Remarks	This function is implemented to return the value of CMT4.CMCNT. This function is used when the value of OS_TIMER is 1 in the configuration settings of the CMSIS-RTOS RTX. The default setting of the CMSIS-RTOS RTX sample programs is "OS_TIMER==1 (the peripheral timer is used)" and this function is used.	

6.5.4 os_tick_ovf

os_tick_ovf	CMSIS-RTOS RTX dependent internal function
Peripheral timer for the CMSIS-RTOS RTX	Synchronous function

Header	RTX_Config.h	
Declaration	uint32_t os_tick_ovf(void);	
Arguments	None	
Return values	0	No overflow
	1	The overflow occurred.
Caller	This function is called by calling function osKernelSysTick.	
Description	This function is implemented when the timer from a peripheral module macro is used as the system timer for the CMSIS-RTOS RTX instead of the internal timer of the Cortex-R4F core. This function returns the result of testing whether the system timer for the CMSIS-RTOS RTX has overflowed.	
Remarks	This function unconditionally returns 0 (an overflow occurred) at present. This function is used when the value of OS_TIMER is 1 in the configuration settings of the CMSIS-RTOS RTX. The default setting of the CMSIS-RTOS RTX sample programs is "OS_TIMER==1 (the peripheral timer is used)" and this function is used.	

6.5.5 os_tick_irqack

os_tick_irqack	CMSIS-RTOS RTX dependent internal function
Clearing the state of interrupts from the CMSIS-RTOS RTX peripheral timer	Synchronous function

Header	RTX_Config.h
Declaration	void os_tick_irqack (void);
Arguments	None
Return values	None
Caller	OS_Tick_Handler
Description	This function handles processing for clearing the state of interrupts from the timer when the timer from a peripheral module macro is used as the system timer for the CMSIS-RTOS RTX instead of the internal timer of the Cortex-R4F core. CMT4 of the RZ/T1 is operated for this purpose.
Remarks	This function is used when the value of OS_TIMER is 1 in the configuration settings of the CMSIS-RTOS RTX. The default setting of the CMSIS-RTOS RTX sample programs is "OS_TIMER==1 (the peripheral timer is used)" and this function is used.

6.5.6 os_error

os_error	CMSIS-RTOS RTX dependent internal function
Causing the system to hang when a runtime error occurs in the CMSIS-RTOS RTX	Synchronous function

Header	RTX_Config.h		
Declaration	void os_error (uint32_t err_code)		
Arguments	Uint32_t err_code	I	Following CMSIS-RTOS RTX runtime error codes OS_ERR_STK_OVF, OS_ERR_FIFO_OVF, OS_ERR_MBX_OVF
Return values	None		
Caller	This function is called when the CMSIS-RTOS RTX scheduler encounters a runtime error.		
Description	This is function causes the system to hang when a fatal runtime error occurs in the CMSIS-RTOS RTX.		
Remarks	This function is called from the CMSIS-RTOS RTX scheduler and makes the system hang in an endless loop. Since this error is fatal, do not end this function without restarting the system.		

6.5.7 OS_Tick_Handler

OS_Tick_Handler	CMSIS-RTOS RTX dependent internal function
ISR handler for the CMSIS-RTOS RTX peripheral timer	Synchronous function

Header	None		
Declaration	void OS_Tick_Handler(uint32_t IRQn);		
Arguments	uint32_t IRQn	I	Not used
Return values	None		
Caller	This function is called from within the interrupt function specified for interrupts from the timer for the CMSIS-RTOS RTX.		
Description	This handler is for interrupts from the timer when the timer from a peripheral module macro is used as the system timer for the CMSIS-RTOS RTX instead of the internal timer of the Cortex-R4F core. CMT4 of the RZ/T1 is operated for this purpose.		
Remarks	This function is used when the value of OS_TIMER is 1 in the configuration settings of the CMSIS-RTOS RTX. The default setting of the CMSIS-RTOS RTX sample programs is "OS_TIMER==1 (the peripheral timer is used)" and this function is used.		

6.5.8 os_pendsv_init

os_pendsv_init	CMSIS-RTOS RTX dependent internal function
Initializing the PendSV interrupt for the CMSIS-RTOS RTX	Synchronous function

Header	RTX_Config.h
Declaration	void os_pendsv_init (void);
Arguments	None
Return values	None
Caller	osKernelInitialize
Description	This function initializes the PendSV interrupt used for the CMSIS-RTOS RTX. CMT5 of the RZ/T1 is initialized for this purpose.
Remarks	—

6.5.9 os_pendsv_irqack

os_pendsv_irqack	CMSIS-RTOS RTX dependent internal function
Clearing the state of the PendSV interrupt for the CMSIS-RTOS RTX	Synchronous function

Header	RTX_Config.h
Declaration	void os_pendsv_irqack (void);
Arguments	None
Return values	None
Caller	PendSV_Handler
Description	This function handles processing to cause transitions of the PendSV interrupt used for the CMSIS-RTOS RTX from the active to the inactive level. CMT5 of the RZ/T1 is operated for this purpose.
Remarks	—

6.5.10 os_pendsv

os_pendsv	CMSIS-RTOS RTX dependent internal function
CMSIS-RTOS RTX PendSV interrupt handler processing	Synchronous function

Header	None
Declaration	void os_pendsv (void);
Arguments	None
Return values	None
Caller	SVC_Handler, osMailFree, osSignalSet, osMessagePut, osMessageGet, or osSemaphoreRelease
Description	This function handles processing for the PendSV interrupt used for the CMSIS-RTOS RTX. CMT5 of the RZ/T1 is operated for this purpose.
Remarks	The PendSV interrupt uses CMT5. The interrupt request flag IRQ300 is polled to wait for generation of the interrupt after startup.

6.6 Utility Functions

6.6.1 RZ_T1_RSK_InitClock

RZ_T1_RSK_InitClock	RZ/T1 evaluation board dependent internal function
Setting the internal clock multiplication factor	Synchronous function

Header	RZ_T1_RSK_Init.h
Declaration	void RZ_T1_RSK_InitClock(void);
Arguments	None
Return values	None
Caller	SystemInit () function
Description	This function sets the internal clock multiplication factor. CPU clock 450 MHz
Remarks	

6.6.2 InterruptHandlerRegister

InterruptHandlerRegister	Utility function
Function for registering an IRQ handler	Synchronous function

Header	system_Renesas_RZ_T1.h		
Declaration	uint32_t InterruptHandlerRegister (IRQn_Type irq, IRQHandler handler);		
Arguments	IRQn_Type irq	I	IRQ number
	IRQHandler handler	I	Pointer to the handler function
Return values	0		An IRQ handler has been registered.
	1		Argument irq is outside the range of allowable interrupt number settings.
Description	This function registers an IRQ handler. If an IRQ number for which a handler has already been registered is specified, the pointer is overwritten by that to the new handler function.		
Remarks	—		
Usage example	InterruptHandlerRegister(OSTMI0TINT_IRQn, OS_Tick_Handler);		

6.6.3 InterruptHandlerUnregister

InterruptHandlerUnregister	Utility function
Function for deregistering an IRQ handler	Synchronous function

Header	system_Renesas_RZ_T1.h		
Declaration	uint32_t InterruptHandlerUnregister (IRQn_Type irq)		
Arguments	IRQn_Type irq	I	IRQ number
Return values	0		An IRQ handler has been deregistered.
	1		Argument irq is outside the range of allowable interrupt number settings.
Description	This function deregisters an IRQ handler. This function ends normally when an IRQ number for which a handler has not been registered is specified.		
Remarks			
Usage example	InterruptHandlerUnregister(OSTMI0TINT_IRQn);		

6.7 Functions for Vector Interrupt Operations

6.7.1 VIC_EnableIRQ

VIC_EnableIRQ	Vector interrupt operation function
Enabling a vector interrupt	Synchronous function

Header	vic.h		
Declaration	void VIC_EnableIRQ(IRQn_Type IRQn);		
Arguments	IRQn_Type IRQn	I	A vector number from 1 to 300
Return values	None		
Description	This function enables the interrupt with the given vector number.		
Remarks	—		

6.7.2 VIC_DisableIRQ

VIC_DisableIRQ	Vector interrupt operation function
Disabling a vector interrupt	Synchronous function

Header	vic.h		
Declaration	void VIC_DisableIRQ(IRQn_Type IRQn);		
Arguments	IRQn_Type IRQn	I	A vector number from 1 to 300
Return values	None		
Description	This function disables the interrupt with the given vector number.		
Remarks	—		

6.7.3 VIC_SetDetectType

VIC_SetDetectType	Vector interrupt operation function
Setting the type of detection for interrupts	Synchronous function

Header	vic.h		
Declaration	void VIC_SetDetectType(IRQn_Type IRQn, uint32_t detecttype);		
Arguments	IRQn_Type IRQn	I	A vector number from 1 to 300
	uint32_t detecttype	I	0: Set detection by level 1: Set detection by edge.
Return values	None		
Description	This function sets the type of detection for the interrupts with the specified vector number (as level or edge).		
Remarks	—		

6.7.4 VIC_SetPriority

VIC_SetPriority	Vector interrupt operation function
Setting the priority level of a vector interrupt	Synchronous function

Header	vic.h		
Declaration	void VIC_SetPriority(IRQn_Type IRQn, uint32_t priority);		
Arguments	IRQn_Type IRQn	I	A vector number from 1 to 300
	uint32_t priority	I	Interrupt priority levels 0 to 31 For a vector number from 1 to 255: 0 to 15 For a vector number from 256 to 300: 16 to 31
Return values	None		
Description	This function sets the priority level of the interrupt with the specified vector number.		
Remarks	—		

6.7.5 VIC_GetPriority

VIC_GetPriority	Vector interrupt operation function
Acquiring the priority level of a vector interrupt	Synchronous function

Header	vic.h		
Declaration	uint32_t VIC_GetPriority(IRQn_Type IRQn);		
Arguments	IRQn_Type IRQn	I	A vector number from 1 to 300
Return values	Interrupt priority level	Acquisition succeeded.	
	0xFFFFFFFF	Acquisition failed.	
Description	This function acquires the priority level of the interrupt with the specified vector number.		
Remarks	—		

6.7.6 VIC_Init

VIC_Init	Vector interrupt operation function
Initializing the vector interrupt	Synchronous function

Header	None
Declaration	void VIC_Init(void)
Arguments	None
Return values	None
Caller	VIC_Enable()
Description	This function initializes the vector interrupt.
Remarks	—

6.7.7 VIC_Enable

VIC_Enable	Vector interrupt operation function
Enabling a vector interrupt	Synchronous function

Header	vic.h
Declaration	void VIC_Enable(void)
Arguments	None
Return values	None
Description	This function enables a vector interrupt.
Remarks	—

6.7.8 VIC_ClearPIC

VIC_ClearPIC	Vector interrupt operation function
Clearing a detected vector interrupt	Synchronous function

Header	vic.h		
Declaration	void VIC_ClearPIC(IRQn_Type IRQn);		
Arguments	IRQn_Type IRQn	I	A vector number from 1 to 300
Return values	None		
Description	This function clears the interrupt with the given vector number.		
Remarks	—		

6.7.9 VIC_EndInterrupt

VIC_EndInterrupt	Vector interrupt operation function
Ending interrupt processing	Synchronous function

Header	vic.h
Declaration	void VIC_EndInterrupt(void);
Arguments	None
Return values	None
Description	This function handles processing required to end interrupts.
Remarks	Write a desired value to the interrupt address register (HVA0). The interrupt controller recognizes that interrupt processing will end and clears the priority levels of interrupts.

7. Sample Code

The sample code can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision History	Application Note: CMSIS-RTOS RTX for Cortex-R4 RTX Sample Programs (for use with the following combinations of environments and compilers: EWARM and ICCARM, e2studio and Renesas GCC, DS-5 and ARMCC)

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 13, 2017	—	First Edition issued

All trademarks and registered trademarks are the property of their respective owners.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

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.

(Rev.3.0-1 November 2016)

RENESAS

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

© 2017 Renesas Electronics Corporation. All rights reserved.

Colophon 6.0