RX Family

Memory Access Driver Interface Module
Using Firmware Integration Technology

Summary

This application note describes the memory access driver interface module using Firmware Integration Technology (FIT). It is referred to below as the MEMDRV FIT module.

Note that the MEMDRV FIT module functions as an adapter between an upper layer consisting of serial NOR or NAND flash command control middleware and a lower layer consisting of an RSPI, QSPI, or SCI (SPI mode) device driver. The MEMDRV FIT module does not include NOR/NAND flash command control middleware or a RSPI/QSPI device driver. These must be obtained separately.

Target Compilers

- Renesas Electronics C/C++ Compiler Package for RX Family
- GCC for Renesas RX
- IAR C/C++ Compiler for Renesas RX

For details of the confirmed operation contents of each compiler, refer to "5.1 Confirmed Operation Environment".

Related Documents

- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Family RSPI Module Using Firmware Integration Technology (R01AN1827)
- RX Family QSPI Clock Synchronous Single Master Control Module Using Firmware Integration Technology (R01AN1940)
- RX Family SCI Multi-Mode Module Using Firmware Integration Technology (R01AN1815)
- RX Family DMA Controller DMACA Control Module Firmware Integration Technology (R01AN2063)
- RX Family DTC Module Using Firmware Integration Technology (R01AN1819)
- RX Family CMT Module Using Firmware Integration Technology (R01AN1856)
- RX Family LONGQ Module Using Firmware Integration Technology (R01AN1889)

Contents

1. Overview

1.1 MEMDRV FIT Module

The MEMDRV FIT module can be incorporated into your project as an API. For instructions for adding the MEMDRV FIT module, refer to 2.12, Adding the FIT Module to Your Project.

1.2 Overview of MEMDRV FIT Module

Table 1.1 gives an overview of MEMDRV FIT Module.

Table 1.1 Overview

Item		Contents
Number of control devices		Max 2
Endian		Corresponds to little endian / big endian
Firmware Integration Technology(FIT)		Standard
FIT module to be used in combination	Basic setting	BSP FIT Module
	Serial communication	RSPI FIT Module(Single-SPI)(Note2) QSPI SMstr FIT Module(Single/Dual/Quad-SPI) (Note3) SCI FIT Module(SPI mode)(Note4)
	I / O setting	GPIO FIT Module
	Port setting	MPC FIT Module
	Data transfer (Note 1)	DMACA FIT Module DTC FIT Module
	Timer (Note 1)	CMT FIT Module
	Data management	LONGQ FIT Module

Note 1: Only when using DMAC transfer or DTC transfer
Note 2: RX Family RSPI Module Using Firmware Integration Technology (R01AN1827)
Note 3: RX Family QSPI Clock Synchronous Single Master Control Module Firmware Integration Technology (R01AN 1940)
Note 4: RX Family SCI Multi-Mode Module Using Firmware Integration Technology (R01AN1815)

1.3 Overview of API Functions

Table 1.2 lists the API functions contained in the MEMDRV FIT module.

Table 1.2 API Functions

Function	Description
R_MEMDRV_Open()	Memory driver open processing
R_MEMDRV_Close()	Memory driver close processing
R_MEMDRV_Enable()	Memory driver enable processing
R_MEMDRV_Disable()	Memory driver disable processing
R_MEMDRV_EnableTxData()	Data transmit enable processing
R_MEMDRV_DisableTxData()	Data transmit disable processing
R_MEMDRV_EnableRxData()	Data receive enable processing
R_MEMDRV_DisableRxData()	Data receive disable processing
R_MEMDRV_Tx()	Command transmit processing
R_MEMDRV_TxData()	Data transmit processing
R_MEMDRV_Rx()	State or ID receive processing
R_MEMDRV_RxData()	Data receive processing
R_MEMDRV_ClearDMACFlagTx()	DMAC transmit–related interrupt flag clear
R_MEMDRV_ClearDMACFlagRx()	DMAC receive–related interrupt flag clear
R_MEMDRV_1msInterval()	Interval timer count processing
R_MEMDRV_SetLogHdlAddress()	LONGQ FIT module handler address setting processing
R_MEMDRV_Log()	Error log acquisition processing using LONGQ FIT module
R_MEMDRV_GetVersion()	Memory driver version acquisition

1.4 Processing Example

1.4.1 Software Configuration (NOR)

Figure 1.1 shows the software configuration. Table 1.3 shows an overview of the various layers.

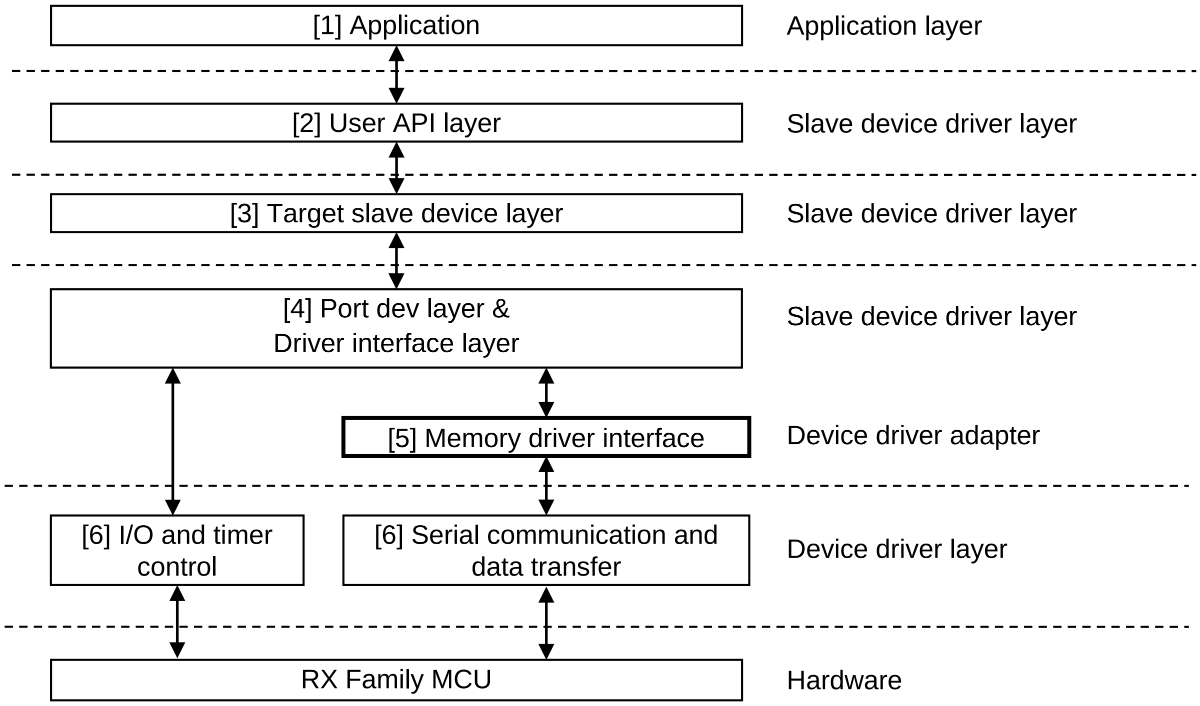

Figure 1.1 Software Configuration

Table 1.3 Overview of Software Blocks

Block Name	Overview
[1] Application	User application
[2] User API layer	The user interface
[3] Target slave device layer	The serial flash memory control module
[4] Port dev layer & Driver interface layer	The control module for controlling the slave device select signal with a microcontroller port. The module for connecting to lower-layer device drivers.
[5] Memory driver interface	**Device driver adapter**
[6] Peripheral IP control (serial communication, ports, etc.)	Device driver layer for executing the control functions listed below. FIT modules can be used to perform these control functions. • Serial communication (clock synchronous single-master control) • I/O control • Pin settings • Data transfer • Timers

1.4.2 API Calling Procedures

Figure 1.2 shows the API calling procedure for CPU transmission and reception.

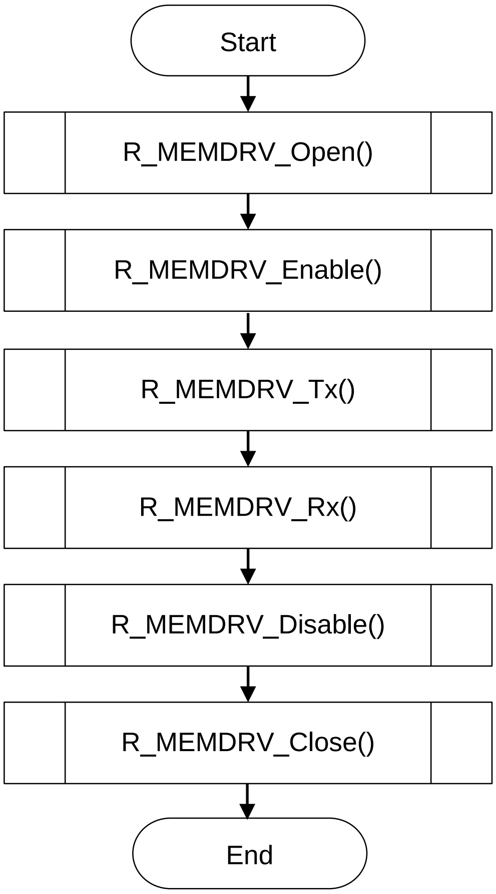

Figure 1.2 API Calling Procedure for CPU Transmission and Reception

Figure 1.3 shows the API calling procedure for using the DMAC or DTC.

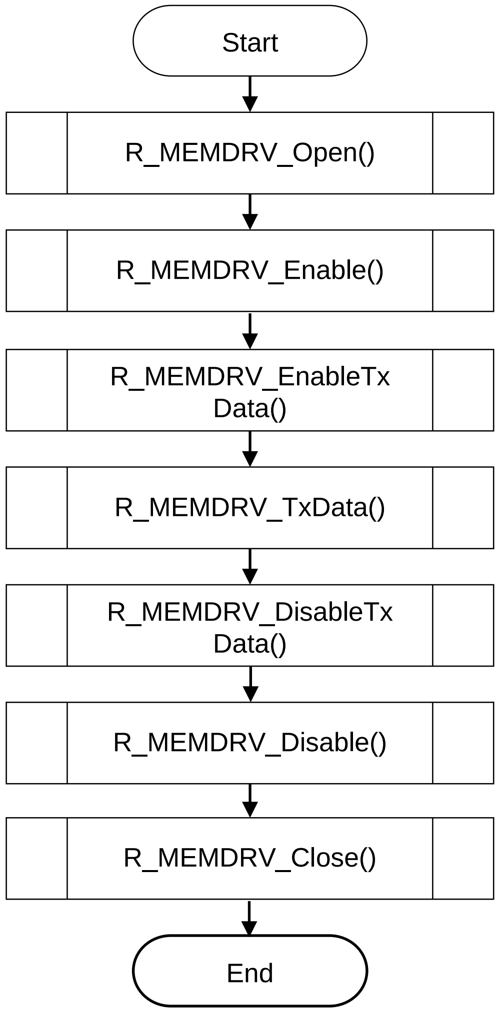

Figure 1.3 API Calling Procedure for Using DMAC/DTC

2. API Information

The operation of the FIT module has been confirmed under the conditions listed below.

2.1 Hardware Requirements

The MCU used must support the following functionality.

- Serial communication (RSPI, QSPI, or simple SPI mode of SCI)
- I/O ports
- DMAC or DTC data transfer (only when using the DMAC or DTC)
- Timers (only when using the DMAC or DTC)

2.2 Software Requirements

The driver is dependent on the following FIT modules.

- r_bsp Rev.5.20 or higher
- r_rspi_rx (when using the RSPI FIT module)
- r_qspi_smstr_rx (when using the QSPI SMstr FIT module for clock synchronous single-master control)
- r_sci_rx (only for CPU transfer when using the SCI FIT module)
- r_dtc_rx (only when using the DTC FIT module for data transfer)
- r_dmaca_rx (only when using the DMACA FIT module for data transfer)
- When substituting another timer or a software timer.
- r_longq (only when using the LONGQ FIT module with error log acquisition functionality enabled)

2.3 Supported Toolchain

The operation of the FIT module has been confirmed with the toolchain listed in 5.1, Confirmed Operation Environment.

2.4 Interrupt Vector

Interrupts are enabled during DMAC or DTC transfer operation. For details on interrupts, refer to the application note of the device driver used to control the MCU's serial communication function (clock synchronous single-master).

2.5 Header Files

All the API calls and interface definitions used are listed in r_memdrv_rx_if.h.

2.6 Integer Types

This project uses ANSI C99. These types are defined in stdint.h.

2.7 Compile Settings

The configuration option settings for the control software are specified in r_memdrv_rx_config.h.
The option names and setting values are described below.

Table 2.1 Configuration Options (config.h)

Configuration options in r_memdrv_rx_config.h	
MEMDRV_CFG_DEVx_INCLUDED The default value for device 0 is “1”. The default value for device 1 is “0”. The “x” in DEVx represents the device number (x = 0 or 1).	This definition is related to device x. (1: enabled, 0: disabled) This option must be set to “enabled” for at least one device.
MEMDRV_CFG_PARAM_CHECKING_ENABLE The default value is “BSP_CFG_PARAM_CHECKING_ENABLE”.	Selects whether or not parameter check processing is included in the code. Parameter check processing is omitted from the code if “0” is selected, resulting in a smaller code size. A setting of “0” means that parameter check processing is omitted from the code. A setting of “1” means that parameter check processing is included in the code.
MEMDRV_CFG_DEVx_MODE_TRNS The DEVx default value is as follows: MEMDRV_TRNS_CPU	Defines the method used to transfer data to the MCU’s on-chip RAM. Choose one of the following settings: MEMDRV_TRNS_CPU: CPU transfer (Software transfer) MEMDRV_TRNS_DMAC: DMAC transfer MEMDRV_TRNS_DTC: DTC transfer
MEMDRV_CFG_DEVx_MODE_DRVR The DEVx default value is as follows: MEMDRV_DRVR_RX_FIT_RSPI	Defines the device driver used. Choose one of the following settings: MEMDRV_DRVR_RX_FIT_RSPI: RSPI FIT module MEMDRV_DRVR_RX_FIT_QSPI_SMSTR: QSPI clock synchronous single-master control FIT module MEMDRV_DRVR_RX_FIT_SCI_SPI: SCI clock synchronous control FIT module(working in SPI mode)
MEMDRV_CFG_DEVx_MODE_DRVR_CH The DEV0 default value is as follows: MEMDRV_DRVR_CH0	Defines the channel number of the device driver used. Choose one of the following (CH0 to CH15): MEMDRV_DRVR_CH0 MEMDRV_DRVR_CH1 ... MEMDRV_DRVR_CH15
MEMDRV_CFG_DEVx_TYPE The default value is 0.	Device type: 0 : NOR FLASH or EEPROM. 1 : NAND FLASH.
MEMDRV_CFG_DEVx_BR The default value is “(uint32_t)(1000000)”.	Sets the bit rate used when issuing commands.
MEMDRV_CFG_DEVx_BR_WRITE_DATA The default value is “(uint32_t)(1000000)”.	Sets the bit rate used when outputting data.
MEMDRV_CFG_DEVx_BR_READ_DATA	Sets the bit rate used when inputting data.

Configuration options in r_memdrv_rx_config.h	
The default value is “(uint32_t)(1000000)”.	
MEMDRV_CFG_DEVx_DMAC_CH_NO_Tx The DEV0 default value is “0”. The DEV1 default value is “2”.	If the DAMC FIT module will be used, specify the DMAC channel number. This DMAC channel is used when transferring data from the MCU’s on-chip RAM to the data buffer of the serial communication function.
MEMDRV_CFG_DEVx_DMAC_CH_NO_Rx The DEV0 default value is “1”. The DEV1 default value is “3”.	If the DAMC FIT module will be used, specify the DMAC channel number. This DMAC channel is used when transferring data from the data buffer of the serial communication function to the MCU’s on-chip RAM.
MEMDRV_CFG_DEVx_DMAC_INT_PRL_Tx The DEVx default value is “10”.	If the DAMC FIT module will be used, specify the DMAC interrupt priority level. This DMAC channel is used when transferring data from the MCU’s on-chip RAM to the data buffer of the serial communication function.
MEMDRV_CFG_DEVx_DMAC_INT_PRL_Rx The DEVx default value is “10”.	If the DAMC FIT module will be used, specify the DMAC interrupt priority level. This DMAC channel is used when transferring data from the data buffer of the serial communication function to the MCU’s on-chip RAM.
MEMDRV_CFG_LONGQ_ENABLE The default value is “0” (disabled).	When using the FIT module’s BSP environment, you can specify whether or not to use debug error log acquisition processing (1: enabled, 0: disabled). When disabled, the processing is omitted from the code. When enabled, the processing is included in the code. The separate LONGQ FIT module is required in order to use this option. Also, enable #define xxx_LONGQ_ENABLE in the device driver.

2.8 Memory Usage

Table 2.2 lists the required memory sizes for the MEMDRV FIT module.

The ROM (code and constants) and RAM (global data) usage are determined by the configuration options described in 2.7, Compile Settings. The values listed apply when using the C compiler referenced in 2.3, Supported Toolchain, with the compile options set to their default values. The default compile options are optimization level: 2, optimization type: size priority, data endian order: little-endian. The memory usage will differ depending on the C compiler version and the compile options.

The values in the table below are confirmed under the following conditions.

Module Revision: r_memdrv_rx rev1.03

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00
(The option of "-lang = c99" is added to the default settings of the integrated development environment.)
GCC for Renesas RX 8.3.0.202002
(The option of "-std=gnu99" is added to the default settings of the integrated development environment.)
IAR C/C++ Compiler for Renesas RX version 4.14.1
(The default settings of the integrated development environment.)

Configuration Options: Default settings

Table 2.2 Memory Sizes

ROM, RAM and Stack Memory Usage(Note 1)								
Device	Category		Memory Used					
			Renesas Compiler		GCC		IAR Compiler	
			With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX130	ROM	1 channel used	2245 bytes	2070 bytes	4732 bytes	4468 bytes	3053 bytes	2841 bytes
		2 channels used	2245 bytes	2070 bytes	4764 bytes	4492 bytes	3076 bytes	2864 bytes
	RAM	1 channel used	25 bytes		28 bytes		18 bytes	
		2 channels used	25 bytes		28 bytes		18 bytes	
	Maximum usable stack size		16 bytes		-		196 bytes	
RX231	ROM	1 channel used	2247 bytes	2072 bytes	4756 bytes	4492 bytes	3053 bytes	2841 bytes
		2 channels used	2247 bytes	2072 bytes	4788 bytes	4516 bytes	3074 bytes	2862 bytes
	RAM	1 channel used	25 bytes		28 bytes		18 bytes	
		2 channels used	25 bytes		28 bytes		18 bytes	
	Maximum usable stack size		16 bytes		-		196 bytes	

RX64M	ROM	1 channel used	2247 bytes	2072 bytes	4756 bytes	4492 bytes	3049 bytes	2837 bytes
		2 channels used	2247 bytes	2072 bytes	4788 bytes	4516 bytes	3070 bytes	2858 bytes
	RAM	1 channel used	25 bytes		28 bytes		18 bytes	
		2 channels used	25 bytes		28 bytes		18 bytes	
	Maximum usable stack size		16 bytes		-		228 bytes	
RX71M	ROM	1 channel used	2257 bytes	2082 bytes	4756 bytes	4492 bytes	3049 bytes	2837 bytes
		2 channels used	2257 bytes	2082 bytes	4788 bytes	4516 bytes	3070 bytes	2858 bytes
	RAM	1 channel used	37 bytes		40 bytes		30 bytes	
		2 channels used	37 bytes		40 bytes		30 bytes	
	Maximum usable stack size		16 bytes		-		228 bytes	

Note 1: The values are confirmed under the following conditions.
Endian: Little endian
The clock synchronous single master control software: RSPI
Data transfer mode: Software

2.9 Arguments

The structure for the arguments of the API functions is shown. This structure is listed in r_memdrv_rx_if.h, along with the prototype declarations of the API functions.

2.10 Return Values

The API function return values are shown below. This enumerated type is listed in r_memdrv_rx_if.h, along with the prototype declarations of the API functions.

```
typedef enum e_memdrv_err
{
    MEMDRV_BUSY                  = 1,  /* Successful operation (device is busy) */
    MEMDRV_SUCCESS               = 0,  /* Successful operation */
    MEMDRV_ERR_PARAM             = -1, /* Parameter error */
    MEMDRV_ERR_HARD              = -2, /* Hardware error */
    MEMDRV_ERR_WP                = -4, /* Write-protection error */
    MEMDRV_ERR_TIMEOUT           = -6, /* Time out error */
    MEMDRV_ERR_OTHER             = -7  /* Other error */
} memdrv_err_t;
```

2.11 Callback Function

The MEMDRV FIT module does not use callback functions.

2.12 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) or (5) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User's Guide: e^2 studio (R20AN0451)" for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to "RX Family Adding Firmware Integration Technology Modules to Projects (R01AN1723)" for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User's Guide: CS+ (R20AN0470)" for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to "RX Family Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)" for details.

(5) Adding the FIT module to your project using the Smart Configurator in IAREW
By using the Smart Configurator Standalone version, the FIT module is automatically added to your project. Refer to "RX Smart Configurator User's Guide: IAREW (R20AN0535)" for details.

2.13 “for”, “ while”, and “do while” Expressions

This module uses for, while, and do while expressions (loop processing) for standby states such as waiting for register values to be updated. These instances of loop processing are indicated by the keyword WAIT_LOOP in the comments. Therefore, if you wish to incorporate failsafe processing into the instances of loop processing, you can locate them in the code by searching for the keyword WAIT_LOOP.

An example code listing is shown below.

Example use of *while* expression:

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

Example use of *for* expression:

```
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

Example use of *do while* expression:

```
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

2.14 Limitations

2.14.1 RAM Location Limitations

In FIT, if a value equivalent to NULL is set as the pointer argument of an API function, error might be returned due to parameter check. Therefore, do not pass a NULL equivalent value as pointer argument to an API function.
The NULL value is defined as 0 because of the library function specifications. Therefore, the above phenomenon would occur when the variable or function passed to the API function pointer argument is located at the start address of RAM (address 0x0). In this case, change the section settings or prepare a dummy variable at the top of the RAM so that the variable or function passed to the API function pointer argument is not located at address 0x0.
In the case of CCRX project (e^2 studio V7.5.0), the RAM start address is set as 0x4 to prevent the variable from being located at address 0x0. In the case of GCC project (e^2 studio V7.5.0) and IAR project (EWRX V4.12.1), the start address of RAM is 0x0, so the above measures are necessary.
The default settings of the section may be changed due to IDE version upgrade. Please check the section settings when using the latest IDE.

3. API Functions

3.1 R_MEMDRV_Open()

This function opens the memory driver. This function must be run before calling other API functions.

Format

```
memdrv_err_t R_MEMDRV_Open (
        uint8_t            devno,
        st_memdrv_info_t   *p_memdrv_info
)
```

Parameters

uint8_t devno
    Device number
st_memdrv_info_t *p_memdrv_info
    Memory driver information structure
    uint32_t cnt;
        Data length
    uint8_t * p_data;
        Pointer to data storage location
    uint8_t io_mode;
        QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/*parameter error*/*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Initializes the FIT module used for SPI/QSPI/SCI communication and data transfer (only when DTC/DMAC is selected).

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_Open(devno,&memdrv_info);
```

Special Notes

None

3.2 R_MEMDRV_Close()

This function closes the memory driver.

Format

```
memdrv_err_t R_MEMDRV_Close (
        uint8_t              devno,
        st_memdrv_info_t     *p_memdrv_info
)
```

Parameters

uint8_t devno
  Device number
st_memdrv_info_t *p_memdrv_info
  Memory driver information structure
  uint32_t cnt;
    Data length
  uint8_t * p_data;
    Pointer to data storage location
  uint8_t io_mode;
    QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/*parameter error*/*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Closes the FIT module used for SPI/QSPI/SCI communication and data transfer (only when DTC/DMAC is selected).

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_Close(devno,&memdrv_info);
```

Special Notes

None

3.3 R_MEMDRV_Enable()

This function enables the memory driver.

Format

```
memdrv_err_t R_MEMDRV_Enable (
        uint8_t             devno,
        st_memdrv_info_t    *p_memdrv_info
)
```

Parameters

uint8_t devno
  Device number
st_memdrv_info_t *p_memdrv_info
  Memory driver information structure
  uint32_t cnt;
    Data length
  uint8_t * p_data;
    Pointer to data storage location
  uint8_t io_mode;
    QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS */* Normal end */*

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/*parameter error*/*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Processing to enable the memory driver.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_Enable(devno,&memdrv_info);
```

Special Notes

None

3.4 R_MEMDRV_Disable()

This function disables the memory driver.

Format

memdrv_err_t R_MEMDRV_Disable (
    uint8_t devno,
    st_memdrv_info_t *p_memdrv_info
)

Parameters

uint8_t devno
    Device number
st_memdrv_info_t *p_memdrv_info
    Memory driver information structure
    uint32_t cnt;
        Data length
    uint8_t * p_data;
        Pointer to data storage location
    uint8_t io_mode;
        QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/*parameter error*/*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Disables the memory driver communication settings.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_Disable(devno,&memdrv_info);
```

Special Notes

None

3.5 R_MEMDRV_EnableTxData()

This function enables data transmission.

Format

memdrv_err_t R_MEMDRV_EnableTxData (
    uint8_t devno,
    st_memdrv_info_t *p_memdrv_info
)

Parameters

uint8_t devno
    Device number
st_memdrv_info_t *p_memdrv_info
    Memory driver information structure
    uint32_t cnt;
        Data length
    uint8_t * p_data;
        Pointer to data storage location
    uint8_t io_mode;
        QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Enables the DMAC/DTC FIT module used for data transmission.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_EnableTxData(devno,&memdrv_info);
```

Special Notes

None

3.6 R_MEMDRV_DisableTxData()

This function disables data transmission by the memory driver.

Format

```
memdrv_err_t R_MEMDRV_DisableTxData (
        uint8_t            devno,
        st_memdrv_info_t   *p_memdrv_info
)
```

Parameters

uint8_t devno
  Device number
st_memdrv_info_t *p_memdrv_info
  Memory driver information structure
  uint32_t cnt;
    Data length
  uint8_t * p_data;
    Pointer to data storage location
  uint8_t io_mode;
    QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Disables the DMAC/DTC FIT module settings used for data transmission.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_DisableTxData(devno,&memdrv_info);
```

Special Notes

None

3.7 R_MEMDRV_EnableRxData()

This function enables data reception.

Format

```
memdrv_err_t R_MEMDRV_EnableRxData (
        uint8_t            devno,
        st_memdrv_info_t   *p_memdrv_info
)
```

Parameters

uint8_t devno
    Device number
st_memdrv_info_t *p_memdrv_info
    Memory driver information structure
    uint32_t cnt;
        Data length
    uint8_t * p_data;
        Pointer to data storage location
    uint8_t io_mode;
        QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Enables the DMAC/DTC FIT module settings used for data reception.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_EnableRxData(devno,&memdrv_info);
```

Special Notes

None

3.8 R_MEMDRV_DisableRxData()

This function disables data reception by the memory driver.

Format

memdrv_err_t R_MEMDRV_DisableRxData (
    uint8_t   devno,
    st_memdrv_info_t   *p_memdrv_info
)

Parameters

uint8_t   devno
    Device number
st_memdrv_info_t   *p_memdrv_info
    Memory driver information structure
    uint32_t   cnt;
        Data length
    uint8_t   * p_data;
        Pointer to data storage location
    uint8_t   io_mode;
        QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Disables the DMAC/DTC FIT module settings used for data reception.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;

memdrv_info.cnt = 0;
memdrv_info.p_data = NULL;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_DisableRxData(devno,&memdrv_info);
```

Special Notes

None

3.9 R_MEMDRV_Tx()

This function performs command transmission processing.

Format

memdrv_err_t R_MEMDRV_Tx (
    uint8_t devno,
    st_memdrv_info_t *p_memdrv_info
)

Parameters

uint8_t devno
    Device number
st_memdrv_info_t *p_memdrv_info
    Memory driver information structure
    uint32_t cnt;
        Data length
    uint8_t * p_data;
        Pointer to data storage location
    uint8_t io_mode;
        QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_HARD	*/* Hardware error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Transmits the data specified by the memory driver information structure. Supports CPU transfer only.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;
uint8_t * p_data;

memdrv_info.cnt = 16;
memdrv_info.p_data = p_data;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_Tx(devno,&memdrv_info);
```

Special Notes

None

3.10 R_MEMDRV_TxData()

This function performs data transmission processing.

Format

```
memdrv_err_t R_MEMDRV_TxData (
        uint8_t            devno,
        st_memdrv_info_t   *p_memdrv_info
)
```

Parameters

uint8_t devno
  Device number
st_memdrv_info_t *p_memdrv_info
  Memory driver information structure
  uint32_t cnt;
    Data length
  uint8_t * p_data;
    Pointer to data storage location
  uint8_t io_mode;
    QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_HARD	*/* Hardware error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Transmits the data specified by the memory driver information structure. Supports CPU, DMAC, and DTC transfer.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;
uint8_t * p_data;

memdrv_info.cnt = 16;
memdrv_info.p_data = p_data;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_TxData(devno,&memdrv_info);
```

Special Notes

None

3.11 R_MEMDRV_Rx()

This function performs processing to receive the state and ID.

Format

```
memdrv_err_t R_MEMDRV_Rx (
        uint8_t              devno,
        st_memdrv_info_t     *p_memdrv_info
)
```

Parameters

uint8_t devno
  Device number
st_memdrv_info_t *p_memdrv_info
  Memory driver information structure
  uint32_t cnt;
    Data length
  uint8_t * p_data;
    Pointer to data storage location
  uint8_t io_mode;
    QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_HARD	*/* Hardware error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Receives the data specified by the memory driver information structure. Supports CPU transfer only.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;
uint8_t * p_data;

memdrv_info.cnt = 16;
memdrv_info.p_data = p_data;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_Rx(devno,&memdrv_info);
```

Special Notes

None

3.12 R_MEMDRV_RxData()

This function performs data reception processing.

Format

memdrv_err_t R_MEMDRV_RxData (
    uint8_t devno,
    st_memdrv_info_t *p_memdrv_info
)

Parameters

uint8_t devno
    Device number
st_memdrv_info_t *p_memdrv_info
    Memory driver information structure
    uint32_t cnt;
        Data length
    uint8_t * p_data;
        Pointer to data storage location
    uint8_t io_mode;
        QSPI transfer mode (SINGLE/DUAL/QUAD)

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_HARD	*/* Hardware error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Receives the data specified by the memory driver information structure. Supports CPU, DMAC, and DTC transfer.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;
uint8_t * p_data;

memdrv_info.cnt = 16;
memdrv_info.p_data = p_data;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_RxData(devno,&memdrv_info);
```

Special Notes

None

3.13 R_MEMDRV_ClearDMACFlagTx()

This function clears the transmit empty interrupt flag set at DMAC transfer end.

Format

```
void R_MEMDRV_ClearDMACFlagTx (
        uint8_t         channel
)
```

Parameters

uint8_t channel
Device channel number

Return Values

None

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Clears the transmit empty interrupt flag. Use an interrupt generated at DMAC transfer end.

Example

```
uint8_t channel = 0;

R_MEMDRV_ClearDMACFlagTx(channel);
```

Special Notes

None

3.14 R_MEMDRV_ClearDMACFlagRx()

This function clears the receive buffer full interrupt flag set at DMAC transfer end.

Format

```
void R_MEMDRV_ClearDMACFlagRx (
        uint8_t         channel
)
```

Parameters

uint8_t channel
    Device channel number

Return Values

None

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Clears the receive buffer full interrupt flag. Use an interrupt generated at DMAC transfer end.

Example

```
uint8_t channel = 0;

R_MEMDRV_ClearDMACFlagRx(channel);
```

Special Notes

None

3.15 R_MEMDRV_1msInterval()

This function performs interval timer count processing.

Format

void R_MEMDRV_1msInterval (
void
)

Parameters

None

Return Values

None

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Increments the driver software's internal timer counter while waiting for DMAC or DTC transfer end.

Example

```
R_MEMDRV_1msInterval();
```

Special Notes

None

3.16 R_MEMDRV_SetLogHdlAddress()

This function processes setting of the LONGQ FIT module handler address.

Format

```
memdrv_err_t R_MEMDRV_SetLogHdlAddress (
        uint32_t        user_long_que
)
```

Parameters

uint32_t user_long_que
LONGQ FIT module handler address

Return Values

MEMDRV_SUCCESS	*/* Normal end */*
MEMDRV_ERR_PARAM	*/* Parameter error */*
MEMDRV_ERR_OTHER	*/* Other error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Sets the handler address of the LONGQ FIT module in the memory driver.

Example

```
memdrv_err_t ret = MEMDRV_SUCCESS;
uint32_t long_que_hndl_address = 0;

ret = R_MEMDRV_SetLogHdlAddress(long_que_hndl_address);
```

Special Notes

If MEMDRV_CFG_LONGQ_ENABLE == 0 and this function is called, this function does nothing.

3.17 R_MEMDRV_Log()

This function performs processing to get the error log using the LONGQ FIT module.

Format

```
uint32_t R_MEMDRV_Log (
        uint32_t flg,
        uint32_t fid,
        uint32_t line
)
```

Parameters

flg
    0x00000001 (fixed value)

fid
    0x0000003f (fixed value)

line
    0x00001fff (fixed value)

Return Values

0	*/* Successful operation */*
1	*/* Error */*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Gets the error log.
Call this function to complete acquisition of the error log.

Example

```
memdrv_err_t ret_drv = MEMDRV_SUCCESS;
st_memdrv_info_t  memdrv_info;
uint8_t * p_data;

memdrv_info.cnt = 16;
memdrv_info.p_data = p_data;
memdrv_info.io_mode = MEMDRV_MODE_SINGLE;

ret_drv = R_MEMDRV_Tx(devno,&memdrv_info);
if(MEMDRV_SUCCESS != ret_drv)
{
    R_MEMDRV_Log(0x00000001, 0x0000003f, 0x00001fff);
    R_MEMDRV_Close(devno,&memdrv_info);
}
```

Special Notes

Do not fail to add the separate LONGQ FIT module.

3.18 R_MEMDRV_GetVersion()

This function gets the memory driver version information.

Format

```
uint32_t R_MEMDRV_GetVersion (
        void
)
```

Parameters

None

Return Values

Upper 2 bytes:	*Major version (decimal notation)*
Lower 2 bytes:	*Minor version (decimal notation)*

Properties

Prototype declarations are contained in r_memdrv_rx_if.h.

Description

Returns the driver version information.

Example

```
uint32_t version = 0;

version = R_MEMDRV_GetVersion();
```

Special Notes

None

4. Pin Settings

In order to use the MEMDRV FIT module, it is necessary to assign pins to the peripheral function input and output signals with the multi-function pin controller (MPC) (referred to as “pin settings”). These settings should be made via the lower-layer driver config procedure.

5. Appendix

5.1 Confirmed Operation Environment

This section describes operation confirmation environment for the MEMDRV FIT module.

Table 5.1 Confirmed Operation Environment (Rev.1.00)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio V7.3.0
C compiler	Renesas Electronics C/C++ compiler for RX Family V.3.01.00
	Compile option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian order	Big-endian/little-endian
Module revision	Ver. 1.00
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxxxx)
	Renesas Starter Kit for RX72T (product No.: RTK5572Txxxxxxxxxx)

Table 5.2 Confirmed Operation Environment (Rev.1.01)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/little-endian
Module revision	Ver. 1.01
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxx)

Table 5.3 Confirmed Operation Environment (Rev.1.02)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio V7.7.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/little-endian
Module revision	Ver. 1.02
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 5.4 Confirmed Operation Environment (Rev.1.03)

Item	Description
Integrated development environment	Renesas Electronics e^2 studio 2020-07 IAR Embedded Workbench for Renesas RX 4.14.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.03.00.202002 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/little-endian
Module revision	Ver. 1.03
Board used	Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxxx)

5.2 Troubleshooting

1. Q: I added the FIT module to my project, but when I build it I get the error “Could not open source file ‘platform.h’.”

   A: The FIT module may not have been added to the project properly. Refer to the documents listed below to confirm the method for adding FIT modules:

   - Using CS+
     Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)
   - Using e^2 studio
     Adding Firmware Integration Technology Modules to Projects (R01AN1723)

   When using the FIT module, the RX Family board support package FIT module (BSP module) must also be added to the project. Refer to the application note “RX Family: Board Support Package Module Using Firmware Integration Technology” (R01AN1685) for instructions for adding the BSP module.

2. Q: I added the FIT module to the project, but when I build it I get the error “This MCU is not supported by the current r_memdrv_rx module.”

   A: The FIT module you added may not support the target device chosen in the user project. Check to make sure the FIT module supports the target device.

3. Q: When using the GCC Projects (e^2 studio V7.5.0) or IAR Projects (EWRX V4.12.1), the function R_MEMDRV_Open execution failed.

   A: The situation may correspond to the limitations in Chapter 2.14.1. Please refer to chapter 2.14.1.

6. Reference Documents

User’s Manual: Hardware

The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News

The latest information can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools

RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)

The latest version can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.

There are no applicable technical updates.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Feb. 20, 2019		First edition issued
1.01	May. 20, 2019	-	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Added Target Compilers.
		1	Deleted R01AN1723, R01AN1826, R20AN0451 from Related Documents.
		9	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		10	Table 2.1 Configuration Options (config.h), MEMDRV_CFG_DEVx_MODE_DRVR fixed.
		12	2.8 Memory Usage, amended.
		16	Deleted Target devices describing “WAIT_LOOP” in 2.13 "for", "while" and "do while" comment.
		35	Added Table 5.2 Operation Confirmation Environment (Ver. 1.01).
1.02	Nov. 22, 2019	-	Modified the MEMDRV FIT module
		1	Deleted Target Devices.
		1	Deleted R01AN1833 from Related Documents.
		4	Changed 1.2 Overview of MEMDRV FIT Module.
		6	Changed 1.4.1 Software Configuration (NOR).
		10	Added MEMDRV_CFG_DEVx_TYPE in 2.7 Compile Settings.
		12	2.8 Memory Usage, amended.
		14	Changed 2.10 Return Values.
		16	2.14 Limitations, added.
		17-34	Deleted the Reentrancy for each API in 3 API Functions. Changed “Special Notes” in 3.16 R_MEMDRV_SetLogHdlAddress().
		37	Added Table 5.3 Operation Confirmation Environment (Ver. 1.02).
		37	Added Troubleshooting 3.
		Program	Modified the MEMDRV FIT module due to the software issue [Description] The serial flash FIT sets the upper limit of the number of bytes written/number of bytes read to 4,294,967,295 bytes (0xffffffff), but if you try to write 1024 bytes, no data is transferred. [Workaround] Use rev. 1.02 or a later version of the MEMDRV FIT module.
1.03	Sep. 10, 2020	-	Modified the callback function processing during DMAC/DTC transfer.
		5	Changed Table 1.2 API Functions.
		12	2.8 Memory Usage, amended.
		14	Changed Section 2.12 Adding the FIT Module to Your Project.

Rev.	Date	Description	
		Page	Summary
1.03	Sep. 10, 2020	37	Added Table 5.4 Operation Confirmation Environment (Ver. 1.03).
		Program	Since the r_rspi_rx FIT module has been updated, the following processing has been corrected. R_MEMDRV_ClearDMACFlagTx R_MEMDRV_ClearDMACFlagRx r_memdrv_rspi_callback
		Program	Modified the MEMDRV FIT module due to the software issue [Description] In IAR and big endian, set the device driver to be used to RSPI and set it to transfer data by Software transfer. If you transfer 4 bytes or more of data with R_MEMDRV_TxData() and R_MEMDRV_RxData(), more data than the specified transfer size will be transferred. [Workaround] Use rev. 1.03 or a later version of the MEMDRV FIT module.

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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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 claims 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, drawings, charts, programs, algorithms, and 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 reverse engineer 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, copying or reverse engineering.
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; 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.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; 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 any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using 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 dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of 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. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a 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. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using 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 comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

© 2020 Renesas Electronics Corporation. All rights reserved.