RX Family

MMC Mode MMCIF Driver Firmware Integration Technology

Introduction

This application note describes the MMC Mode MMCIF driver which uses Firmware Integration Technology (FIT). This driver controls MultiMediaCard (MMC card) and Embedded MultiMediaCard (eMMC) in MMC mode using the MultiMediaCard Interface (MMCIF) module included in Renesas Electronics RX Family microcontrollers. In this document, this driver is referred to as the MMCIF driver.

Target Device

RX Family MCU with MMCIF

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

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 “6.1 Confirmed Operation Environment".

Related Documents

- RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)
- RX Family DMA Controller DMACA Control Module Using 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 MMCIF driver

By using this product in conjunction with a separately supplied FAT file system, files can be accessed on a MMC card and eMMC.

Note that MMC card and eMMC are referred to collectively as “MMC” in this document.

The MMCIF driver can be used by being implemented in a project as an API. See section 2.12 Adding the FIT Module to Your Project for details on methods to implement this FIT module into a project.

1.2 Overview of the MMCIF driver

Table 1.1 and Table 1.2 list this driver function.

Table 1.1 MMCIF Functions

Item	Function
Conforming standard	JEDEC Standard JESD84-A441 JEDEC Standard JESD84-B50
MMC control driver	Block type device driver with 512-byte/sector
MMC operating voltage	Only 2.7-3.6 V operation, with 3.3 V signal levels, is supported
MMC bus interface	MMC mode (1-bit/4-bit/8-bit) is supported
Number of MMC devices controlled	One device/channel
MMC speed mode	The Backward-compatible and High-speed mode are supported. This MMCIF driver discriminates the speed mode and mounts the card
MMC memory capacity	Media up to 2 GB is supported with byte access mode, and media over 2 GB is supported with sector access mode.
MMC memory control objects	Only a user area is supported Boot area control is not supported
MMC detection function	MMC card detection is possible
Boot operation mode	Not supported
Background operation	Not supported
High priority interrupt (HPI)	Not supported

Table 1.2 Microcontroller Functions

Item	Function
Target microcontroller	RX Family microcontrollers that include the MMCIF
Microcontroller internal data transfer method	Either software, DMAC, or DTC transfer can be selected When DMAC or DTC transfer is used, separate DMAC transfer or DTC transfer software is required.
Wait time processing	Waiting using a 1 ms counter standard is supported It is necessary for the user to provide, separately every 1 ms, calls to an interval timer count processing function for calling every 1 ms.
Replaceable processing when an OS is used	The wait processing can be replaced with the invoking task delay processing provided by the OS.
Endian order	Both big endian and little endian are supported
Other functions	Firmware Integration Technology (FIT) is supported

1.2.1 Application Structure

Figure 1.1 shows the application structure when a FAT file system is constructed using this MMCIF driver.

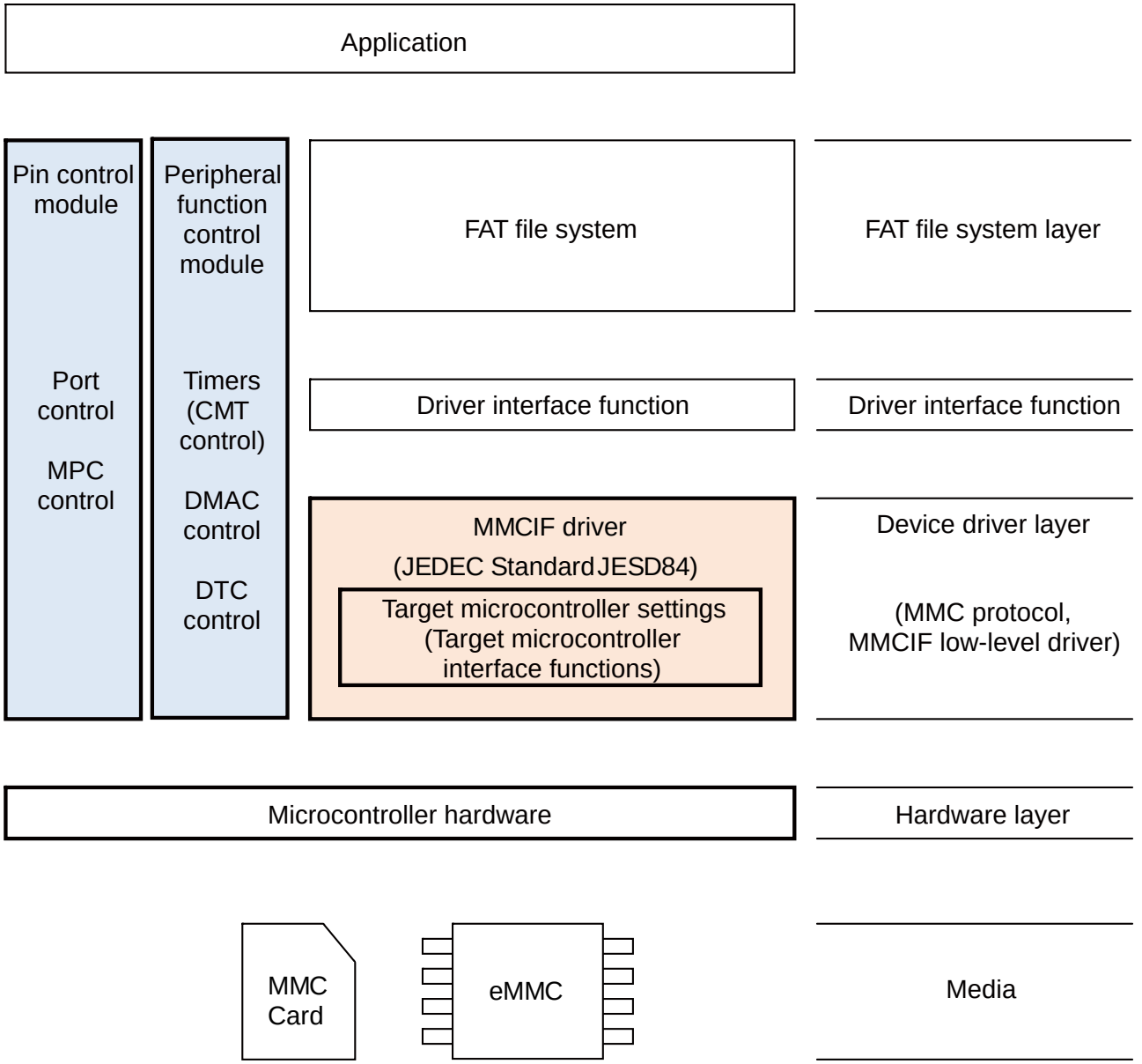


Figure 1.1 Application Structure

(1) FAT File System

This is the software used for MMC file management. A FAT file system must be provided separately.

When high-capacity (over 2 GB) devices are used, the file system, which is an upper layer for this MMCIF driver, must be FAT32.

Open Source FAT File System M3S-TFAT-Tiny: https://www.renesas.com/mw/tfat

(2) Driver Interface Functions

This is the software that implements the layer that connects the Renesas Electronics FAT file system API with the MMCIF driver API. If necessary, please obtain it from the M3S-TFAT-Tiny web page above.

RX Family M3S-TFAT-Tiny Memory Driver Interface Module Firmware Integration Technology

(3) MMCIF Driver

This software implements the JEDEC Standard JESD84 MMC protocol control and MMCIF low-level access control.

This software also includes target microcontroller interface functions that depend on the microcontroller used and interrupt setup files.

(4) Peripheral Function Control Module (Sample Program)

This software implements timer control, DMAC control, and DTC control. It can be acquired as a sample program. See, Related Documents on the first page, for details on acquiring this software.

(5) Pin Control Module (Sample Program)

This is the pin control software used for MMCIF control. The microcontroller resources used consist of the port control (MMCIF function control and MMC card power supply port control) and MPC control (MMCIF function control).

Regarding pin allocation, we recommend allocating system pins at the same time so that the pins used do not conflict.

Note that a sample program that matches the RX64M RSK board is included. This is stored in the FITDemos directory. Refer to this demo program to embed this functionality in an application system.

1.3 API Overview

This MMCIF driver uses the JEDEC STANDARD JESD84 protocol. The table below lists the library functions.

Table 1.3 shows the API Functions for this driver.

Table 1.3 API Functions

Function	Functional Overview
R_MMCIF_Open()	Driver open processing
R_MMCIF_Close()	Driver close processing
R_MMCIF_Get_CardDetection()	Insertion verification processing
R_MMCIF_Mount()	Mount processing
R_MMCIF_Unmount()	Unmount processing
R_MMCIF_Read_Memory()	Read processing *1
R_MMCIF_Read_Memory_Software_Trans()	Read processing (software transfers)
R_MMCIF_Write_Memory()	Write processing *1
R_MMCIF_Write_Memory_Software_Trans()	Write processing (software transfers)
R_MMCIF_Control()	Driver control processing
R_MMCIF_Get_ModeStatus()	Mode status information processing
R_MMCIF_Get_CardStatus()	Card status information processing
R_MMCIF_Get_CardInfo()	Register information processing
R_MMCIF_Int_Handler0()	Interrupt handler
R_MMCIF_Cd_Int()	Insertion interrupt setup (Includes insertion interrupt function callback registration)
R_MMCIF_IntCallback()	Protocol status interrupt callback function registration processing
R_MMCIF_Get_ErrCode()	Driver error code processing
R_MMCIF_Get_BuffRegAddress()	Data register address processing
R_MMCIF_Get_ExtCsd()	Extended CSD processing
R_MMCIF_1ms_Interval()	Interval timer count processing
R_MMCIF_Set_DmacDtc_Trans_Flg()	DMAC/DTC transfer complete flag setting processing
R_MMCIF_Set_LogHdlAddress()	LONGQ module handler address setup *2
R_MMCIF_Log()	Error log processing *2
R_MMCIF_GetVersion()	Driver version information processing

Notes: 1. When DMAC transfers or DTC transfers are set as the data transfer for the operating mode during mount processing, either a DMAC control program or a DTC control program is required. See section 1.4.4.2, DMAC and DTC Control Methods, for the setup procedure.

2. The error log function is provided as a dedicated library module. The LONGQ FIT module is also required.

1.4 Processing Example

1.4.1 Quick Start Guide

The procedure for performing read and write access to an MMC Card using a Renesas Starter Kits (RSK) is described below.

1.4.1.1 Hardware Settings

An SD card socket has already been implemented in the RSK of the MCU equipped with MMCIF.

The SDHI control pins of the MCU and the MMCIF control pins (4-bit / 1-bit bus) are allocated identically[1]. Therefore, SD card socket on RSK can be treated as 9 pin MMC card socket. In addition, eMMC manufacturers provide SD memory card shape compatible boards to eMMC for device evaluation purposes.

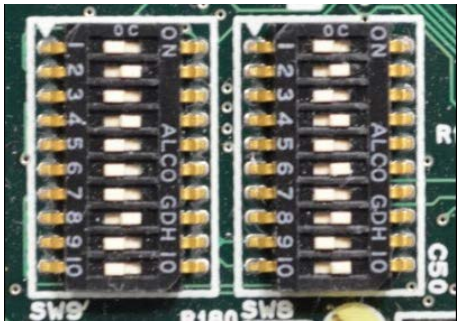
Therefore, it is possible to evaluate the 4-bit / 1-bit bus eMMC by using the MMCIF-equipped MCU and the eMMC's RSK loaded SD memory card supporting board.

Settings must be made on the RSK for each target microcontroller.

(1) RSK for RX64M or RX71M

Make the settings described below to enable the SD Card socket.

SW9		SW8	
Pin Number	**Setting**	**Pin Number**	**Setting**
Pin 1	OFF	Pin 1	OFF
Pin 2	ON	Pin 2	ON
Pin 3	OFF	Pin 3	OFF
Pin 4	ON	Pin 4	ON
Pin 5	OFF	Pin 5	OFF
Pin 6	OFF	Pin 6	ON
Pin 7	OFF	Pin 7	OFF
Pin 8	ON	Pin 8	ON
Pin 9	OFF	Pin 9	OFF
Pin 10	OFF	Pin 10	ON



[1] RSK (type name: RTK50565N2SxxxxxBE) excluding the Renesas Starter Kits for RX65N-2MB.

(2) RSK for RX65N

Make the settings described below to enable the SD Card socket.

SW7		SW8	
Pin Number	**Setting**	**Pin Number**	**Setting**
Pin 1	OFF	Pin 1	OFF
Pin 2	ON	Pin 2	ON
Pin 3	OFF	Pin 3	OFF
Pin 4	ON	Pin 4	ON
Pin 5	OFF	Pin 5	OFF
Pin 6	ON	Pin 6	ON
Pin 7	OFF	Pin 7	OFF
Pin 8	ON	Pin 8	ON
Pin 9	OFF	Pin 9	OFF
Pin 10	ON	Pin 10	OFF

(3) RSK for RX65N-2MB

It cannot be used.

(4) RSK for RX72M

It cannot be used.

(5) RSK for RX72N

It cannot be used.

1.4.1.2 Software Settings

Follow the procedure below to add the software to your project.

1. Create a new project in e^2 studio and download the RX Driver Package.
2. Copy r_mmcif_rx_vX.XX.zip and r_mmcif_rx_vX.XX.xml to the folder containing the e^2 studio FIT modules (normally C:\Renesas\e2_studio\FITModules).
3. Refer to RX Family: Adding Firmware Integration Technology Modules to Projects (R01AN1723), and add r_bsp, r_mmcif_rx and r_cmt_rx to your project.
4. Copy the sample program r_mmcif_rx_demo_main*1 to the src folder of your project.
   Make settings to the configuration options of the sample program. For how to make these settings, see 5.3, Configuration Overview.
5. Please set the media target of #define MMC_CFG_DRIVER_MODE of r_mmcif_rx_config.h to "MMC_MODE_MMC (MMC card)*2".

Note1: *1 Contained in the FITDemos folder in the product package.

Note2: *2 r_mmcif_rx_pin.c has r_mmcif_demo_power_on () function and r_mmcif_demo_power_off () function. Since these functions are based on MMC card control, when the media object of MMC_CFG_DRIVER_MODE is "MMC_MODE_MMC", the power supply voltage is supplied to the SD card socket of RSK. Since these functions are based on MMC card control, when the media object of MMC_CFG_DRIVER_MODE is "MMC_MODE_MMC", the power supply voltage is supplied to the SD card socket of RSK.

1.4.2 Basic Control

1.4.2.1 Supported Commands

This MMCIF driver uses the following commands.

The table below lists the MMC commands, the MMC specifications version, the User's Manual: Hardware and the status of support in this MMCIF driver. The values in the JEDEC Standard JESD84 column indicate the version (for version 4.41 and later) that the command supports.

Table 1.4 Commands Supported by this MMCIF Driver

(—: Not included, ○: Supported, ×: Not supported)

Command	JEDEC Standard JESD84	Microcontrollers Supported	This Product (MMCIF Driver)	Remarks
CMD0	A441	○	○	Used in MMC initialization
CMD1	A441	○	○	Used in MMC initialization
CMD2	A441	○	○	Used in MMC initialization
CMD3	A441	○	○	Used in MMC initialization
CMD4	A441	○	○	Used in MMC initialization
CMD5	A441	○	×	Not used by this MMCIF driver
CMD6	A441	○	○	Used in MMC initialization
CMD7	A441	○	○	Used in MMC initialization
CMD8	A441	○	○	Used in MMC initialization
CMD9	A441	○	○	Used in MMC initialization
CMD10	A441	○	×	Not used by this MMCIF driver
CMD11	A441*	—	×	Not used by this MMCIF driver
CMD12	A441	○	○	Used in read/write processing
CMD13	A441	○	○	Used in read/write processing
CMD14	A441	○	○	Used in MMC initialization
CMD15	A441	○	×	Not used by this MMCIF driver
CMD16	A441	○	○	Used in MMC initialization
CMD17	A441	○	○	Used in read/write processing
CMD18	A441	○	○	Used in read/write processing
CMD19	A441	○	○	Used in MMC initialization
CMD20	A441*	—	×	Not used by this MMCIF driver
CMD21-22	Reserved	—	—	—
CMD23	A441	○	○	Used in read/write processing
CMD24	A441	○	○	Used in read/write processing
CMD25	A441	○	○	Used in read/write processing
CMD26	A441	○	×	Not used by this MMCIF driver
CMD27	A441	○	×	Not used by this MMCIF driver
CMD28	A441	○	×	Not used by this MMCIF driver
CMD29	A441	○	×	Not used by this MMCIF driver
CMD30	A441	○	×	Not used by this MMCIF driver
CMD31	A441	○	×	Not used by this MMCIF driver
CMD32-34	Reserved	—	—	—
CMD35	A441	○	×	Not used by this MMCIF driver
CMD36	A441	○	×	Not used by this MMCIF driver

Command	JEDEC Standard JESD84	Microcontrollers Supported	This Product (MMCIF Driver)	Remarks
CMD37	Reserved	—	—	—
CMD38	A441	○	×	Not used by this MMCIF driver
CMD39	A441	○	×	Not used by this MMCIF driver
CMD40	A441	○	×	Not used by this MMCIF driver
CMD41	Reserved	—	—	—
CMD42	A441	○	×	Not used by this MMCIF driver
CMD43-54	Reserved	—	—	—
CMD55	A441	○	×	Not used by this MMCIF driver
CMD56	A441	○	×	Not used by this MMCIF driver
CMD57-63	Reserved	—	—	—
Root Operation		○	×	Not used by this MMCIF driver

Note: * Commands removed in JEDEC Standard JESD84-B45

1.4.2.2 Relation Between Data Buffers and Data in the MMC

This MMCIF driver is set up with the transmit/receive data pointers passed as arguments. As shown in Figure 1.2, the relationship between the transmit/receive order and the order of the data in the data buffers in RAM is such that data in the transmit buffer is transmit data in the order it appears in the buffer and data is written to the receive buffer in the order received regardless of the endian order.

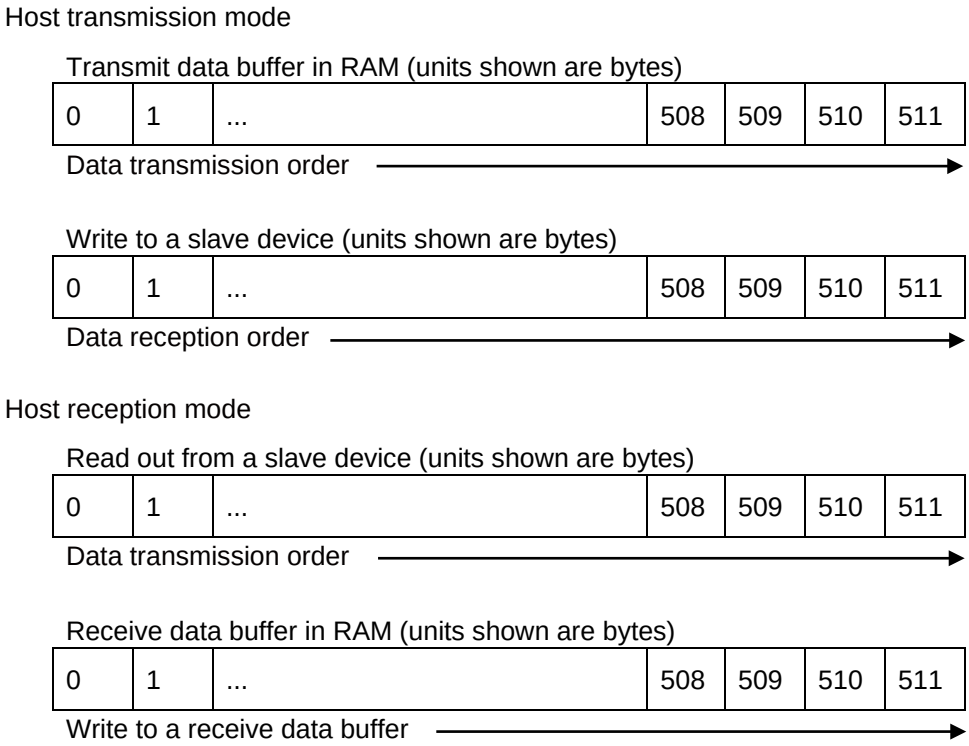


Figure 1.2 Transmission Data Storage

1.4.2.3 Operating Voltage Settings When Initialization

The operating voltage must be set as an argument to the R_MMCIF_Mount() function. During MMC initialization, if it is determined that the MMC cannot operate at the set voltage, the MMC will transition to the inactive state.

For an MMC card, call the R_MMCIF_Unmount() function and after it reaches the unmounted state, the MMC card should be removed. After that, reinsert the card, set the operating voltage again, and perform mount processing again.

For an eMMC, call the R_MMCIF_Unmount() function and after it reaches the unmounted state, stop supply of power to the eMMC. After that, restart supply voltage supply to the eMMC, set the operating voltage again, and perform the mount processing again.

1.4.2.4 Stopping MMC_CLK

To save power, this MMCIF driver only outputs the MMC_CLK signal during library function execution, and stops output of the MMC_CLK signal when the library function terminates.

1.4.2.5 MMCIF Status Verification

To use the MMC, it is necessary to verify the MMCIF status, such as detecting communication completion and detect the MMC card insertion/removal state. This section describes the status verification methods when this MMCIF driver library functions are used.

Status Verification Methods

This MMCIF driver allows users to select either MMCIF interrupts or software poling as MMCIF status verification methods.

The follow status items can be verified.

- MMC Card insertion/removal detection
- MMC protocol

Table 1.5 lists the status items verified by the MMCIF driver library functions.

Table 1.5 Status Items Verified

Type	Status	Remarks
MMC card insertion/removal (Interrupt enable/disable setting with the R_MMCIF_Cd_Int() function)	MMC card inserted/removed state	Detection is possible with the R_MMCIF_Get_CardDetection() function.
MMC protocol (Interrupt enable setting with the R_MMCIF_Mount() function)	Response reception complete	Occurs on each command transmission
	Data transfer request	Occurs on each 512 bytes of transmission
	Protocol error	Occurs when a CRC or other error occurs
	Timeout error	Occurs when a no response state is detected

Setup Methods

When interrupts are selected as the MMC card insertion verification method, interrupts (the MMC_MODE_HWINT setting) must also be selected for MMC protocol status verification with the R_MMCIF_Mount() function.

Note that the R_MMCIF_Int_HandlerX() functions (where X is the channel number) are already registered in the system as the interrupt handlers for the MMCIF interrupts.

MMC Card Insertion Verification by Software Poling and Interrupt

The MMC card insertion state can be verified using the R_MMCIF_Get_CardDetection() function regardless of the enabled/disabled setting of the MMC card insertion interrupt.

When the interrupt has been enabled (MMC_CD_INT_ENABLE) with the R_MMCIF_Cd_Int() function, a callback function can also be executed when the interrupt occurs when an MMC card is inserted. This allows real-time processing for MMC card insertion. Use the R_MMCIF_Cd_Int() function to register the MMC card insertion interrupt callback function.

MMC Protocol Status Verification Using Software Poling

When poling (MMC_MODE_POLL) is set with the R_MMCIF_Mount() function as the MMC protocol status verification method, status items such as data transfer complete wait or response reception wait during communication with the MMC can be verified with software poling.

When software poling is set up, use the target microcontroller interface function, r_mmcif_dev_int_wait() function, within that function, call interrupt status flag register acquisition processing (the r_mmcif_get_intstatus() function), and check the interrupt status flag register (CEINT) value.

Figure 1.3 shows the flowchart for MMC protocol status verification when poling is used.

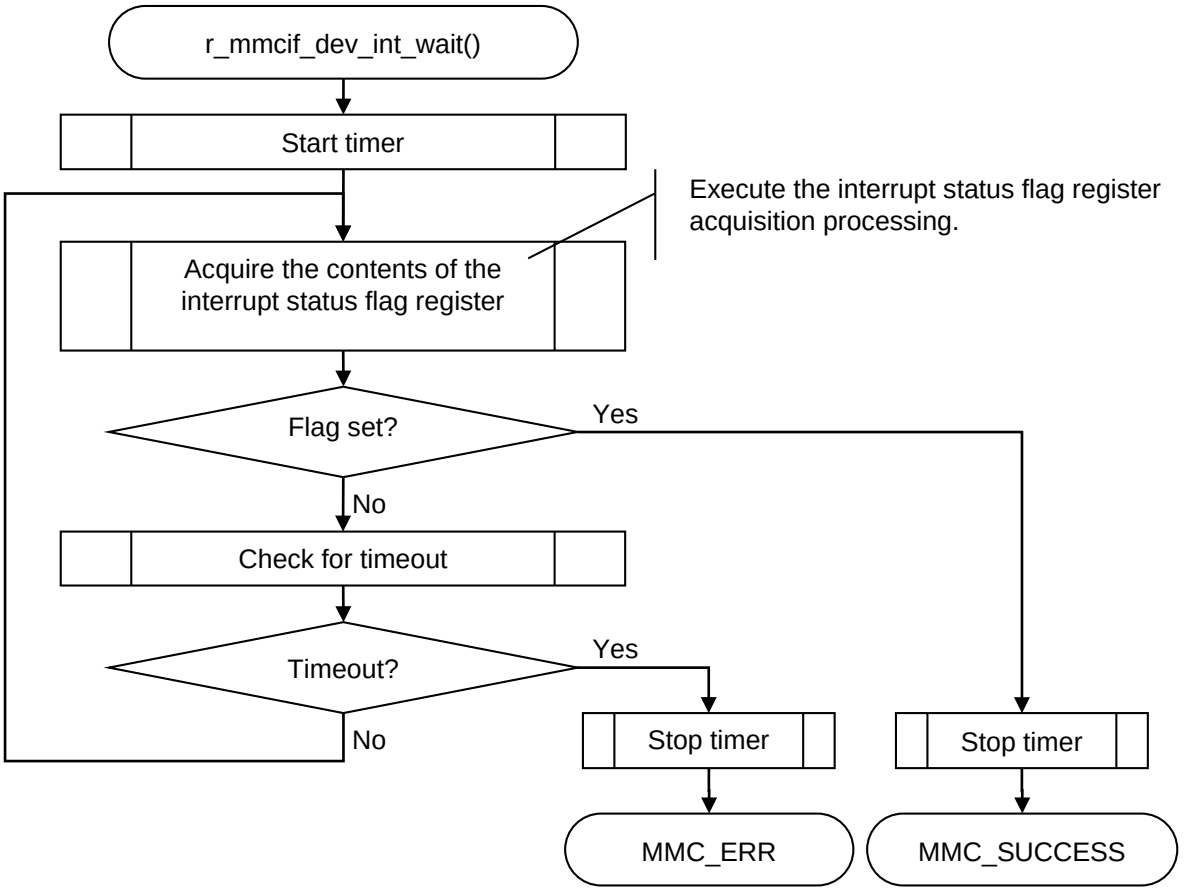


Figure 1.3 MMC Protocol Status Verification Using Software Poling

MMC Protocol Status Verification Using Interrupts

When interrupts (MMC_MODE_HWINT) are set as the MMC protocol status verification method with the R_MMCIF_Mount() function, the status is stored in an internal buffer when the status verification interrupt occurs.

A user registered callback function can be called when a status verification interrupt occurs. The user should register an MMC protocol status interrupt callback function with the R_MMCIF_IntCallback() function.

When waiting for interrupts is enabled, use the target microcontroller interface function, r_mmcif_dev_int_wait() function, within that function, call interrupt status flag register acquisition processing (the r_mmcif_get_intstatus() function), and verify the interrupt occurrence state.

Figure 1.4 shows the flowchart for MMC protocol status verification when interrupts are used.

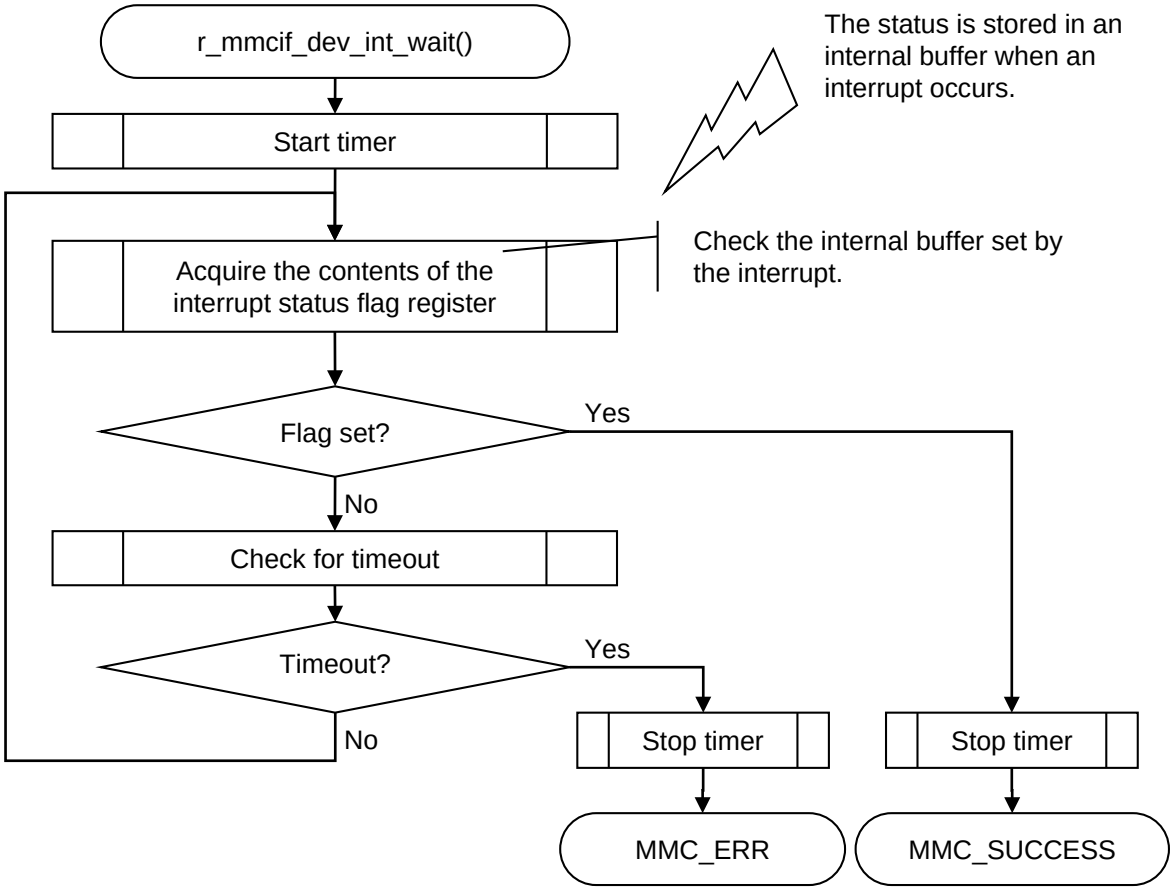


Figure 1.4 MMC Protocol Status Verification Using Interrupts

1.4.3 Control After an Error

1.4.3.1 Handling When an Error Occur

We recommend retrying the processing when an error occurs in read, write, or other processing.

If an error occurs even after retrying the processing, remove the MMC card and initialize the card again. See section 4.4, 4.5, for details on the processing related to MMC card insertion and removal. For the eMMC, temporarily turn off the power supply, then reapply power, and then initialize it.

Also, if using a file system as the upper level application for this MMCIF driver, before removing and reinserting the MMC card, implement any required processing in advance in that upper level application.

1.4.3.2 Handling Error Termination After Transition to the Transfer State (tran)

If an error occurs after transition to the transfer state (tran), a CMD12 command is issued regardless of whether or not there was a data transfer. The purpose of issuing the CMD12 command is to transition to the transfer state (tran). Note, however, that the CMD12 is issued during write processing, the MMC may transition to the busy state. This can cause the next read or write function call to return an error.

1.4.3.3 Error Log Acquisition Methods

Use MMCIF driver source code. Also, the LONGQ FIT module should be acquired separately as well.

Use the following setup procedure to acquire an error log.

R_LONGQ_Open() Setup

Set the third argument of the R_LONGQ_Open() function in the LONGQ FIT module, ignore_overflow, to 1. This will make it possible to use the error buffer as a ring buffer.

Control Procedures

Before calling the R_MMCIF_Open() function, call the following functions in the order shown. See section 3.22, R_MMCIF_Set_LogHdlAddress(), for a setup example.

1. R_LONGQ_Open()
2. R_MMCIF_Set_LogHdlAddress()

Set Up R_MMCIF_Log()

Call this function to terminate error acquisition. See section 3.23, R_MMCIF_Log(), for a setup example.

1.4.4 Control of Other Modules

1.4.4.1 Timers

Timers are used to detect timeouts.

Applications should call R_MMCIF_1ms_Interval() at 1 ms intervals. Note, however, that this is not required when the r_mmcif_dev_int_wait() and r_mmcif_dev_wait() functions in r_mmcif_dev.c have been replaced with operating system processing.

1.4.4.2 DMAC and DTC Control Methods

This section describes the control methods used when DMAC or DTC transfers are used.

This MMCIF driver performs DMAC or DTC transfer starts and transfer complete waiting. For other DMAC or DTC register settings, either use the DMAC or DTC FIT module, or implement your own user processing.

Note that when DMAC transfers are set up, clearing the DMAC transfer complete flag when a DMAC start completes must be performed by user code.

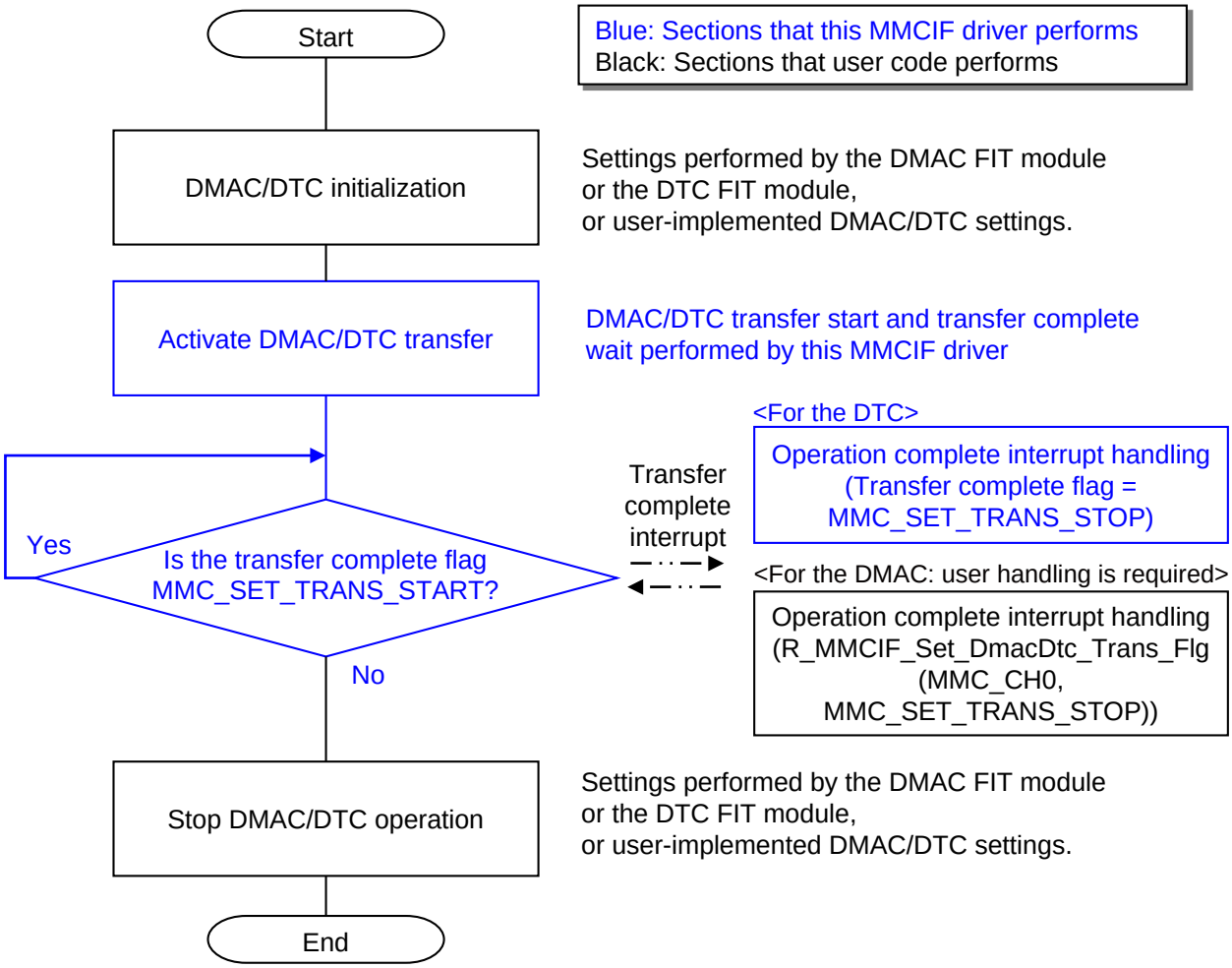
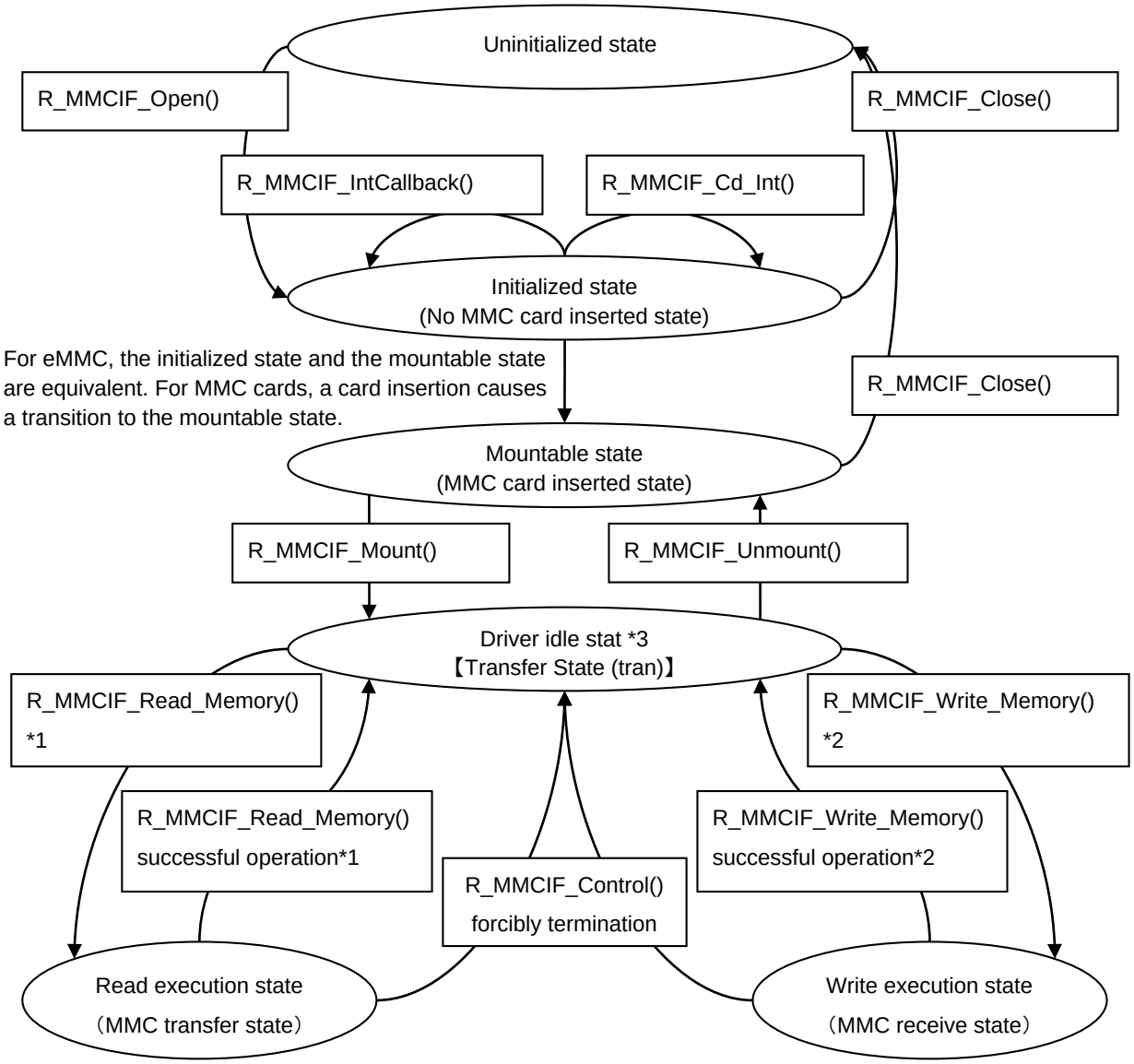


Figure 1.5 Processing for DMAC/DTC Transfer Setup

1.5 State Transition Diagram

Figure 1.6 shows the state transition diagram for this driver.



Note: 1. The MMC transfer state transition command processing, which includes the R_MMCID_Read_Memory_Software_Trans() function, is similar.
2. The MMC receive state transition command processing, which includes the R_MMCID_Write_Memory_Software_Trans() function, is similar.
3. [ ] indicates MMC described in JEDEC Standard JESD84.

Figure 1.6 MMCIF driver State Transition Diagram

1.6 Limitations

1.6.1 Usage Notes

- Argument setup rules and register warranty rules
  The functions provided by this library are coded assuming that they will be called from application programs coded in the C language. The argument setup rules and register warranty rules used by this MMCIF driver conform to those of the C compiler. See the related manuals for details.
- Sections
  Sections with no initialization value should be initialized to 0.
- Notes on using interrupt callback functions
  The interrupt callback functions are called as subroutines from the interrupt handlers.
- All software settings should conform to the hardware actually used.

1.6.2 Notes on MMC Power Supply

A power supply voltage that conforms to the MMC specifications must be provided. See the Power-Up chapter in JEDEC Standard JESD84 for details.

In particular, an application must provide from the system side circuits and cutoff/reapplication control timing that conform to the stipulations on the voltage values and voltage maintenance periods if that application performs MMC card reinsertion control after an MMC card has been removed or if it performs power reapplication control after power to an MMC card has been cut off.

If correct power supply application and cutoff processing is not performed, the power supply system may become unstable due to MMC card insertion or removal and this could cause the microcontroller to go to the reset state.

Applications should call the R_MMCIF_Mount() function only after the power supply has reached the operating voltage. If the time required to reach the operating voltage after power supply voltage starts is insufficient, adjust this wait time.

Also, the application program must implement the wait time processing required to reach the voltage at which MMC removal is allowed after stopping supply of the MMC supply voltage.

1.6.3 Notes on MMC card Insertion and Removal Detection

If the MMC card is removed during communication with this MMCIF driver, a response error for the command will occur and as a result the MMCIF driver will return an error. Also, the time until an error is returned depends on the processing being performed.

However, if a card is removed only briefly during communication, it is possible that an error may not be returned by the MMCIF driver as discussed below.

- If card poling based on a fixed period is used, a removal for less than a single period cannot be detected by the MMCIF driver and if a no response error from the MMC is detected, processing will continue.
- If the card is removed briefly during write, the MMCIF driver may incorrectly recognize the state as write complete. This is because the MMC notifies the write busy signal complete state with a high-level output. (The MMC_Dn (n = 0 to 7) pins are pulled up by external resistors.)

Applications should modify their detection methods for MMC card removal with appropriate measures on the system side by using hardware interrupt control or revising the poling period.

1.6.4 Software Write Protection

This MMCIF driver does not support software protection state control functions.

An error is returned if an attempt is made to write to as software protected area set by another system.

1.6.5 Chattering Control at MMC Card Insertion

This MMCIF driver does not perform any exclusion control for chattering that occurs when an MMC card is inserted or removed. Furthermore, there is no hardware chattering exclusion function. For controlling MMC card, applications should implement card detection processing that take chattering into account by referring to the sample programs and the examples in section 3.3 and section 3.15 of “Example”.

2. API Information

This FIT module has been confirmed to operate under the following conditions.

2.1 Hardware Requirements

The MCU used must support the following functions:

- MMCIF

2.2 Software Requirements

This MMCIF driver depends on the following packages.

- r_bsp Rev.5.20 or higher
- r_dmaca_rx (Only when DMAC transfers using the DMACA FIT module are used)
- r_dtc_rx (Only when DTC transfers using the DTC FIT module are used)
- r_cmt_rx (Only when the compare match timer CMT FIT module is used)
  Other timers or software times can be used for this functionality.

2.3 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 6.1, Operation Confirmation Environment.

2.4 Interrupt Vector

If the macro definition MMC_CFG_DRIVER_MODE is set to MMC_MODE_HWINT, MMCIF interrupts are enabled. Put the system in the interrupts permitted state before the MMCIF driver's open processing, R_MMCIF_Open(), is called.

Table 2.1 lists the interrupt vectors used by the MMCIF driver.

Table 2.1 Interrupt Vectors

Device	Interrupt Vectors
RX64M RX65N RX66N RX71M RX72M RX72N	MMCIF buffer access interrupt (MBFAI) (vector number: 45) GROUPBL1 interrupt (vector number: 111) • MMC detection interrupt (CDETIO) (group interrupt source number: 6) • Error/timeout access interrupt (ERRIO) (group interrupt source number: 7) • Normal operation interrupt (ACCIO) (group interrupt source number: 8)

(1) API Functions that Can be Called from Within Interrupt Handling

Table 2.2 lists the API functions (recommended) that can be called from within interrupt handling.

Note that the interrupt callback functions are called as subroutines from the interrupt handlers.

Table 2.2 MMCIF Driver Library Functions that may be Called from Within an Interrupt Handler

Function	Functional Overview	Remarks
R_MMCIF_Control()	Driver control processing	MMC_SET_STOP (Forced stop request command)

2.5 Header Files

The API calls and interface definitions used are defined in r_mmcif_rx_if.h.

The configuration options for each build are selected in r_mmcif_rx_config.h.

```
#include "r_mmcif_rx_if.h"
```

2.6 Integer Types

This MMCIF driver is coded in ANSI C99. These types are defined in stdint.h.

2.7 Configuration Overview

This MMCIF driver configuration options are set in r_mmcif_rx_config.h.

The table below lists the option names and set values when the RX64M RSK is used.

Configuration options in *r_mmcif_rx_config.h*	
#define MMC_CFG_USE_FIT Note: The default value is "enabled".	Selects whether this MMCIF driver will be used in the BSP environment. If set to "disabled", the r_bsp and other FIT module control will be disabled. Also, it will be necessary to embed this processing some other way. When set to "enabled", the r_bsp and other FIT modules are enabled.
#define MMC_CFG_CHx_INCLUDED Note: The channel 0 default value is "enabled". "x" indicates the channel number.	Selects whether the corresponding channel is used. If set to "disabled", the processing related to the corresponding channel will be omitted from the code. If set to "enabled", the processing related to the corresponding channel will be included in the code. Other channel definitions must be added if a microcontroller that supports multiple channels is used.
#define MMC_CFG_DRIVER_MODE (MMC_MODE_HWINT | MMC_MODE_eMMC | MMC_MODE_1BIT) Note: The default value is "status verification: hardware interrupt, data transfers: software transfers, supported media: eMMC, MMC bus width: MMC mode 1-bit bus".	This definition can be used as the argument p_mmc_Config->mode to the R_MMCIF_Mount() mount processing function. Refer to the section "Mount processing function R_MMCIF_Mount()" and define p_mmc_Config->mode.
#define MMC_CFG_CHx_CD_ACTIVE (0) Note: The default value is "disabled". "x" indicates the channel number.	When allocation of the MMC_CD pin is not required, it can be removed individually from the objects of MMCIF driver control. If the pin is to be allocated as an object of the MMCIF functions, set the value to (1). To remove the pin as an object, set the value to (0). When this pin is removed as an object of control, it can be used for another purpose (such as a general-purpose I/O port). This MMCIF driver does not have the functionality to remove pins from control, and does not have a function for setting the pins that will be used. Therefore, if this pin is to be used for another purpose, it must be set up separately to match the pins used. These settings are required for each channel used.
#define MMC_CFG_DIV_HIGH_SPEED MMC_DIV_2 /* 52MHz or less clock */ Note: The default value is "MMC_DIV_2" (divide by 2)*1	Defines the clock frequency for High-speed mode. For the JEDEC Standard JESD84, the maximum clock frequency is 52 MHz. Set the MMCIF clock frequency setting bits (CLKDIV[3:0]) to the PCLKB divisor. The set value should be in the range MMC_DIV_2 to MMC_DIV_1024 (from divide by 2 to divide by 1024). For example, if PCLKB = 60 MHz and the High-speed mode clock frequency is 30 MHz, then MMC_DIV_2 (divide by 2) should be set.
#define MMC_CFG_DIV_BACKWARD_COM_SPEED MMC_DIV_4 /* 26MHz or less clock */ Note: The default value is "MMC_DIV_4" (divide by 4)*1	Defines the clock frequency for Backward-compatible mode. For the JEDEC Standard JESD84, the maximum clock frequency is 26 MHz. This item is set in the same way as is the High-speed mode setting described above. For example, if PCLKB = 60 MHz and the Backward-compatible mode clock frequency is 15 MHz, then MMC_DIV_4 (divide by 4)

	should be set.
#define MMC_CFG_DIV_INIT_SPEED MMC_DIV_1024 /* 400KHz or less clock */ Note: The default value is "MMC_DIV_1024" (divide by 1024)*1*2	Defines the clock frequency for card identification mode. For the JEDEC Standard JESD84, the maximum clock frequency is 400 kHz. This item is set in the same way as is the High-speed mode setting described above. For example, if PCLKB = 60 MHz and the card identification mode clock frequency is 58 kHz, then MMC_DIV_1024 (divide by 1024) should be set.
#define MMC_CFG_TIMEOUT_TRANS (0x000000a0ul) /* CECLKCTR register : Write data/read data timeout */ Note: The default value is "0x000000a0ul"	This is the timeout setting for write data and read data. Set the timeout value for the clock control register (CECLKCTRL) SRWDTO[3:0] bits in bits 7 to 4.
#define MMC_CFG_TIMEOUT_RESBUSY (0x00000f00ul) /* CECLKCTR register : Response busy timeout */ Note: The default value is "0x00000f00ul"	This is the response busy timeout setting. Set the timeout value for the clock control register (CECLKCTRL) SRBSYTO[3:0] bits in bits 11 to 8.
#define MMC_CFG_TIMEOUT_RES (0x00002000ul) /* CECLKCTR register : Response timeout */ Note: The default value is "0x00002000ul"	This is the response reception timeout setting. Set the timeout value for the clock control register (CECLKCTRL) SRSPT[3:0] bits in bits 13 and 12.
#define MMC_CFG_CHx_INT_LEVEL (10) /* MMC channel x interrupt level */ Note: The default value is "(10)"	This sets the normal operation interrupt (ACCIO), the error/timeout interrupt (ERRIO), and the MMC detection interrupt (CDETIO) levels.
#define MMC_CFG_CHx_INT_LEVEL_DMADTC (10) /* MMC channel x DMA/DTC interrupt level */ Note: The default value is "(10)"	Set this define macro to the MMCIF buffer access interrupt (MBFAI) level. This is the interrupt level when DMAC/DTC is used and data is written to the MMCIF buffer and when data is read from the MMCIF buffer.
/* #define MMC_CFG_LONGQ_ENABLE */ Note: The default value is "disabled"	This must be set when an error log acquisition function using the LONGQ FIT module is used. When this function is used, a debugging module (a dedicated module created with this definition enabled) must be used and the LONGQ FIT module must be included.

Notes: 1. The symbols MMC_DIV_n (where n is an integer and indicates the divisor) indicate the divisor for the MMCIF PCLK. There are cases where it is not possible to set the JEDEC Standard JESD84 maximum transfer frequency due to the electrical characteristics of the microcontroller used. See the User's Manual: Hardware document for the microcontroller used to determine the maximum transfer frequency that can be set.

2. In card identification mode, the MMC CMD line becomes an open-drain circuit. Therefore the clock frequencies that can be used depend on the pull-up resistor and the load capacitance for the MMC_CMD pin. See section 4.6, Hardware Settings, for details.

2.8 Code Size

The code sizes for the latest version of the driver are shown below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview.

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

Module Revision: r_mmcif_rx rev1.07

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(The option of "-lang = c99" is added to the default settings of the integrated development environment.)

GCC for Renesas RX 4.8.4.201902

(The option of "-std=gnu99" is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.12.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

Table 2.3 Code Size

ROM, RAM and Stack Code Sizes (Note1)				
Device	Category	Memory Used		
		Renesas Compiler	GCC	IAR Compiler
RX65N	ROM Note2	9650 bytes	18,004 bytes	12,749 bytes
	RAM Note2, 3	276 bytes (Work buffer) Note4	624 bytes	624 bytes
	Maximum user stack used	320 bytes	-	288 bytes
	Maximum interrupt stack used	40 bytes	-	68 bytes

Note 1: This is the value in the case of little-endian. The memory sizes indicated above differ depending on the endian order.
Note 2: The memory sizes depend on the data transfer method used and other aspects.
Note 3: Data buffers used for read/write are not included.
Note 4: For details of work buffer, see Open processing function.

2.9 Parameters

This section presents the structures used as arguments to the API functions. These structures are included in the file r_mmcif_rx_if.h along with the API function prototype declarations.

(1) e_mmc_enum_cmd Structure Definition

```
typedef enum e_mmc_enum_cmd
{
   MMC_SET_STOP,
} mmc_enum_cmd_t;
```

(2) e_mmc_enum_trans Structure Definition

```
typedef enum e_mmc_enum_trans
{
   MMC_SET_TRANS_STOP,
   MMC_SET_TRANS_START
} mmc_enum_trans_t;
```

(3) mmc_cmd_t Structure Definition

```
typedef struct
{
   mmc_enum_cmd_t cmd;
   uint32_t       mode;    /* Lock/Unlock operation code */
   uint8_t        *p_buff;
   uint32_t       size;
} mmc_cmd_t;
```

(4) mmc_cfg_t Structure Definition

```
typedef struct
{
   uint32_t       mode;    /* MMC Driver operation mode */
   uint32_t       voltage;    /* Operation voltage */
} mmc_cfg_t;
```

(5) mmc_access_t Structure Definition

```
typedef struct
{
   uint8_t        *p_buff;
   uint32_t       lbn;
   int32_t        cnt;
   uint32_t       mode;
   uint32_t       rw_mode;
} mmc_access_t;
```

(6) mmc_card_status_t Structure Definition

```
typedef struct
{
   uint32_t    card_sector_size; /* Sector size (user area) */
   uint8_t     csd_structure; /* CSD structure
                  0 : CSD version No.1.0
                  1 : CSD version No.1.1
                  2 : CSD version No.1.2
                  3 : Version is coded in the CSD_STRUCTURE byte
                      in the EXT_CSD register */
   uint8_t     speed_mode; /* Card speed mode
                  Supported speed bit 5,4 :  0 0 Backward-compatible
                                             0 1 High-speed 26MHz Max
                                             1 1 High-speed 52MHz Max
                  Current speed   bit 1,0 :  0 0 Backward-compatible
                                             0 1 High-speed 26MHz Max
                                             1 1 High-speed 52MHz Max */
   uint8_t     csd_spec;   /* MMC spec version */
   uint8_t     if_mode;    /* Bus width */
   uint8_t     density_type;  /* Card density type */
   uint8_t     rsv[3];        /* Reserve */
} mmc_card_status_t;
```

(7) mmc_card_reg_t Structure Definition

```
typedef struct
{
   uint32_t ocr[1];     /* OCR value */
   uint32_t cid[4];     /* CID value */
   uint32_t csd[4];     /* CSD value */
   uint32_t dsr[1];     /* DSR value */
   uint32_t rca[1];     /* RCA value */
} mmc_card_reg_t;
```

2.10 Return Values / Error Codes

This section presents the return values from the API functions. This enumeration type is defined in the file r_mmcif_rx_if.h along with the API function prototype declarations.

If an error occurs during processing, these MMCIF driver library functions return an error code in their return value. Also, the error code can be acquired using the R_MMCIF_Get_ErrCode() function after execution of the R_MMCIF_Mount(), R_MMCIF_Read_Memory(), and R_MMCIF_Write_Memory() functions*.

Table 2.4 lists the error codes. Note that values not listed in the table are reserved for future expansion.

Note: * The R_MMCIF_Read_Memory_Software_Trans() and R_MMCIF_Write_Memory_Software_Trans() functions operate similarly.

Table 2.4 Error Codes

Macro Definition	Value	Meaning	
MMC_SUCCESS_LOCKED_CARD	1	Successful operation (card locked state)	Successful operation. Note, however, that the card is in the locked state.
MMC_SUCCESS	0	Successful operation	Successful operation.
MMC_ERR	–1	General error	R_MMCIF_Open() function not yet executed, argument parameter error, and other errors
MMC_ERR_WP	–2	Write protect error	Write to an MMC card in the write protected state
MMC_ERR_HOST_TOE	–9	Host timeout error	Error in the r_mmcif_dev_int_wait() function.
MMC_ERR_CARD_LOCK	–11	Card locked error	R1 response card status error (CARD_IS_LOCKED)
MMC_ERR_CARD_UNLOCK	–12	Card unlocked error	R1 response card status error (LOCK_UNLOCK_FAILED)
MMC_ERR_CARD_CRC	–13	Card CRC error	R1 response card status error (COM_CRC_ERROR)
MMC_ERR_CARD_ECC	–14	Card ECC error	R1 response card status error (CARD_ECC_FAILED)
MMC_ERR_CARD_CC	–15	Card CC error	R1 response card status error (CC_ERROR)
MMC_ERR_CARD_ERROR	–16	Card error	R1 response card status error (ERROR)
MMC_ERR_NO_CARD	–18	No card inserted error	No card is inserted.
MMC_ERR_CPU_IF	–30	Target microcontroller interface function error	Target microcontroller interface function error (other than the r_mmcif_dev_int_wait() function)
MMC_ERR_STOP	–31	Forced stop error	Forced stop state due to the R_MMCIF_Control() function.
MMC_ERR_CMD	–32	Command issue error	MMCIF internal error (command issued)
MMC_ERR_BUFACC	–33	MMCIF buffer access error	MMCIF internal error (Illegal access to MMCIF buffer)
MMC_ERR_WRITE	–34	Write data error	MMCIF internal error (CRC error token status or end bit)
MMC_ERR_READ	–35	Read data error	MMCIF internal error (Read data CRC 16 or end bit)
MMC_ERR_RESPIND	–36	Response index error	MMCIF internal error (Response command index field value or check bits field value error)

MMC_ERR_RESP	–37	Response error	MMCIF internal error (Response or boot acknowledge error)
MMC_ERR_CRC_TOE	–38	CRC status timeout error	MMCIF internal error (When a CRC status token could not be received)
MMC_ERR_WRITE_TOE	–39	Write data timeout error	MMCIF internal error (When a busy state continued for longer than the stipulated period during a data write)
MMC_ERR_READ_TOE	–40	Read data timeout error	MMCIF internal error (Data could not be received within the stipulated period)
MMC_ERR_RESPB_TOE	–41	Response busy timeout error	MMCIF internal error (When the response busy state continued for longer than the stipulated period)
MMC_ERR_RESP_TOE	–42	Response timeout error	MMCIF internal error (When a response or boot acknowledge could not be received within the stipulated period)
MMC_ERR_FAST_IO	–43	Fast I/O response error	Fast I/O data error
MMC_ERR_CHANGE_BUS	–44	Bus test error	Error during bus test processing
MMC_ERR_SWITCH	–47	SWITCH command error	R1 response card status error (SWITCH_ERROR)
MMC_ERR_CSD_RLEN	–49	READ_BL_LEN error	CSD register[83:80] bits maximum read block length error
MMC_ERR_CSD_VER	–50	CSD Ver error	Version error for CSD version 5.0 or later. Note, however, that this MMCIF driver does not check for this error.
MMC_ERR_CSD_WLEN	–54	WRITE_BL_LEN error	CSD register[25:22] bits maximum write block length error
MMC_ERR_OUT_OF_RANGE	–80	Argument range error	R1 response card status error (OUT_OF_RANGE)
MMC_ERR_ADDRESS_ERROR	–81	Address error	R1 response card status error (ADDRESS_ERROR)
MMC_ERR_BLOCK_LEN_ERROR	–82	Block length error	R1 response card status error (BLOCK_LEN_ERROR)
MMC_ERR_ILLEGAL_COMMAND	–83	Illegal command error	R1 response card status error (ILLEGAL_COMMAND)
MMC_ERR_CBSY_ERROR	–87	Command error	MMCIF internal error (Command busy)
MMC_ERR_NO_RESP_ERROR	–88	No response error	MMCIF internal error (Response could not be received)
MMC_ERR_ADDRESS_BOUNDARY	–89	Buffer address error	Argument buffer address error The address does not fall on a 4-byte boundary
MMC_ERR_INTERNAL	–99	Internal error	Internal driver error

2.11 Callback Function

This driver calls the callback function specified by the user when the MMC protocol status interrupt, or MMC Card insertion interrupt occurs.

For information regarding how to register callback functions, see “3.15, R_MMCIF_Cd_Int()” and “3.16, R_MMCIF_IntCallback()”.
For the timing at which the callback function occurs, see “1.4.2.5 MMCIF Status Verification”.

2.12 Adding the FIT Module to Your Project

This driver must be added to each project in which it is used. Renesas recommends using “Smart Configurator” described in (1) or (3). However, “Smart Configurator” only supports some RX devices. Please use the methods of (2) or (4) for unsupported RX devices.

(1) Adding the FIT module to your project using “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 “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using “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 “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using “Smart Configurator” on CS+
By using the “Smart Configurator Standalone version” in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” 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 “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

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

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

while statement example :

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

for statement example :

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

do while statement example :

```
/* 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 */
```

3. API Functions

3.1 R_MMCIF_Open()

This is the first function called when this MMCIF driver API is used.

Format

```
mmc_status_t R_MMCIF_Open(
  uint32_t channel,
  void *p_mmc_WorkArea
)
```

Parameters

channel

Channel number — The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_WorkArea*

Pointer to a working area on a 4-byte boundary (area size: 164 bytes)

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*
MMC_ERR_CPU_IF	*Target microcontroller interface error*
MMC_ERR_ADDRESS_BOUNDARY	*Argument buffer address error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function acquires the MMCIF channel resource specified with the argument channel and initializes this MMCIF driver and the MMCIF channel. Also, this function exclusively acquires that MMCIF channel resource.

The working area is also retained until MMCIF driver close processing completes, and the application must not modify the working area contents.

Example

```
uint32_t       g_mmcif_work[164/sizesof(uint32_t)];

/* ==== Please add the processing to set the pins. ==== */

if (R_MMCIF_Open(MMC_CH0, &g_mmcif_work) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

The pins must be set up before this function is called. See section 4.4, MMC card Insertion and Power-On Timing, for details.

If this function does not complete successfully, do not call any library functions other than R_MMCIF_GetVersion(), R_MMCIF_Log() or R_MMCIF_Set_LogHdlAddress().

If this function does complete successfully, the card insertion interrupt may be enabled. If the MMC card insertion interrupt is used, after this function has run, use the R_MMCIF_Cd_Int() function to enable the card insertion interrupt.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

The microcontroller pin states do not change from before to after the execution of this function.

3.2 R_MMCIF_Close()

This function releases the resources being used by the MMCIF driver.

Format

```
mmc_status_t R_MMCIF_Close(
  uint32_t channel
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function terminates all MMCIF driver processing and releases the resources for the MMCIF channel specified in the argument channel.

That MMCIF channel is set to the module stop state.

After this function is called, the insertion interrupt will be in the disabled state.

The working area specified with the R_MMCIF_Open() function is not used after this function has been executed. This are may be used for other purposes.

Example

```
/* ==== Please add the processing to set the pins. ==== */

if (R_MMCIF_Close(MMC_CH0) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

The pins must be set up after this function is called. See section 4.5, MMC card Removal and Power-Off Timing, for details. Before running this function, driver open processing must be performed by the R_MMCIF_Open() function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.3 R_MMCIF_Get_CardDetection()

This function verifies the MMC Card insertion state.

Format

```
mmc_status_t R_MMCIF_Get_CardDetection(
  uint32_t channel
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

Return Values

MMC_SUCCESS	*The MMC_CD pin was at the low level or card detection was invalid.*
MMC_ERR	*The MMC_CD pin was at the high level.*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.hh.

Description

This function verifies the MMC card insertion state.

<When MMC_CFG_CHx_CD_ACTIVE == 1 (card detection enabled)>
If the MMC_CD pin is low, this function returns MMC_SUCCESS.
If the MMC_CD pin is high, this function returns MMC_ERR.
<When MMC_CFG_CHx_CD_ACTIVE == 0 (card detection disabled)>
This function will always return MMC_SUCCESS.

Example

```
mmc_status_t r_mmcif_pin_check_card_detection(uint32_t channel, uint8_t
derection)
{
#if (MMC_CFG_DRIVER_MODE & MMC_MODE_MMC)
   mmc_status_t      ret_old = MMC_ERR;
   mmc_status_t      ret_new = MMC_ERR;
   uint32_t          chat_cnt = MMC_CFG_CHAT_CNT;
   uint32_t          loop_cnt = MMC_CHAT_LOOP_CNT;

   /* ==== Check card insertion ==== */
   if (MMC_CD_REMOVE == derection)
   {
      ret_old = MMC_SUCCESS;
      ret_new = MMC_SUCCESS;
   }
   do
   {
      ret_new = R_MMCIF_Get_CardDetection(MMC_CH0);
      if (ret_new != ret_old) /* Status change */
      {
         if (((MMC_SUCCESS == ret_new) && (MMC_CD_INSERT == derection)) ||
             ((MMC_SUCCESS != ret_new) && (MMC_CD_REMOVE == derection)))
         {
            chat_cnt--;
            if (0 == chat_cnt)
            {
               return MMC_SUCCESS;
            }

            if (true != r_mmcif_pin_softwaredelay(1, MMC_DELAY_MILLISECS))
            {
               return MMC_ERR;
            }
         }
         else
         {
            chat_cnt = MMC_CFG_CHAT_CNT;
         }
      }
      else
      {
         chat_cnt = MMC_CFG_CHAT_CNT;
      }
      loop_cnt--;
   }
   while(0 != loop_cnt);
   return MMC_ERR;
#else
   return MMC_SUCCESS;
#endif /* (MMC_CFG_DRIVER_MODE & MMC_MODE_MMC) */
}
```

Special Notes

When using the card insertion detection function, pin setting is necessary after this function is executed. See section 4.4, MMC card Insertion and Power-On Timing, for details. Before running this function, driver open processing must be performed by the R_MMCIF_Open() function.

When using with card removal detection, pin setting is required before this function is executed. See section 4.5, MMC card Removal and Power-Off Timing, for details.

The MMC_CD pin, which is connected to the MMC card socket CD pin, is used as the MMC card insertion detection pin.

In this MMCIF, there is no hardware function to remove the chattering generated when an MMC card is inserted. Users should implement card detection processing that takes chattering into consideration based on this example.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

See section 4.6, Hardware Settings, for MMC_CD pin handling methods.

After an MMC card has been detected, the processing that provides the power supply voltage to the MMC card must be performed.

3.4 R_MMCIF_Mount()

This function initializes the MMC and transitions the state from the mountable state to the driver idle state.

Format

```
mmc_status_t R_MMCIF_Mount(
  uint32_t channel,
  mmc_cfg_t *p_mmc_Config,
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_Config*

Structure that holds the operating settings

mode: The operating mode

The application should set the operating mode to the logical OR of each of the types shown as macro definitions in Table 3.1, MMCIF Driver Operating Mode (mode).

voltage: The power supply voltage

Specify the voltage supplied to the MMC (For the setting values, see the macro definitions in Table 3.2, Supply Voltage (voltage). An MMC that cannot operate at the specified supply voltage will not be initialized. See section 1.4.2.3, Operating Voltage Settings When Initialization.

Table 3.1 MMCIF Driver Operating Mode (mode)

Type	Macro Definition	Value (Bits)	Definition
Status verification type	MMC_MODE_POLL	0x0000	Software polling
	MMC_MODE_HWINT	0x0001	Hardware interrupt
Data transfer type	MMC_MODE_SW	0x0000	Software transfers
	MMC_MODE_DMA*1*4	0x0002	DMAC transfers*3
	MMC_MODE_DTC*2*4	0x0004	DTC transfers*3
Media support type	MMC_MODE_MMC	0x0000	MMC card
	MMC_MODE_eMMC	0x0020	eMMC
MMC bus support type	MMC_MODE_1BIT*5	0x0100	MMC mode 1-bit bus
	MMC_MODE_4BIT*5	0x0400	MMC mode 4-bit bus
	MMC_MODE_8BIT*5	0x0800	MMC mode 8-bit bus

Notes: 1. DMAC control software must be provided separately.
2. DTC control software must be provided separately.
3. Software transfers are performed by the library functions used.
4. Do not set up MMC_MODE_DMA and MMC_MODE_DTC at the same time.
5. Select one of MMC_MODE_1BIT, MMC_MODE_4BIT, and MMC_MODE_8BIT.
   When MMC_MODE_8BIT is selected, bus connection will be attempted in the order 8-bit → 4-bit → 1-bit bus by the bus test.
   When MMC_MODE_4BIT is selected, bus connection will be attempted in the order 4-bit → 1-bit by the bus test.
   When MMC_MODE_1BIT is selected, the bus will be connected as a 1-bit bus by the bus test.

Table 3.2 Supply Voltage (voltage)

Supply Voltage [V]	Macro Definition	Value (Bits)
2.7-2.8	MMC_VOLT_2_8	0x00008000
2.8-2.9	MMC_VOLT_2_9	0x00010000
2.9-3.0	MMC_VOLT_3_0	0x00020000
3.0-3.1	MMC_VOLT_3_1	0x00040000
3.1-3.2	MMC_VOLT_3_2	0x00080000
3.2-3.3	MMC_VOLT_3_3	0x00100000
3.3-3.4	MMC_VOLT_3_4	0x00200000
3.4-3.5	MMC_VOLT_3_5	0x00400000
3.5-3.6	MMC_VOLT_3_6	0x00800000

Table 3.3 MMCIF Control Pins

Pin Name	eMMC 1-Bit Mode	eMMC 4-Bit Mode	eMMC 8-Bit Mode	MMC Card 1-Bit Mode	MMC Card 4-Bit Mode	MMC Card 8-Bit Mode
MMC_CLK	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control
MMC_CMD	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control
MMC_D0	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control
MMC_D1	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control
MMC_D2	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control
MMC_D3	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control	MMCIF control
MMC_D4	Not used	Not used	MMCIF control	Not used	Not used	MMCIF control
MMC_D5	Not used	Not used	MMCIF control	Not used	Not used	MMCIF control
MMC_D6	Not used	Not used	MMCIF control	Not used	Not used	MMCIF control
MMC_D7	Not used	Not used	MMCIF control	Not used	Not used	MMCIF control
MMC_CD	Not used	Not used	Not used	MMCIF control	MMCIF control	MMCIF control

Return Values

MMC_SUCCESS	*Successful operation*
MMC_SUCCESS_LOCKED_CARD	*Successful operation, and, furthermore, MMC is in the locked state*
Other than the above	*Error termination (See the error code for details.)*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function performs MMC mount processing. Execute this function after detecting an MMC card.

When the return value is MMC_SUCCESS, the MMC will have transitioned to the transfer state (tran) and MMC read and write access will be possible. When the return value is MMC_SUCCESS_LOCKED_CARD, the MMC will have transitioned to the transfer state (tran) but MMC read and write access will not be possible. The locked state must be cleared by other means.

Example

```
mmc_cfg_t      mmc_Config;

/* ==== Please add the processing to set the pins. ==== */

mmc_Config.mode = MMC_CFG_DRIVER_MODE;
mmc_Config.voltage = MMC_VOLT_3_3;
if (R_MMCIF_Mount(MMC_CH0, &mmc_Config) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

This MMCIF driver discriminates between High-speed mode and Backward-compatible mode when mounting.

The pins must be set up before executing this function. See section 4.4, MMC card Insertion and Power-On Timing, for details. Also, initialization using the R_MMCIF_Open() function is required before executing this function.

If this function returns an error, after setting the hardware to the unmounted state by calling the R_MMCIF_Unmount() function, perform the mount processing again.

After mounting has completed normally, unmounting must be performed before performing another mount operation.

When voltage in p_mmc_Config is set to an arbitrary value in the range 2.7 to 3.6 V, the output voltage will be taken to be in the 2.7 to 3.6 V range.

If the R_MMCIF_Cd_Int() function is used, set MMC_MODE_HWINT as the p_mmc_Config mode status.

3.5 R_MMCIF_Unmount()

This function clears the MMC mounted state and transitions from the transfer state to the state from which the driver can enter the idle state.

Format

```
mmc_status_t R_MMCIF_Unmount(
  uint32_t channel
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function performs MMC unmount processing. If this function is called in the transfer state, it initializes the MMC extended CSD register.

For MMC card, it switches to the MMC card removable state. Note that even if this function has been called and the MMC card mounted state has been cleared, the MMC card insertion interrupt and the MMC card insertion verification interrupt callback function remain enabled.

Example

```
if (R_MMCIF_Unmount(MMC_CH0) != MMC_SUCCESS)
{
   /* Error */
}

/* ==== Please add the processing to set the pins. ==== */
```

Special Notes

If the MMC card is removed after this function has been called, the pins must be set up. See section 4.5, MMC card Removal and Power-Off Timing, for details. Also, initialization using the R_MMCIF_Open() function is required before executing this function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.6 R_MMCIF_Read_Memory()

This function performs read processing.

Format

```
mmc_status_t R_MMCIF_Read_Memory(
  uint32_t channel,
  mmc_access_t *p_mmc_Access
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_Access*

Access information structure

**p_buff: Read buffer pointer*

This must be set to an address on a 4-byte boundary.

lbn: Read start block number

cnt: Block count

The maximum value that this argument may be set to is 65,535.

mode: Transfer mode (Does not need to be set and may not be changed)

rw_mode: Read mode

The setting values are shown in Table 3.4. For eMMC, specify MMC_PRE_DEF. For MMC card, specify MMC_OPEN_END.

Table 3.4 MMCIF Driver Read Mode (rw_mode)

Type	Macro Definition	Value (Bits)	Target MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC card
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

MMC_SUCCESS	*Successful operation*
Other than the above	*Error termination (See the error code for details.)*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

Reads the number of blocks of data specified by cnt in the argument p_mmc_Access starting at the block specified by lbn in the argument p_mmc_Access and stores that data in the buffer specified by p_buff in the argument p_mmc_Access.

If MMC card removal is detected at the start of this function's execution, processing is interrupted and processing is terminated and an error is returned.

If a forced stop request due to an R_MMCIF_Control() function MMC_SET_STOP (forced stop request) command is detected at the start of this function's execution, the forced stop is cleared and processing is terminated and an error is returned.

The following commands are used to read out the block data.

First block: READ_SINGLE_BLOCK command (CMD17)
Second and later blocks: READ_MULTIPLE_BLOCK command (CMD18)

Example

```
#define TEST_BLOCK_CNT     (4)
#define BLOCK_NUM       (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_clear(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_r_buff[0];
mmc_Access.lbn    = 0x10000000;
mmc_Access.cnt    = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Read_Memory(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

We recommend repeating the read operation when this function terminates with a read error.

If the number of blocks to be transferred exceeds 65,535, break up the read into multiple function calls. This issue requires care when this functionality is called from upper layer application programs such as the FAT file system.

Note that the size of a block is 512 bytes.

3.7 R_MMCIF_Read_Memory_Software_Trans()

This function performs read processing (software transfers).

Format

```
mmc_status_t R_MMCIF_Read_Memory_Software_Trans(
  uint32_t channel,
  mmc_access_t *p_mmc_Access
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_Access*

Access information structure

**p_buff: Read buffer pointer*

There are no address boundary restrictions. We recommend using an address that falls on a 4-byte boundary for faster processing.

lbn: Read start block number

cnt: Block count

The maximum value that this argument may be set to is 65,535.

mode: Transfer mode (Does not need to be set and may not be changed)

rw_mode: Read mode

The setting values are shown in Table 3.5. For eMMC, specify MMC_PRE_DEF. For MMC card, specify MMC_OPEN_END.

Table 3.5 MMCIF Driver Read Mode (rw_mode)

Type	Macro Definition	Value (Bits)	Target MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC card
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

```
MMC_SUCCESS             Successful operation
Other than the above    Error termination (See the error code for details.)
```

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

Reads the number of blocks of data specified by cnt in the argument p_mmc_Access starting at the block specified by lbn in the argument p_mmc_Access and stores that data in the buffer specified by p_buff in the argument p_mmc_Access.

Software transfer is used, regardless of the operating mode data transfer setting at command processing time.

If MMC card removal is detected at the start of this function’s execution, processing is interrupted and processing is terminated and an error is returned.

If a forced stop request due to an R_MMCIF_Control() function MMC_SET_STOP (forced stop request) command is detected at the start of this function’s execution, the forced stop is cleared and processing is terminated and an error is returned.

The following commands are used to read out the block data.

First block: READ_SINGLE_BLOCK command (CMD17)
Second and later blocks: READ_MULTIPLE_BLOCK command (CMD18)

Example

```
#define TEST_BLOCK_CNT  (4)
#define BLOCK_NUM       (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_set(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_w_buff[0];
mmc_Access.lbn    = 0x10000000;
mmc_Access.cnt    = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Write_Memory(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

We recommend repeating the read operation when this function terminates with a read error.

If the number of blocks to be transferred exceeds 65,535, break up the read into multiple function calls. This issue requires care when this functionality is called from upper layer application programs such as the FAT file system.

Note that the size of a block is 512 bytes.

3.8 R_MMCIF_Write_Memory()

This function performs write processing.

Format

```
mmc_status_t R_MMCIF_Write_Memory(
  uint32_t channel,
  mmc_access_t *p_mmc_Access
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_Access*

Access information structure

**p_buff: Write buffer pointer*

This must be set to an address on a 4-byte boundary.

lbn: Write start block number

cnt: Block count

The maximum value that this argument may be set to is 65,535.

mode: Transfer mode (Does not need to be set and may not be changed)

rw_mode: Write mode

The setting values are shown in Table 3.6. For eMMC, specify MMC_PRE_DEF. For MMC card, specify MMC_OPEN_END.

Table 3.6 MMCIF Driver Read Mode (rw_mode)

Type	Macro Definition	Value (Bits)	Target MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC card
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

```
MMC_SUCCESS             Successful operation
Other than the above    Error termination (See the error code for details.)
```

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

Writes the data from p_buff in the argument p_mmc_Access to an area with the number of blocks set by cnt in the argument p_mmc_Access. That area starts at the block specified by lbn in the argument p_mmc_Access.

If MMC card removal is detected at the start of this function's execution, processing is interrupted and processing is terminated and an error is returned.

If a forced stop request due to an R_MMCIF_Control() function MMC_SET_STOP (forced stop request) command is detected at the start of this function's execution, the forced stop is cleared and processing is terminated and an error is returned.

The following commands are used to write out the block data.

First block: WRITE_SINGLE_BLOCK command (CMD24)
Second and later blocks: WRITE_MULTIPLE_BLOCK command (CMD25)

Example

```
#define TEST_BLOCK_CNT  (4)
#define BLOCK_NUM       (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_set(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_w_buff[0];
mmc_Access.lbn    = 0x10000000;
mmc_Access.cnt    = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Write_Memory(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

We recommend repeating the write operation when this function terminates with a write error.

If the number of blocks to be transferred exceeds 65,535, break up the read into multiple function calls. This issue requires care when this functionality is called from upper layer application programs such as the FAT file system.

Note that the size of a block is 512 bytes.

3.9 R_MMCIF_Write_Memory_Software_Trans()

This function performs write processing (software transfers).

Format

```
mmc_status_t R_MMCIF_Write_Memory_Software_Trans(
  uint32_t channel,
  mmc_access_t *p_mmc_Access
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_Access*

Access information structure

**p_buff: Write buffer pointer*

There are no address boundary restrictions. We recommend using an address that falls on a 4-byte boundary for faster processing.

lbn: Write start block number

cnt: Block count

The maximum value that this argument may be set to is 65,535.

mode: Transfer mode (Does not need to be set and may not be changed)

rw_mode: Write mode

The setting values are shown in Table 3.7. For eMMC, specify MMC_PRE_DEF. For MMC card, specify MMC_OPEN_END.

Table 3.7 MMCIF Driver Read Mode (rw_mode)

Type	Macro Definition	Value (Bits)	Target MMC
Open-ended	MMC_OPEN_END	0x00000000	MMC card
Pre-defined	MMC_PRE_DEF	0x00000001	eMMC

Return Values

```
MMC_SUCCESS             Successful operation
Other than the above    Error termination (See the error code for details.)
```

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

Writes the data from p_buff in the argument p_mmc_Access to an area with the number of blocks set by cnt in the argument p_mmc_Access. That area starts at the block specified by lbn in the argument p_mmc_Access.

Software transfer is used, regardless of the operating mode data transfer setting at command processing time.

If MMC card removal is detected at the start of this function’s execution, processing is interrupted and processing is terminated and an error is returned.

If a forced stop request due to an R_MMCIF_Control() function MMC_SET_STOP (forced stop request) command is detected at the start of this function’s execution, the forced stop is cleared and processing is terminated and an error is returned.

The following commands are used to write out the block data.

First block: WRITE_SINGLE_BLOCK command (CMD24)
Second and later blocks: WRITE_MULTIPLE_BLOCK command (CMD25)

Example

```
#define TEST_BLOCK_CNT  (4)
#define BLOCK_NUM       (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_set(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_w_buff[0];
mmc_Access.lbn    = 0x10000000;
mmc_Access.cnt    = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Write_Memory_Software_Trans(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

We recommend repeating the write operation when this function terminates with a write error.

If the number of blocks to be transferred exceeds 65,535, break up the read into multiple function calls. This issue requires care when this functionality is called from upper layer application programs such as the FAT file system.

Note that the size of a block is 512 bytes.

3.10 R_MMCIF_Control()

This function performs driver control processing.

Format

```
mmc_status_t R_MMCIF_Control(
  uint32_t channel,
  mmc_cmd_t *p_mmc_Cmd
)
```

Parameters

channel

Channel number — The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_Cmd*

Control information structure

cmd: Command macro definition

mode: Mode

**p_buff: Transfer buffer pointer*

size: Transfer size

Return Values

MMC_SUCCESS	*Successful operation*
Other than the above	*Error termination (See the error code for details.)*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This is an MMC control utility.

Table 3.8, Commands, lists the commands that can be controlled. These commands are described individually starting on the following page.

Table 3.8 Commands

Command macro Definition cmd	Mode mode	Transfer Content *p_buff	Transfer Size size	Control Performed
MMC_SET_STOP (Forced stop request command)	Setting invalid	Setting invalid	Setting invalid	Transitions to the forced stop request state When a forced stop request command due to this function call is issued during read or write processing, a transfer processing forced stop is requested.

Example

Examples are shown for each command on the following pages.

Special Notes

Before running this function, driver open processing must be performed by the R_MMCIF_Open() function and initialization by the R_MMCIF_Mount() function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

(a) **MMC_SET_STOP**

This function forcibly stops read/write processing.

Return Values

MMC_SUCCESS	*Successful operation*

Description

This function requests a forced stop and transitions the MMCIF driver to the forced stop state. This function can be called from within an interrupt handler when an application program wants to stop processing.

When there is a data transfer to or from the MMC in progress, a CMD12 is issued to transition the MMC to the transfer state (tran), the read/write processing for the transfer in progress is forcibly terminated, and an error is returned.

Note that if this function is executed during write processing, a CMD12 is issued and the MMC may transition to the busy state. As a result, there are cases where an error is returned when a next read or write function is called. In such cases, we recommend performing the read or write processing again. If this occurs during a write, it will be necessary to wait until the MMC is in the ready state.

Also, if a forced stop request is issued with a timing such that it occurs after a transfer has completed, the processing will return with the MMC still in the forced stop request state.

Example

```
mmc_cmd_t       mmc_Cmd;

mmc_Cmd.cmd = MMC_SET_STOP;
mmc_Cmd.mode = 0;
mmc_Cmd.p_buff = 0;
mmc_Cmd.size = 0;

if (R_MMCIF_Control(MMC_CH0, &mmc_Cmd) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

When a forced stop is performed during write processing, the MMC data is not guaranteed.

The following are the forced stop request checkpoints in library functions.

(1) After read/write processing has started, before any commands are issued to the MMC.
(2) During software transfers, after the completion of transfer of a 512-byte block unit and before transfer of the next block.
(3) Forced stop requests are always accepted during DMAC or DTC transfers.

Also, a forced stop request clear is performed in the following cases.

(1) When forced stop processing is performed during execution of the R_MMCIF_Read_Memory() or R_MMCIF_Write_Memory()* function.
(2) When the R_MMCIF_Read_Memory() or R_MMCIF_Write_Memory()* function is called in the forced stop state. In this case, the forced stop request is detected at the start of processing, the processing is stopped, and an error is returned.

Note: * The R_MMCIF_Read_Memory_Software_Trans() and R_MMCIF_Write_Memory_Software_Trans() functions operate similarly.

3.11 R_MMCIF_Get_ModeStatus()

This function acquires the transfer mode status.

Format

```
mmc_status_t R_MMCIF_Get_ModeStatus(
  uint32_t channel,
  uint8_t *p_mode
)
```

Parameters

channel

Channel number — The number of the MMCIF channel used (numbering starts at 0)

**p_mode*

Mode status information storage pointer (1 byte). See the macro definitions in Table 3.1, MMCIF Driver Operating Mode (mode) for the values of this parameter.

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function acquires the transfer mode status and stores it in the mode status information storage pointer.

Example

```
uint8_t * p_mode;

if(R_MMCIF_Get_ModeStatus(MMC_CH0, p_mode) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.12 R_MMCIF_Get_CardStatus()

This function acquires the card status information.

Format

```
mmc_status_t R_MMCIF_Get_CardStatus(
  uint32_t channel,
  mmc_card_status_t *p_mmc_CardStatus
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_CardStatus*

Card status information structure pointer

card_sector_size: User area block count

csd_structure: the CSD_STRUCTURE[127:126] field in the CSD register

0: CSD version No. 1.0
1: CSD version No. 1.1
2: CSD version No. 1.2 (Version 4.1-4.2-4.3)
3: Version is coded in the CSD_STRUCTURE byte in the EXT_CSD register

speed_mode: Speed mode

<Corresponding MMC support mode>	<Current transfer mode>
bit 5-4	bit 1-0
00: Backward-compatible mode	00: Backward-compatible mode
01: High-speed mode 26 MHz (Maximum)	01: High-speed mode 26 MHz (Maximum)
11: High-speed mode 52 MHz (Maximum)	11: High-speed mode 52 MHz (Maximum)

csd_spec: the SPEC_VERS[125:122] field in the CSD register

0: MMC_SPEC_10 /* MMC system spec: 1.0-1.2 */
1: MMC_SPEC_14 /* MMC system spec: 1.4 */
2: MMC_SPEC_20 /* MMC system spec: 2.0-2.2 */
3: MMC_SPEC_30 /* MMC system spec: 3.1-3.2-3.31 */
4: MMC_SPEC_40 /* MMC system spec: 4.0-4.1-4.2-4.3-4.4-4.41 */

if_mode: Data bus width mode

0: 1-bit
1: 4-bit
2: 8-bit

density_type: Access mode (OCR bit[30:29])

0: Byte mode
1: Sector mode

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function gets the MMC card status information and stores it in a card status information structure.

Example

```
mmc_card_status_t    mmc_CardStatus;

if (R_MMCIF_Get_CardStatus(MMC_CH0, &mmc_CardStatus) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.13 R_MMCIF_Get_CardInfo()

This function acquires the MMC register information.

Format

```
mmc_status_t R_MMCIF_Get_CardInfo(
  uint32_t channel,
  mmc_card_reg_t *p_mmc_CardReg
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

**p_mmc_CardReg*

MMC register information structure pointer

ocr[1]

OCR information

cid[4]

CID information

csd[4]

CSD information

dsr[1]

DSR information

rca[1]

RCA information

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function acquires the MMC register information and stores it in the MMC register information structure.

Example

```
mmc_card_reg_t    mmc_CardReg;

if (R_MMCIF_Get_CardInfo(MMC_CH0, &mmc_CardReg) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.14 R_MMCIF_Int_Handler0

This function is the MMCIF interrupt handler.

Format

```
void R_MMCIF_Int_Handler0(
  void *vect
)
```

Parameters

**vect*

Vector table

Return Values

None

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function is the interrupt handler for the MMCIF driver.

This function is already embedded in the system as the interrupt factor processing routine for the MMCIF.

When an MMC card insertion interrupt setup callback function and a status verification interrupt callback function are registered, these callback functions will be called from this function.

Example

Since this function is already embedded in the system, no user settings are required.

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

When other channels are used, interrupt handlers must be created for each channel in a similar manner. (Example: R_MMCIF_Int_Handler1() for channel 1)

3.15 R_MMCIF_Cd_Int()

This function sets up the insertion interrupt (including registering the insertion interrupt callback function).

Format

```
mmc_status_t R_MMCIF_Cd_Int(
  uint32_t channel,
  int32_t enable,
  mmc_status_t (*callback)(int32_t)
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

enable: Specifies enable/disable of the MMC card insertion interrupt.

When MMC_CD_INT_ENABLE is specified, the MMC card insertion interrupt is enabled.

When MMC_CD_INT_DISABLE is specified, the MMC card insertion interrupt is disabled.

*(*callback)(int32_t): Callback function to be registered*

If a null pointer is specified, no callback function is registered. If a callback function is to be used, execute this function to register the callback function before an MMC card is inserted.

The MMC_CD pin detection state is stored in the (int32_t).

0: MMC_CD_INSERT (A falling edge on the MMC_CD pin was detected)

1: MMC_CD_REMOVE (A rising edge on the MMC_CD pin was detected)

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function sets up the MMC card insertion interrupt and registers a callback function.

The callback function registered by this function is called as a subroutine from the interrupt handler when an MMC card insertion interrupt occurs.

Note that the MMC card insertion state can be verified with the R_MMCIF_Get_CardDetection() function regardless of the enabled/disabled state of the MMC card insertion interrupt.

Example

```
uint32_t g_cd_int;

/* Callback function */
mmc_status_t r_mmcif_cd_callback(int32_t cd)
{
   g_cd_int = 1;
   /* ==== Disable card detect interrupt ==== */
   if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_DISABLE, 0) != MMC_SUCCESS)
   {
      /* Error */
   }
   return MMC_SUCCESS;
}

/* main */
void main(void)
{
   if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_ENABLE, r_mmcif_cd_callback) !=
MMC_SUCCESS)
   {
      /* Error */
   }

   /* ==== Check card insertion ==== */
   g_cd_int = 0;
   while (1)
   {
      if (1 == g_cd_int)
      {
         g_cd_int = 0;
         if (r_mmcif_pin_check_card_detection(MMC_CH0, MMC_CD_INSERT) ==
MMC_SUCCESS)
         {
            /* ==== Enable card detect interrupt ==== */
            if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_ENABLE,
r_mmcif_cd_callback) != MMC_SUCCESS)
            {
               /* Error */
            }
            break;
         }
         else
         {
            /* ==== Enable card detect interrupt ==== */
            if (R_MMCIF_Cd_Int(MMC_CH0, MMC_CD_INT_ENABLE,
r_mmcif_cd_callback) != MMC_SUCCESS)
            {
               /* Error */
            }
         }
      }
      else
      {
         /* Do nothing. */
      }
  }
```

```
}
```

Special Notes

To enable card detection, set #define MMC_CFG_CHx_CD_ACTIVE to 1.

Initialization by the R_MMCIF_Open() function is required before this function is executed.

After this function has been executed, the MMC card insertion interrupt will be caused by an MMC card insertion.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.16 R_MMCIF_IntCallback()

This function registers an MMC protocol status interrupt callback function.

Format

```
mmc_status_t R_MMCIF_IntCallback(
  uint32_t channel,
  mmc_status_t (*callback)(int32_t)
)
```

Parameters

channel

Channel number The number of the MMCIF channel used (numbering starts at 0)

*(*callback)(int32_t): Callback function to be registered*

If a null pointer is specified, no callback function is registered. If a callback function is to be used, register a callback function before the R_MMCIF_Mount() function is executed.

The value 0 is always stored in (int32_t).

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function registers an MMC protocol status interrupt callback function.

The callback function registered by this function is called as a subroutine from the interrupt handler when an interrupt occurs due to a change in the MMC protocol status (ACCIO or ERRIO).

Example

```
/* Callback function */
mmc_status_t r_mmcif_callback(int32_t cd)
{
   /* ACCIO, ERRIO */
   return MMC_SUCCESS;
}

if (R_MMCIF_IntCallback(MMC_CH0, r_mmcif_callback) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Initialization by the R_MMCIF_Open() function is required before this function is executed.

The stack wait state clear operation and other processing is performed in registered callback function.

The callback function registered by this function differs from the MMC card insertion interrupt callback function.

The callback function registered by this function is not called when an MMC card insertion interrupt occurs.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.17 R_MMCIF_Get_ErrCode()

This function acquires the driver error codes.

Format

```
mmc_status_t R_MMCIF_Get_ErrCode(
  uint32_t channel
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

Return Values

Error code	*See the error code documentation.*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function returns the error codes that occur when the R_MMCIF_Mount(), R_MMCIF_Read_Memory(), and R_MMCIF_Write_Memory() function* are executed. Note that the error code is cleared when a library function is executed again.

Note: * The R_MMCIF_Read_Memory_Software_Trans() and R_MMCIF_Write_Memory_Software_Trans() functions operate similarly.

Example

```
mmc_cfg_t      mmc_Config;
mmc_status_t   error_code = MMC_SUCCESS;

/* ==== Please add the processing to set the pins. ==== */

mmc_Config.mode = MMC_CFG_DRIVER_MODE;
mmc_Config.voltage = MMC_VOLT_3_3;
if (R_MMCIF_Mount(MMC_CH0, &mmc_Config) != MMC_SUCCESS)
{
   /* Error */
   error_code = R_MMCIF_Get_ErrCode(MMC_CH0);
}
```

Special Notes

Initialization by the R_MMCIF_Open() function is required before this function is executed.

Use this function when an application program needs to acquire the MMCIF driver error code.

3.18 R_MMCIF_Get_BuffRegAddress()

This function acquires the address of the MMCIF data register (CEDATA).

Format

```
mmc_status_t R_MMCIF_Get_BuffRegAddress(
  uint32_t channel,
  uint32_t *p_reg_buff
)
```

Parameters

channel

Channel number — The number of the MMCIF channel used (numbering starts at 0)

**p_reg_buff*

Data register (CEDATA) address pointer

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function acquires the address of the data register (CEDATA) and stores it in a buffer.

This function is used, for example, when specifying the data register address when using DMAC or DTC transfers.

Example

```
uint32_t    reg_buff = 0;

if (R_MMCIF_Get_BuffRegAddress(MMC_CH0, &reg_buff) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Initialization by the R_MMCIF_Open() function is required before this function is executed.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.19 R_MMCIF_Get_ExtCsd()

This function acquires the MMC extended CSD information.

Format

```
mmc_status_t R_MMCIF_Get_ExtCsd(
  uint32_t channel,
  uint32_t *p_ext_csd_buffer
)
```

Parameters

channel

Channel number — The number of the MMCIF channel used (numbering starts at 0)

**p_ext_csd_buffer*

Extended CSD receive buffer pointer (512 bytes)

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function stores the MMC extended CSD information in the argument p_ext_csd_buffer.

Example

```
uint8_t  g_mmc_extcsd[512];

if (R_MMCIF_Get_ExtCsd(MMC_CH0, &g_mmc_extcsd[0]) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.20 R_MMCIF_1ms_Interval()

This function increments the MMCIF driver's internal timer counter.

Format

```
void R_MMCIF_1ms_Interval(
  void
)
```

Parameters

None

Return Values

None

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

The internal timer counter is incremented each time this function is called.

Example

```
uint32_t    g_cmt_channel;

void r_cmt_callback(void * pdata)
{
   uint32_t channel;

   channel = *((uint32_t *)pdata);
   if (channel == g_cmt_channel)
   {
      R_MMCIF_1ms_Interval();
   }
}

main()
{
    /* Create CMT timer */
    R_CMT_CreatePeriodic(1000, &r_cmt_callback, &g_cmt_channel);      /* 1ms */
}
```

Special Notes

The application must call this function once each millisecond. However, this is not required if the timer functionality has been replaced by the r_mmcif_dev_int_wait() and r_mmcif_dev_wait() functions in r_mmcif_dev.c.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.21 R_MMCIF_Set_DmacDtc_Trans_Flg()

This function sets the DMAC/DTC transfer complete flag.

Format

```
mmc_status_t R_MMCIF_Set_DmacDtc_Trans_Flg(
  uint32_t channel,
  uint32_t flg
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

flg

DMAC/DTC transfer complete flag	MMC_SET_TRANS_STOP

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error (channel error)*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function sets the DMAC/DTC transfer complete flag.

Table 3.9, DMAC Transfer/DTC Transfer Flag Processing Methods, lists the processing methods for handling the DMAC/DTC transfer complete flag.

Note that the DMAC/DTC transfer complete flag processing method differs depending on the transfer state.

For the DMAC, the application should set the MMC_SET_TRANS_STOP in the interrupt handler called when a DMAC transfer completes and call this function.

For the DTC, no user processing is required to set MMC_SET_TRANS_STOP in the MMCIF MBFAI interrupt handler.

If an error occurs during a transfer, the user code must set MMC_SET_TRANS_STOP, regardless of whether DMAC or DTC is used and then call this function.

Table 3.9 DMAC Transfer/DTC Transfer Flag Processing Methods

Data Transfer	On Successful Termination	On Error Termination
DMAC transfer	Transfer in progress The user code should execute R_MMCIF_Set_DmacDtc_Trans_Flg() in the interrupt handler called when a DMAC transfer completes and set the transfer complete state.	Transfer in progress The user code should execute R_MMCIF_Set_DmacDtc_Trans_Flg() and set the transfer complete state.
DTC transfer	Transfer complete (Transfer complete processing is performed in the DTC handler) No user processing is required.	As above

Example

<When the DMAC transfer completes successfully>

```
void r_dmaca_callback(void)
{
   R_MMCIF_Set_DmacDtc_Trans_Flg(MMC_CH0, MMC_SET_TRANS_STOP);
}
```

<When the DMAC transfer terminates with an error>

```
#define TEST_BLOCK_CNT  (4)
#define BLOCK_NUM       (512)

mmc_access_t   mmc_Access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

test_data_clear(&g_def_buf[0], TEST_BLOCK_CNT);
mmc_Access.p_buff = (uint8_t *)&g_test_r_buff[0];
mmc_Access.lbn    = 0x10000000;
mmc_Access.cnt    = TEST_BLOCK_CNT;
mmc_Access.rw_mode= MMC_PRE_DEF;

if(R_MMCIF_Read_Memory(MMC_CH0, &mmc_Access) != MMC_SUCCESS)
{
   /* Error */
   R_MMCIF_Set_DmacDtc_Trans_Flg(MMC_CH0, MMC_SET_TRANS_STOP);
}
```

Special Notes

Both initialization processing by the R_MMCIF_Open() function and mount processing by the R_MMCIF_Mount() function are required prior to executing this function.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

3.22 R_MMCIF_Set_LogHdlAddress()

This function sets the LONGQ FIT module handler address.

Format

```
mmc_status_t R_MMCIF_Set_LogHdlAddress(
  uint32_t user_long_que
)
```

Parameters

user_long_que

LONGQ FIT module handler address

Return Values

MMC_SUCCESS	*Successful operation*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function sets the LONGQ FIT module handler address in the MMCIF driver.

Example

```
#define MMC_USER_LONGQ_MAX    (8)/* Max error log count*/
#define MMC_USER_LONGQ_BUFSIZE   (MMC_USER_LONGQ_MAX * 4)
                                 /* Error log buffer size */
#define MMC_USER_LONGQ_IGN_OVERFLOW (1)/* Ignore_overflow of error log
buffer.*/

uint32_t             g_mmc_user_longq_buf[MMC_USER_LONGQ_BUFSIZE];
                                 /* Error log buffer */
static longq_hdl_t   p_mmc_user_long_que; /* LongQ handler */
longq_err_t          err = LONGQ_SUCCESS;
uint32_t             user_long_que = 0;

err = R_LONGQ_Open(g_mmc_user_longq_buf,
                   MMC_USER_LONGQ_BUFSIZE,
                   MMC_USER_LONGQ_IGN_OVERFLOW,
                   &p_mmc_user_long_que);
if (LONGQ_SUCCESS != err)
{
   /* Error */
}
user_long_que = (uint32_t)p_mmc_user_long_que;
if (R_MMCIF_Set_LogHdlAddress(user_long_que) != MMC_SUCCESS)
{
   /* Error */
}
```

Special Notes

This function performs the preparatory processing required to acquire an error log using the LONGQ FIT module. This processing should be performed before the R_MMCIF_Open() function is called.

The LONGQ FIT module needs to be embedded in the application as a separate operation.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

If the MMC_CFG_LONGQ_ENABLE disable and this function is called, this function does nothing.

3.23 R_MMCIF_Log()

This function acquires an error log.

Format

```
uint32_t R_MMCIF_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*

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function acquires an error log.

To terminate error log acquisition, call this function.

Example

```
#define USER_DRIVER_ID        (1)
#define USER_LOG_MAX       (63)
#define USER_LOG_ADR_MAX   (0x00001fff)

mmc_cfg_t      mmc_Config;

/* ==== Please add the processing to set the pins. ==== */

mmc_Config.mode = MMC_CFG_DRIVER_MODE;
mmc_Config.voltage = MMC_VOLT_3_3;
if (R_MMCIF_Mount(MMC_CH0, &mmc_Config) != MMC_SUCCESS)
{
   /* Error */
   R_MMCIF_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
}
```

Special Notes

The LONGQ FIT module needs to be embedded in the application as a separate operation.

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

If the MMC_CFG_LONGQ_ENABLE disable and this function is called, this function does nothing.

3.24 R_MMCIF_GetVersion()

This function acquires the version information for the driver.

Format

```
uint32_t R_MMCIF_GetVersion(
  void
)
```

Parameters

None

Return Values

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

Properties

A prototype declaration for this function appears in r_mmcif_rx_if.h.

Description

This function returns the driver version information.

Example

```
uint32_t version;
version = R_MMCIF_GetVersion();
```

Special Notes

Note that the error code cannot be acquired with the R_MMCIF_Get_ErrCode() function.

4. Pin Setting

To use the MMCIF FIT module, assign input/output signals of the peripheral function to pins with the multi-function pin controller (MPC).
When performing the pin setting in the e^2 studio, the Pin Setting feature of the Smart Configurator can be used. When using the Pin Setting feature, a source file is generated according to the option selected in the Pin Setting window in the Smart Configurator. Then pins are configured by calling the function defined in the source file. Refer to Table 4.1 for details.
The pin assignment is referred to as the “Pin Setting” in this document. Also, GPIO control is required. Refer to 4.4, MMC card Insertion and Power-On Timing, and 4.5, MMC card Removal and Power-Off Timing, and create appropriate program code to provide this processing.

Table 4.1 Function Output by the Smart Configurator

Function to be Output	Function
R_MMCIF_PinSetInit()	Performs initialization of the MMCIF pins. After execution, only MMC_CD pin is valid.
R_MMCIF_PinSetTransfer()	Sets MMCIF pins to MMC command issuance possible state. After execution, all MMCIF pins are valid.
R_MMCIF_PinSetDetection()	Sets MMCIF pins to MMC command issuance impossible state. After execution, only MMC_CD pin is valid.
R_MMCIF_PinSetEnd()	Sets to MMCIF control disabled state. After execution, all MMCIF pins are invalid.

4.1 Pins setting of MMC bus 1 bit communication

Even if the MMC bus used for communication is 1 bit, please set the pin of the MMC bus to 4 bits. If generates the MMC command without controlling the MCC bus other than the MMC_D0 pin, there is the possibility that the MMC transitions the SPI mode.

4.2 Setting of MMC card power control pin

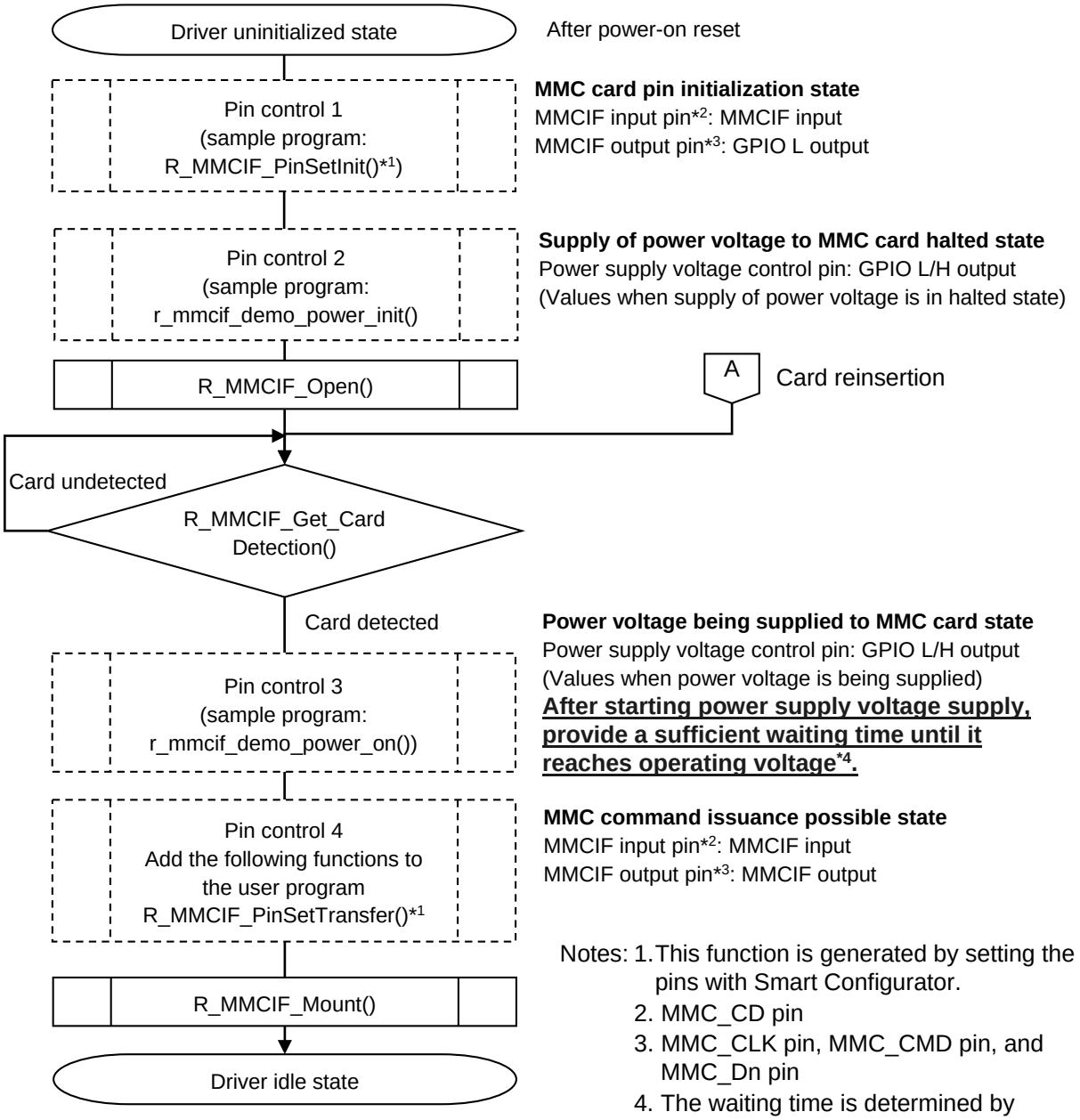
The pin setting function of the Smart Configurator does not include the power supply voltage control pin of the MMC card, create it separately.

4.3 Setting of MMC card MMC reset pin

Since the MMC reset pin (MMC_RES#) is not controlled by the MMCIF driver, create it separately.

4.4 MMC card Insertion and Power-On Timing

Figure 4.1 and Table 4.2 show the control procedure. Perform the MMC card insertion procedure after successful operation of the R_MMCIF_Open() function and when the supply of power voltage to the MMC card is in the halted state and the MMCIF output pin is in the L output state.



Notes: 1. This function is generated by setting the pins with Smart Configurator.
2. MMC_CD pin
3. MMC_CLK pin, MMC_CMD pin, and MMC_Dn pin
4. The waiting time is determined by the configuration of the MMC card power supply circuit.

Figure 4.1 MMC card Insertion and Power-On Timing

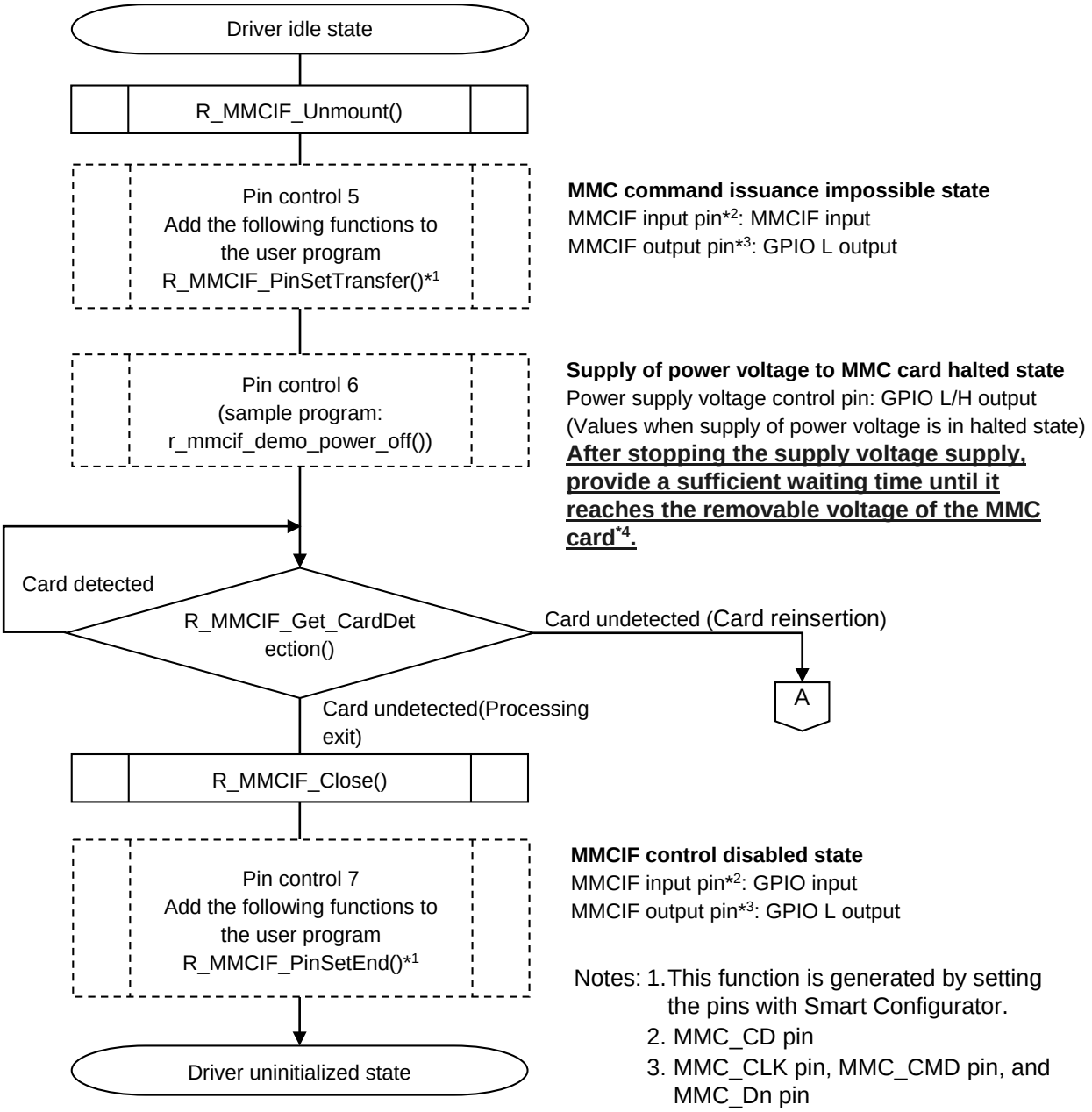
Table 4.2 User Setting Method at MMC Card Insertion

Processing	Target Pin	Pin Settings	Subsequent Pin State
Pin control 1	MMCIF input pin*1	PMR setting: General I/O port PCR setting: Input pull-up resistor disabled*3 PDR setting: Input MPC setting: MMCIF PMR setting: Peripheral module	MMCIF input (MMC Card detection possible state)
	MMCIF output pin*2	PMR setting: General I/O port DSCR setting: High-drive output PCR setting: Input pull-up resistor disabled*3 PODR setting: L output PDR setting: Output MPC setting: Hi-z	GPIO L output
Pin control 2	Power supply voltage control pin	PMR setting: General I/O PCR setting: Input pull-up resistor disabled*4 PODR setting: L output/H output (output of value based on power voltage supplied/halted state) PDR setting: Output	GPIO L/H output (supply of power voltage halted state)
Pin control 3	Power supply voltage control pin	PODR setting: L output/H output (output of value based on power voltage supply state)	GPIO L/H output (supply of power voltage halted state)
Pin control 4	MMCIF input pin*1	MPC setting: MMCIF PMR setting: Peripheral module	MMCIF input
	MMCIF output pin*2	MPC setting: MMCIF PMR setting: Peripheral module	MMCIF output (MMC command issuance possible state)

Notes: 1. MMC_CD pin
2. MMC_CLK pin, MMC_CMD pin, and MMC_Dn pin
3. It is assumed that the pin will be pulled-up external to the microcontroller, so the microcontroller's integrated
pull-up resistor is disabled.
4. Review the setting to match the details of the system.

4.5 MMC card Removal and Power-Off Timing

Figure 4.2 and Table 4.4 show the control procedure. Perform the MMC card removal procedure after successful operation of the R_MMCIF_Unmount() function in the driver idle state and when the supply of power voltage to the MMC card is in the halted state. An equivalent procedure should be used to halt supply of the power voltage in cases where the MMC card is removed unexpectedly.



Notes: 1. This function is generated by setting the pins with Smart Configurator.
2. MMC_CD pin
3. MMC_CLK pin, MMC_CMD pin, and MMC_Dn pin
4. The waiting time is determined by the configuration of the MMC card power supply circuit.

Figure 4.2 MMC card Removal and Power-Off Timing

Table 4.3 User Setting Method at MMC Card Removal

Processing	Target Pin	Pin Settings	Subsequent Pin State
Pin control 5	MMCIF input pin*1	MPC setting: MMCIF PMR setting: Peripheral module	MMCIF input
	MMCIF output pin*2	PMR setting: General I/O port MPC setting: Hi-z	GPIO L output
Pin control 6	Power supply voltage control pin	PODR setting: L output/H output (output of value based on power voltage supply halted state)	GPIO L/H output (supply of power voltage halted state)
Pin control 7	MMCIF input pin*1	PMR setting: General I/O port MPC setting: Hi-z	GPIO input
	MMCIF output pin*2	PMR setting: General I/O port MPC setting: Hi-z	GPIO L output

Notes: 1. MMC_CD pin
2. MMC_CLK pin, MMC_CMD pin, and MMC_Dn pin

4.6 Hardware Settings

This MMCIF driver uses the microcontroller's internal MMCIF and performs 1-bit, 4-bit or 8-bit bus MMC mode control.

The number of MMC that can be connected is one per channel.

4.6.1 Sample Hardware Configuration

Section 4.6.2, MMC Socket (removable media: MMC card), shows the circuit diagram for MMC card.

Section 4.6.3, MMC (Embedded MultiMediaCard: eMMC), shows the circuit diagram when an eMMC is used.

See the JEDEC Standard JESD84 for pull-up resistor values. Users should consider the addition of damping resistors and capacitors for circuit matching if high-speed operation is expected.

4.6.1.1 Pin Descriptions

MMC_CLK Pin

Since there is no stipulation regarding pulling up this pin in the JEDEC Standard JESD84, there is no stipulation regarding that here.

MMC_CMD Pin

The MMC sets the CMD line to open drain in card identification mode. Therefore, the JEDEC Standard JESD84 stipulates the pull-up resistor values for CMD and DAT0 to DAT7 separately. The user must determine pull-up resistor values that match the environment used.

MMC_CD Pin

When MMC card are used, configure the circuit for this pin so that a high level is input to the MMC_CD pin when no card is inserted, and a low level is input when a card is inserted.

Set up control for the MMC_CD pin in r_mmcif_rx_config.h. If this control is disabled, the MMC_CD pin can be used for peripheral functions other than the MMC.

Table 4.4 MMC_CD Pin #define Definition Setting in r_mmcif_rx_config.h

#define MMC_CFG_CHx_CD_ACTIVE	MMC_CD pin	Target MMC
(0)	Not controlled	eMMC
(1)	Controlled	MMC card

MMC_RES# Pin

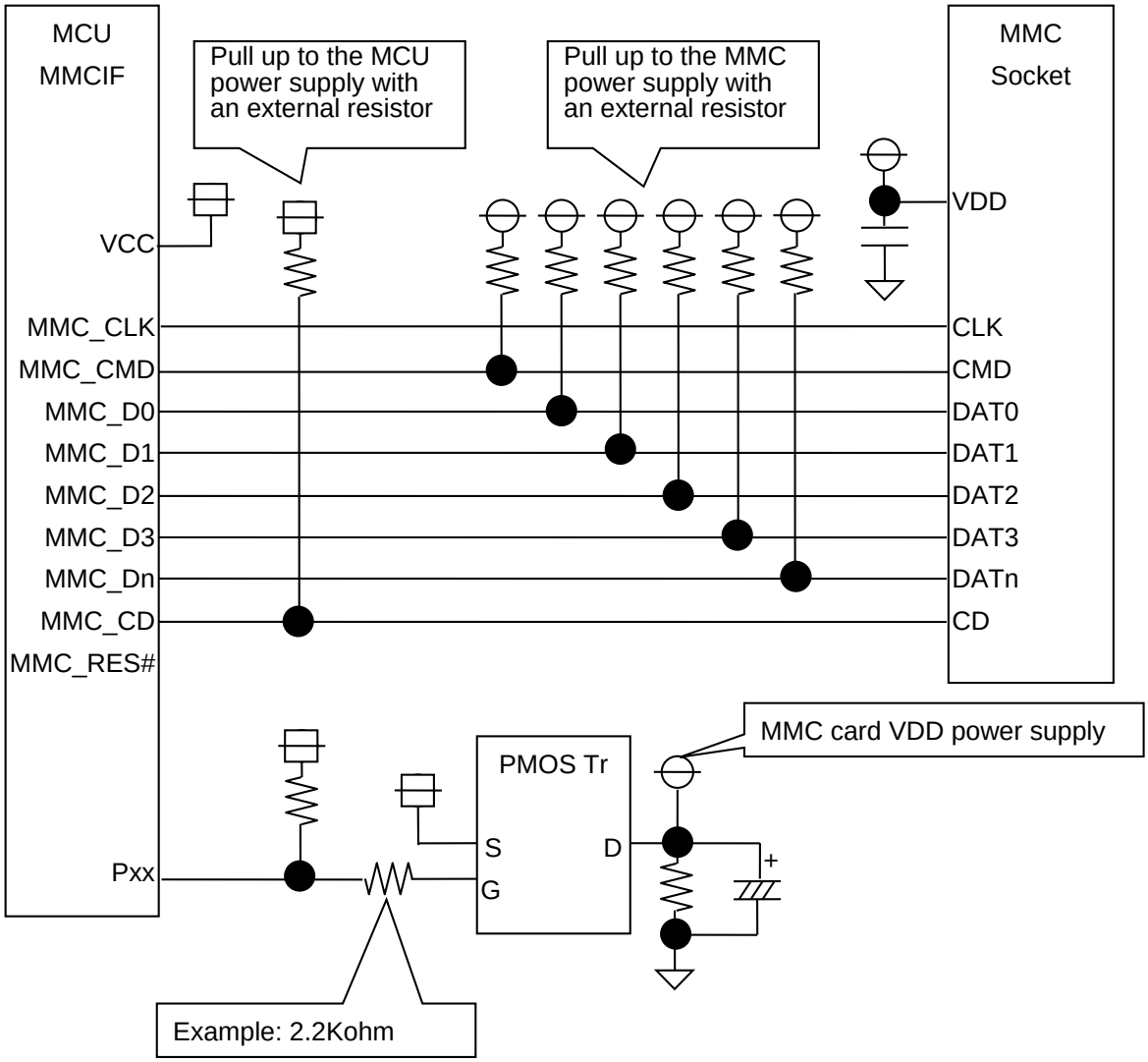
The MMC_RES pin is not controlled. It may be used for peripheral functions other than the MMC. Note, however, that the Bit[1:0] field (RST_n_FUNCTION[162]) in the Extended CSD must be used with the value 0x0 (the default).

MMC Power Supply Control Pins

When MMC power supply control is required, construct an external circuit using PMOS transistors and other devices. In particular, be sure to attach an adequately large capacitor and discharge resistor between power supply and ground. When the microcontroller pins are directly controlling PMOS transistors with large gate capacitances, refer to the microcontroller electrical characteristics (in particular, the allowable output low level current and allowable output high level current) and insert current control resistors between the microcontroller pins and the PMOS transistor gates.

4.6.2 MMC Socket (Removable Media: MMC Card)

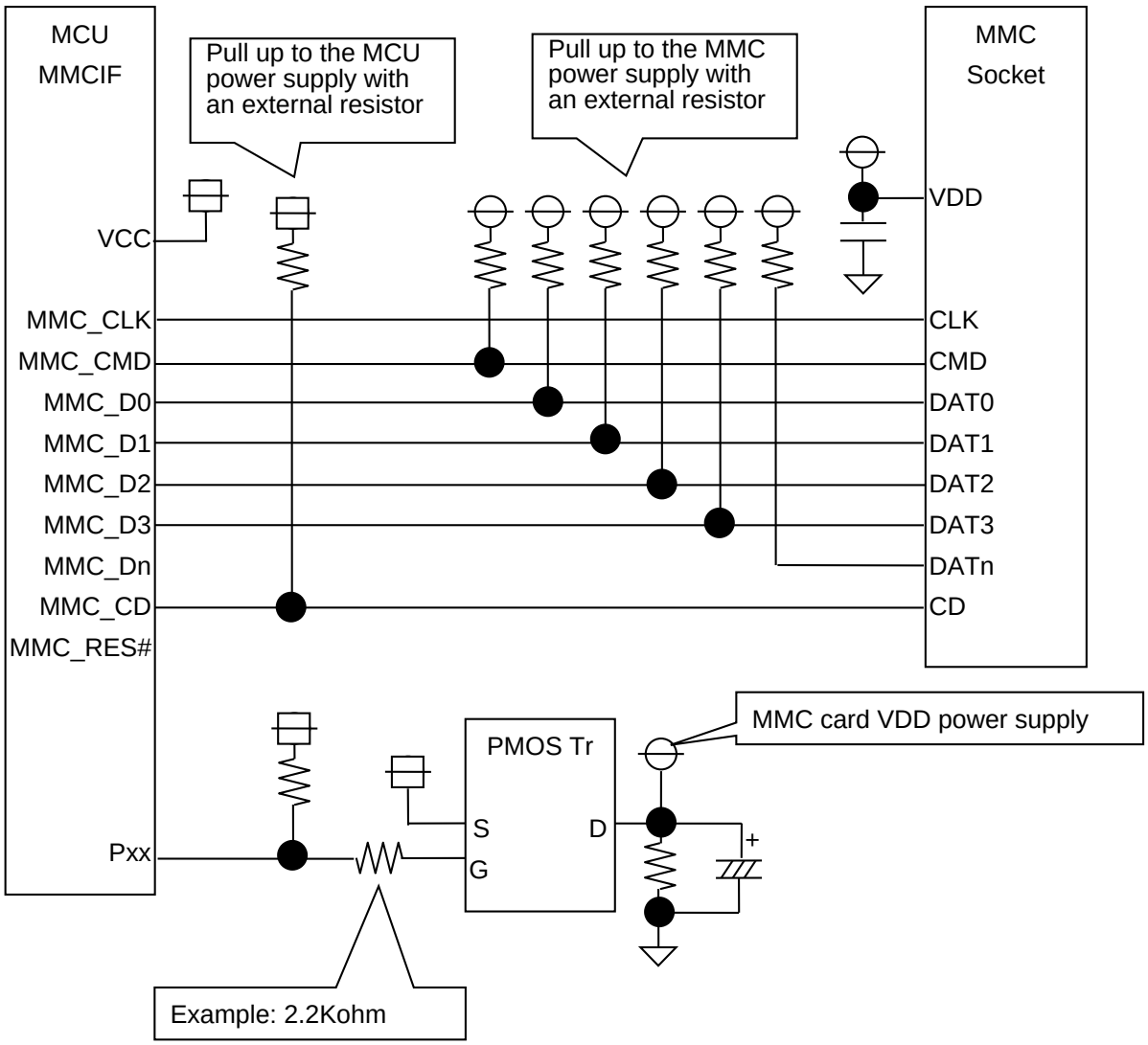
4.6.2.1 Connection Using an 8-bit Bus



The n in DATn indicates lines 4 to 7. Connect a pull-up resistor each line in DAT4 to DAT7.

Figure 4.3 Connection Between Microcontroller and 8-bit Bus MMC (Removable Media: MMC Card)

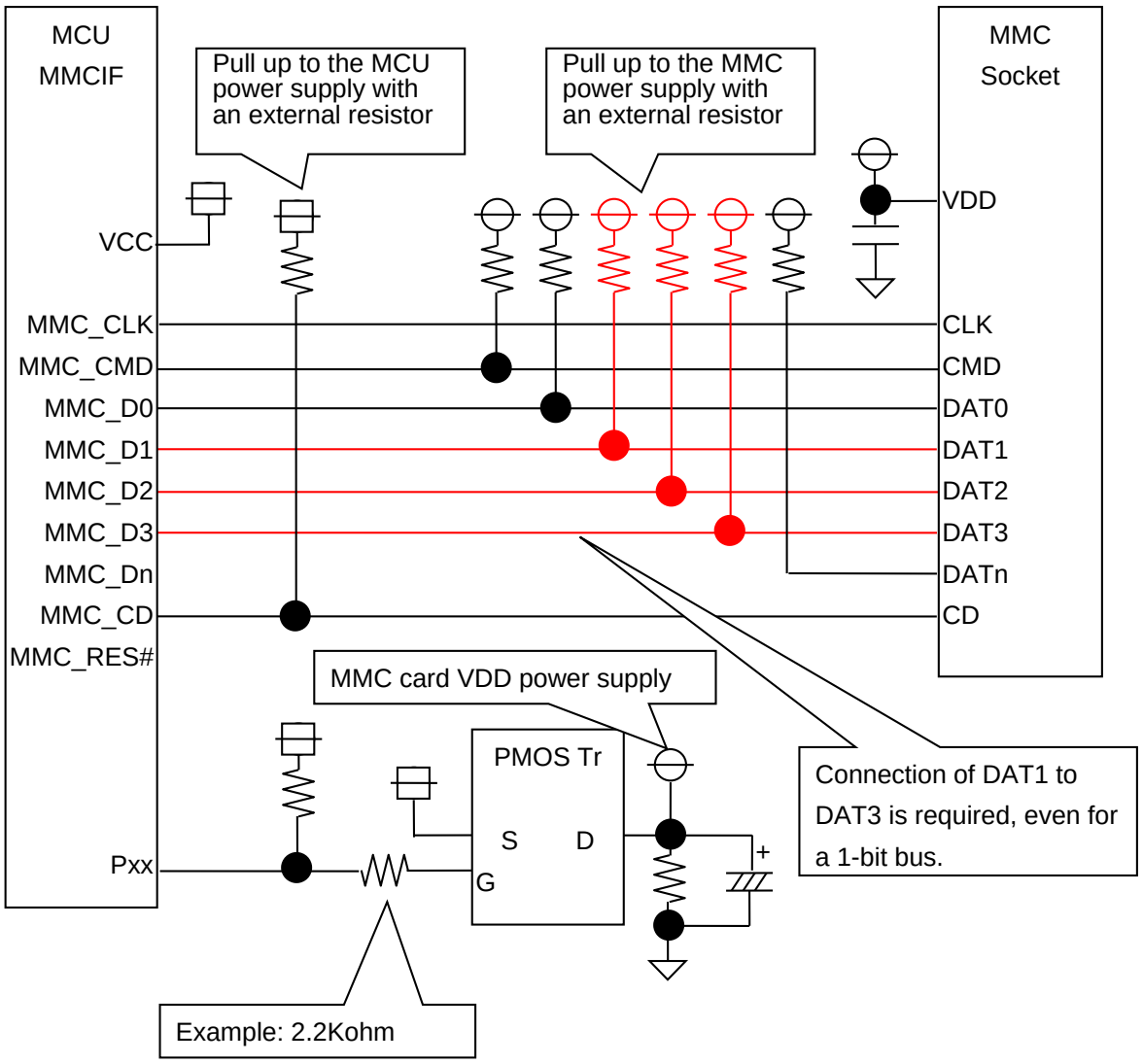
4.6.2.2 Connection Using an 4-bit Bus



The n in DATn indicates lines 4 to 7. Connect a pull-up resistor each line in DAT4 to DAT7.

Figure 4.4 Connection Between Microcontroller and 4-bit Bus MMC (Removable Media: MMC Card)

4.6.2.3 Connection Using an 1-bit Bus



The n in DATn indicates lines 4 to 7. Connect a pull-up resistor each line in DAT4 to DAT7.

Figure 4.5 Connection Between Microcontroller and 1-bit Bus MMC (Removable Media: MMC Card)

4.6.2.4 Microcontroller Resources

This MMCIF driver uses the following microcontroller resources.

Table 4.5 Pins and Functions Used

Used Resource	I/O	Description
MMC_CLK*1	Output	MMC clock output (required)
MMC_CMD*1	I/O	MMC command output/response input (required)
MMC_D0*1	I/O	MMC data 0 (required)
MMC_D1*1*3	I/O	MMC data 1 (required)
MMC_D2*1*3	I/O	MMC data 2 (required)
MMC_D3*1*3	I/O	MMC data 3 (required) Even when a 1-bit bus is used, this pin must be controlled for transition to SPI mode stop control.
MMC_D4*1*3	I/O	MMC data 4 (optional)
MMC_D5*1*3	I/O	MMC data 5 (optional)
MMC_D6*1*3	I/O	MMC data 6 (optional)
MMC_D7*1*3	I/O	MMC data 7 (optional)
MMC_CD*1*4	Input	MMC card detection input (optional)
MMC_RES#*5	Output	MMC reset (not controlled)
Microcontroller power supply	—	Microcontroller power supply, MMC_CD pin pull-up power supply (required)
MMC power supply	—	MMC power supply, MMC_CMD/MMC_Dn pin pull-up power supply (required)
Pxx (MMC power supply control ports) (Allocated to general-purpose I/O pins)*1*2	Output	MMC power supply voltage control output (optional) Allocate the number of pins required for power supply control. Use either active-high or active-low control according to the circuit configuration used.

Notes: 1. The user should allocate these pins.

2. The user should allocate and control these pins.

3. The following settings are possible, depending on the bus size used.

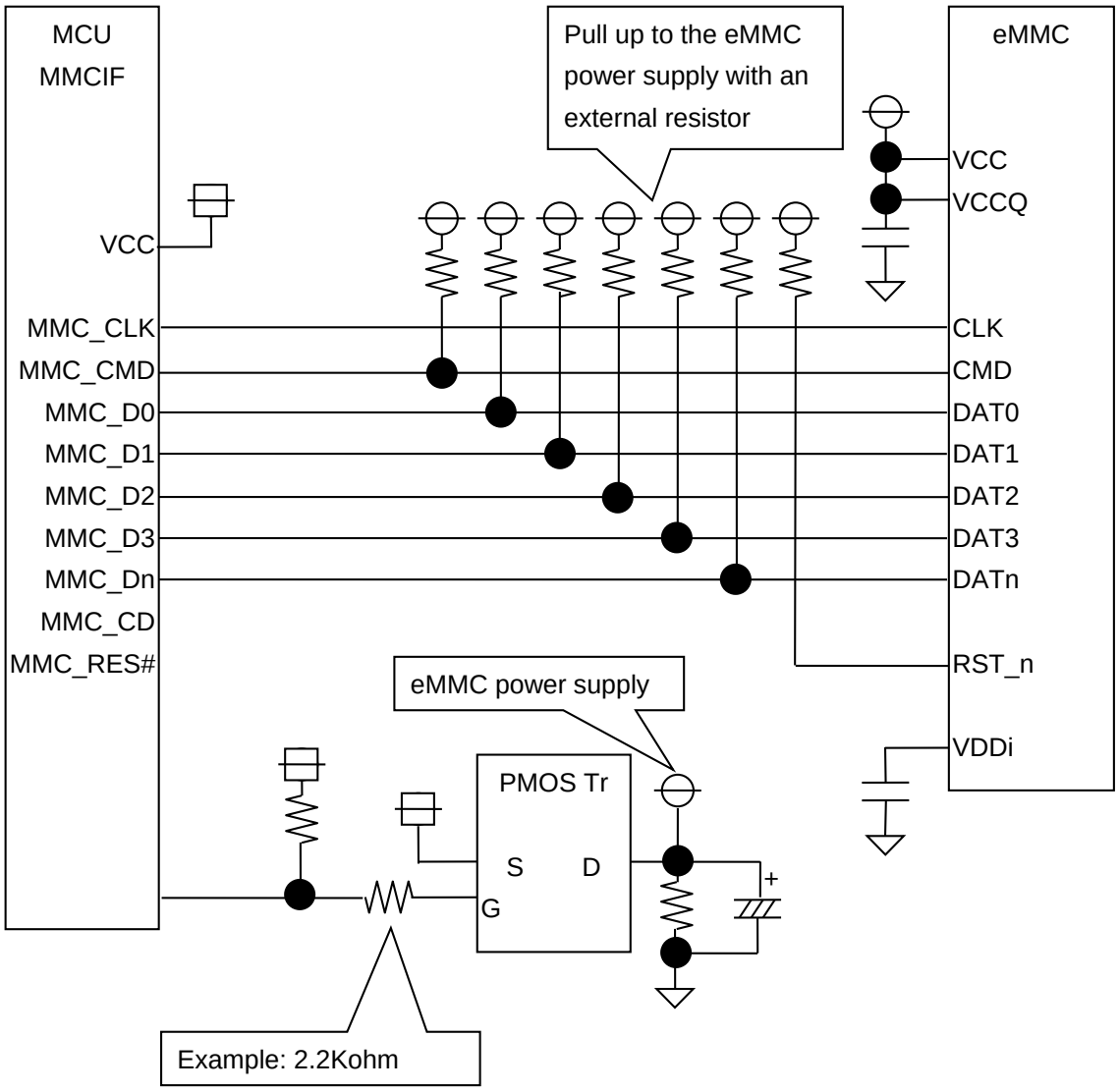
Maximum Bus Width Used by MMC Bus	Required MMC_Dn Pin Allocation
8	MMC_Dn (n = 0 to 7)
4	MMC_Dn (n = 0 to 3) The MMC_D4 to MMC_D7 pins can be used for other purposes.
1	MMC_Dn (n = 0 to 3) The MMC_D4 to MMC_D7 pins can be used for other purposes.

4. Set the MMC card detection function MMC_CFG_CHx_CD_ACTIVE to 1 (enabled) in the MMCIF driver file r_mmcif_rc_config.h.

5. There is no reset pin on an MMC card. This pin can be used for peripheral functions other than the MMC.

4.6.3 MMC (Embedded Multimedia Card: eMMC)

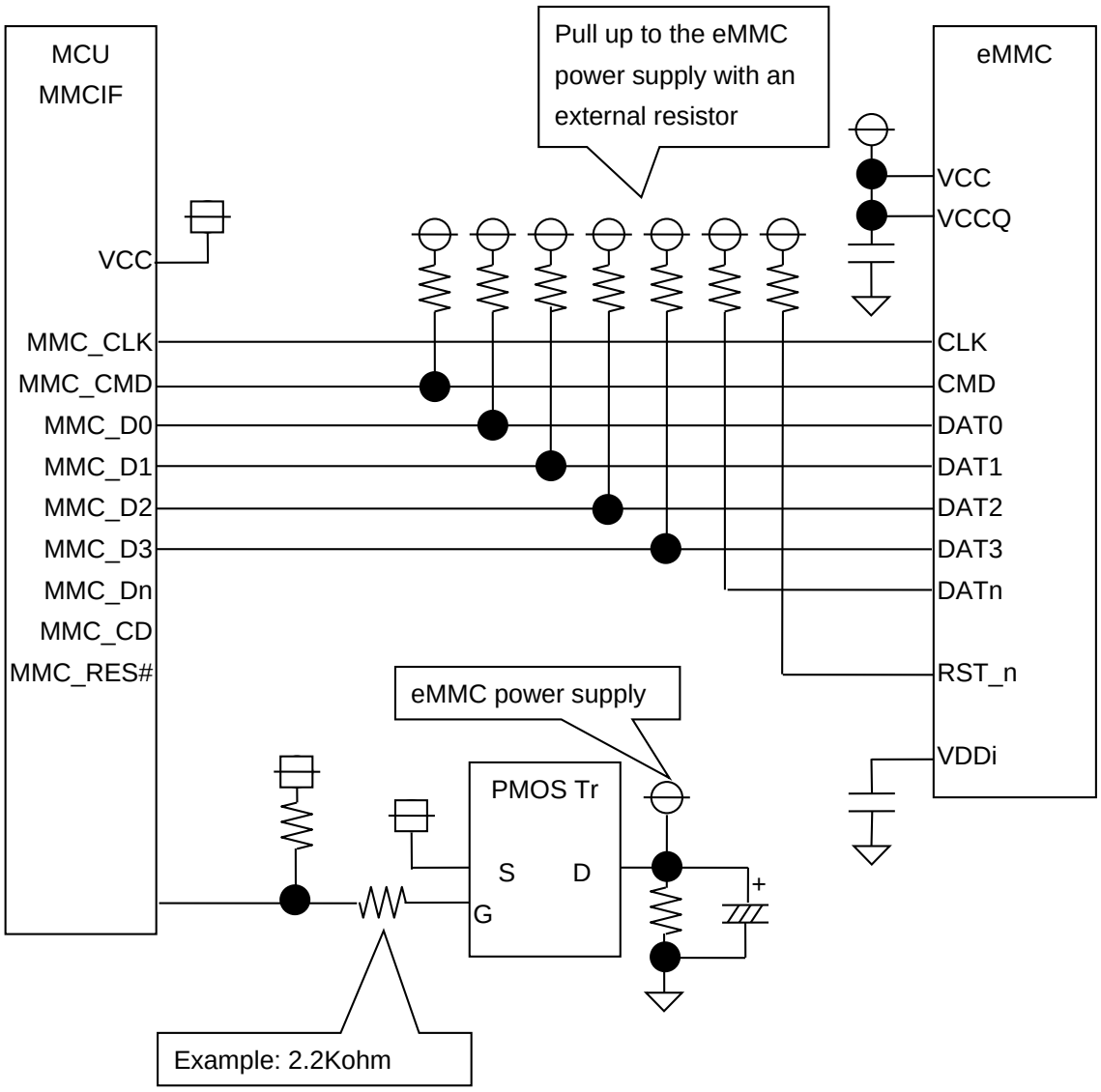
4.6.3.1 Connection Using an 8-bit Bus



The n in DATn indicates lines 4 to 7. Connect a pull-up resistor each line in DAT4 to DAT7.
The figures shows the case where the same 2.7 to 3.6V power supply is supplied to both the VCC and VCCQ eMMC pins

Figure 4.6 Connection Between Microcontroller and 8-bit Bus MMC (Embedded Multimedia Card: eMMC)

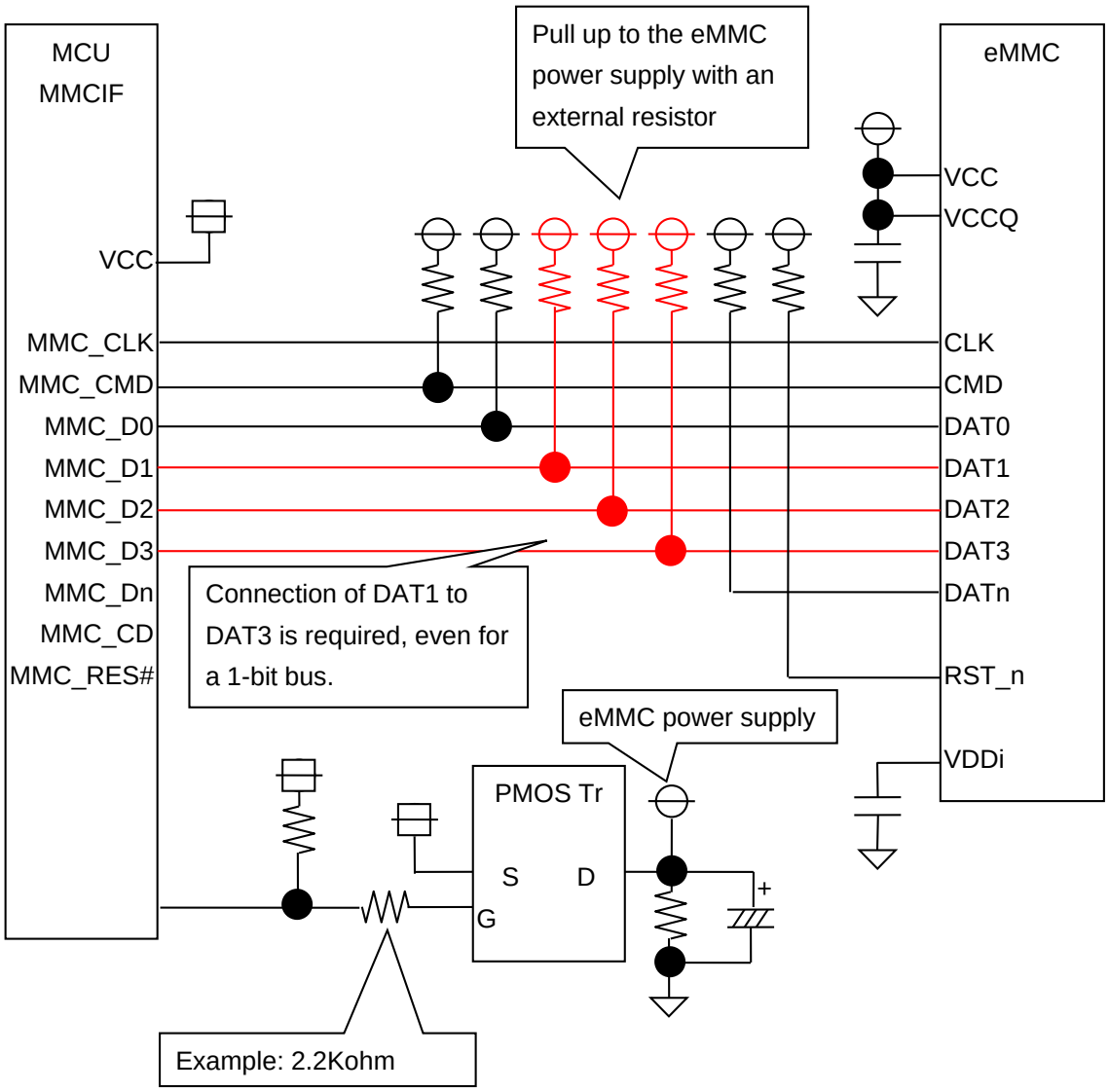
4.6.3.2 Connection Using an 4-bit Bus



The n in DATn indicates lines 4 to 7. Connect a pull-up resistor each line in DAT4 to DAT7.
The figures shows the case where the same 2.7 to 3.6V power supply is supplied to both the VCC and VCCQ eMMC pins.

Figure 4.7 Connection Between Microcontroller and 4-bit Bus MMC (Embedded Multimedia Card: eMMC)

4.6.3.3 Connection Using an 1-bit Bus



The n in DATn indicates lines 4 to 7. Connect a pull-up resistor each line in DAT4 to DAT7. The figures shows the case where the same 2.7 to 3.6V power supply is supplied to both the VCC and VCCQ eMMC pins.

Figure 4.8 Connection Between Microcontroller and 1-bit Bus MMC (Embedded Multimedia Card: eMMC)

4.6.3.4 Microcontroller Resources

This MMCIF driver uses the following microcontroller resources.

Table 4.6 Pins and Functions Used

Used Resource	I/O	Description
MMC_CLK*1	Output	MMC clock output (required)
MMC_CMD*1	I/O	MMC command output/response input (required)
MMC_D0*1	I/O	MMC data 0 (required)
MMC_D1*1*3	I/O	MMC data 1 (required)
MMC_D2*1*3	I/O	MMC data 2 (required)
MMC_D3*1*3	I/O	MMC data 3 (required) Even when a 1-bit bus is used, this pin must be controlled for transition to SPI mode stop control.
MMC_D4*1*3	I/O	MMC data 4 (optional)
MMC_D5*1*3	I/O	MMC data 5 (optional)
MMC_D6*1*3	I/O	MMC data 6 (optional)
MMC_D7*1*3	I/O	MMC data 7 (optional)
MMC_CD*1*4	Input	MMC card detection input (not controlled)
MMC_RES#*5	Output	MMC reset (not controlled)
Microcontroller power supply	—	Microcontroller power supply (required)
eMMC power supply*6	—	eMMC power supply, MMC_CMD/MMC_Dn pin pull-up power supply (required)
Pxx (MMC power supply control ports) (Allocated to general-purpose I/O pins)*1*2	Output	MMC power supply voltage control output (optional) Allocate the number of pins required for power supply control. Use either active-high or active-low control according to the circuit configuration used.

Notes: 1. The user should allocate these pins.
2. The user should allocate and control these pins.
3. The following settings are possible, depending on the bus size used.

Maximum Bus Width Used by MMC Bus	Required MMC_Dn Pin Allocation
8	MMC_Dn (n = 0 to 7)
4	MMC_Dn (n = 0 to 3) The MMC_D4 to MMC_D7 pins can be used for other purposes.
1	MMC_Dn (n = 0 to 3) The MMC_D4 to MMC_D7 pins can be used for other purposes.

4. Set the MMC card detection function MMC_CFG_CHx_CD_ACTIVE to 0 (disabled) in the MMCIF driver file r_mmcif_rx_config.h. Since this disables control of the MMC card detection pin, it can be used for peripheral functions other than the MMC.
5. This MMCIF driver does not control a reset pin. This this pin can be used for peripheral functions other than the MMC. Note, however, that the Bit[1:0] field (RST_n_FUNCTION[162]) in the Extended CSD must be used with the value 0x0 (the default).
6. If the microcontroller power supply and eMMC power supply control voltages differ, set the eMMC power supply voltage here.

5. Demo Projects

5.1 Overview

The sample program is included and can be found in the FITDemos directory. This sample program performs the processing described in section 4.4, MMC card Insertion and Power-On Timing, 4.5, MMC card Removal and Power-Off Timing and 1.6.5,Chattering Control at MMC Card Insertion, as well as MMC read and write processing.

Note that in this sample program, Pre-defined (MMC_PRE_DEF) is taken to be the default setting for read/write mode. With Pre-defined, there is a possibility that MMC card will not work. When using an MMC card, change this setting from Pre-defined to Open-ended (MMC_OPEN_END). The locations where the changes should be made are on lines 66 to 69 of the sample program.

5.2 State Transition Diagram

Figure 5.1 shows the state transition diagram for this driver.

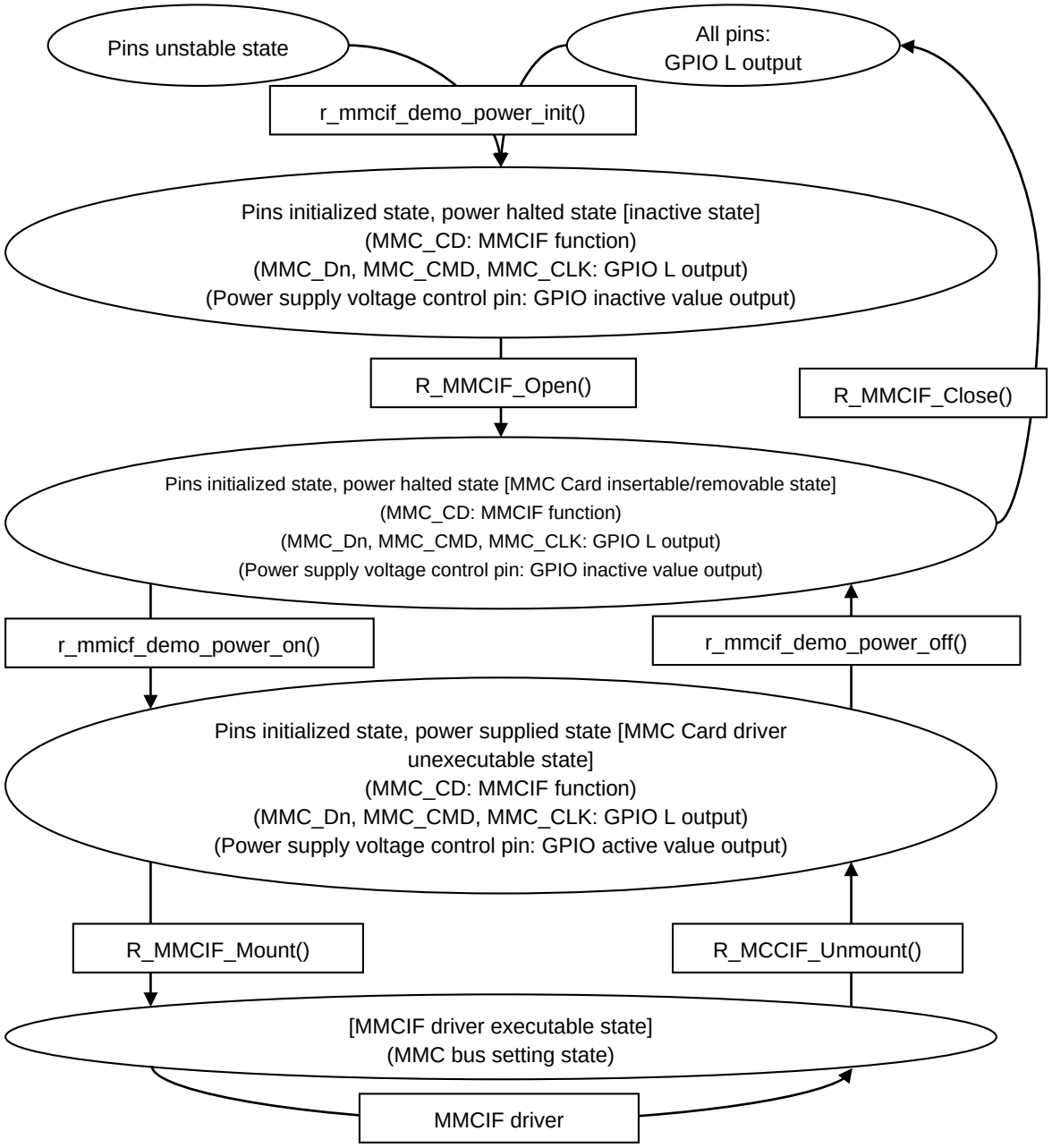


Figure 5.1 State Transition Diagram

5.3 Configuration Overview

The sample program configuration options are set in the file r_mmcif_rx_demo_pin_config.h.

The table below lists the option names and set values when the RX64M RSK is used.

Configuration options in *r_mmcif_rx_demo_pin_config.h*	
#define MMC_CFG_MODE_SW (1) #define MMC_CFG_MODE_DMAC (0) #define MMC_CFG_MODE_DTC (0) Note: Software transfer is selected as the default value.	These set the transfer mode used in the sample program. Only 1 transfer mode should be enabled. (1: Enabled, 0: Disabled) When DMAC or DTC transfers are used, the corresponding DMAC or DTC FIT module must be acquired separately.
#define MMC_CFG_POWER_PORT_NONE Note: The default value is "disabled".	This definition is used when an MMC card is used. If MMC card power supply control is not required, enable this definition. If MMC card power supply control is required, disable this definition.
#define MMC_CFG_POWER_HIGH_ACTIVE (1) Note: The default value is "1 (high level supplied)".	This definition is used when an MMC card is used and furthermore card power supply control is required. When set to 1, a high level is supplied to the port that controls the card power supply circuit to enable the card power supply circuit. When set to 0, a low level is supplied to the port that controls the card power supply circuit to enable the card power supply circuit.
#define MMC_CFG_CHAT_CNT (300) Note: The default value is "300 (300 ms wait)".	This definition is used when an MMC card is used. This is the chattering counter used for card insertion or removal. A wait of 1 ms is provided for each count of the counter. Set this definition to a value appropriate for the system used.
#define MMC_CFG_POWER_ON_WAIT (100) Note: The default value is "100 (100 ms wait)".	This definition is used when an MMC card is used. Set this definition to the wait time after power supply is started to the MMC card power supply circuit until the operating voltage is reached. A wait of 1 ms is provided for each count of the counter. Set this definition to a value appropriate for the system used.
#define MMC_CFG_POWER_OFF_WAIT (100) Note: The default value is "100 (100 ms wait)".	This definition is used when an MMC card is used. Set this definition to the wait time after power supply to the MMC card power supply circuit is stopped until the voltage at which MMC card removal is possible is reached. A wait of 1 ms is provided for each count of the counter. Set this definition to a value appropriate for the system used.
#define MMC_CFG_POWER_CHx_PORT Note: The x in CHx indicates a channel number (x = 0 or 1)	Set these definitions to the port number for each pin allocated for channel x. Surround each setting value with single quotation marks ‘ ‘.
#define MMC_CFG_POWER_CHx_BIT Note: The x in CHx indicates a channel number (x = 0 or 1)	Set these definitions to the bit number for each pin allocated for channel x. Surround each setting value with single quotation marks ’ ‘.

5.4 API Functions

The power supply voltage control pin control functions in the sample program are shown below.

Add or modify functions as necessary.

Table 5.1 Pin Control API Functions

Function	Function Outline
r_mmcif_demo_power_init()	Initializes the power supply voltage control pin settings
r_mmcif_demo_power_on()	Starts supply of power supply voltage
r_mmcif_demo_power_off()	Stops supply of power supply voltage
r_mmcif_demo_softwaredelay()	Performs delay

(1) r_mmcif_demo_power_init()

This function initializes the settings of the MMCIF pins that are used by the MMCIF driver. It also initializes the power supply voltage control pin settings of the eMMC or MMC card.

Format

```
mmc_status_t r_mmcif_demo_power_init(
uint32_t channel
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

Return Values

MMC_SUCCESS	*Successful operation*

Description

Initializes the settings of the MMC_CD, MMC_Dn, MMC_CMD, and MMC_CLK pins, which are used by the MMCIF driver specified in r_mmcif_rx_demo_pin_config.h. Also initializes the power supply voltage control pin settings of the eMMC or MMC card.

Special Notes:

The power supply voltage control pins are set as follows.

- The port mode register (PMR) is set to the general-purpose I/O port.
- Set the pull-up control register (PCR) to input pull-up resistance disabled.
- Pin output is set to the inactive state.

(2) r_mmcif_demo_power_on()

This function controls the power supply voltage control pins of the eMMC or MMC card, and starts the supply of power from the power supply.

Format

```
mmc_status_t r_mmcif_demo_power_on(
  uint32_t channel
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Description

Controls the power supply voltage control pins of the eMMC or MMC card, and starts the supply of power from the power supply. Then, after the time specified by MMC_CFG_POWER_ON_WAIT in r_mmcif_rx_demo_pin_config.h has elapsed, returns the result.

Special Notes:

Modify as necessary.

After starting the supply of power supply voltage, executes the mmcif_pin_softwaredelay() function to wait until the operating voltage is reached. Set the wait time using MMC_CFG_POWER_ON_WAIT in section 5.3,Configuration Overview.

Initialization using the r_mmcif_demo_power_init() function must be performed before executing this function.

(3) r_mmcif_demo_power_off()

This function controls the power supply voltage control pins of the eMMC or MMC card, and stops the supply of power from the power supply.

Format

```
mmc_status_t r_mmcif_demo_power_off(
  uint32_t channel
)
```

Parameters

channel

Channel number	The number of the MMCIF channel used (numbering starts at 0)

Return Values

MMC_SUCCESS	*Successful operation*
MMC_ERR	*General error*

Description

Controls the power supply voltage control pins of the eMMC or MMC card, and stops the supply of power from the power supply. Then, after the time specified by MMC_CFG_POWER_OFF_WAIT in r_mmcif_rx_demo_pin_config.h has elapsed, returns the result.

Special Notes:

After stopping the supply of power supply voltage, executes the mmcif_pin_softwaredelay() function to wait until the removable voltage is reached. Set the wait time using MMC_CFG_POWER_OFF_WAIT in section5.3,Configuration Overview.

Initialization using the r_mmcif_demo_power_init() function must be performed before executing this function.

(4) r_mmcif_demo_softwaredelay()

This function is used when waiting for a particular time.

Format

```
bool r_mmcif_demo_softwaredelay(
  uint32_t delay,
  mmc_delay_units_t units
)
```

Parameters

delay

Timeout time (Units: set with the units)

units

Microseconds: MMC_DELAY_MICROSECS
Milliseconds: MMC_DELAY_MILLISECS
Seconds: MMC_DELAY_SECS

Return Values

true	*Successful operation*
false	*Parameter error*

Description

This function performs wait time processing.

True is returned when the timeout time specified in the argument delay has elapsed.

Special Notes

The wait time processing is listed in Table 5.2. Since this function only waits for the set time, it can be replaced with the operating system activating task delay processing (example: the µITRON dly_tsk() function).

Table 5.2 Wait Time Processing

Type	Description
MMC card power on power supply voltage stabilization time	The wait time until the operating voltage is reached after power supply is started to the MMC card power supply circuit <100 ms> Note: The wait time can be modified with MMC_CFG_POWER_ON_WAIT.
MMC card power off voltage turn-off time	The wait time until the MMC card removable voltage is reached after supply is stopped to the MMC card power supply circuit <100 ms> Note: The wait time can be modified with MMC_CFG_POWER_OFF_WAIT.

Note: The times in angle brackets ("<>") above are the values set when this MMCIF driver is provided.

5.5 Replacing Wait Time Processing with Operating System Processing

The r_mmcif_demo_softwaredelay() function, which processes the delays that arise in the sample program, can be replaced by the task delay processing of the OS itself (for example, dly_tsk() in µITRON).

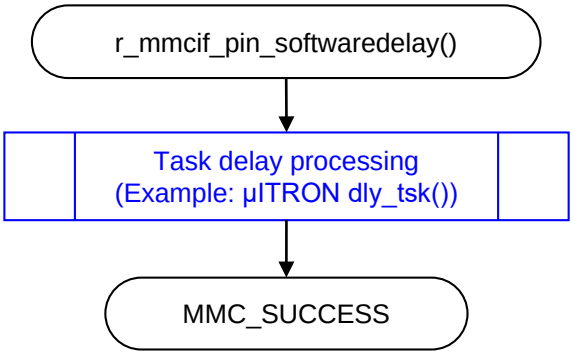


Figure 5.2 Wait Example Using Operating System Task Delay Processing

5.6 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Brower* >> *Application Notes* tab.

6. Appendices

6.1 Operation Confirmation Environment

This section describes operation confirmation environment for this driver.

Table 6.1 Operation Confirmation Environment (Ver. 1.03)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V6.2.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.08.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Version of the module	Ver.1.03
Board used	Renesas Starter Kits for RX64M (product No.:R0K50564MSxxxBE) Renesas Starter Kits for RX71M (product No.:R0K50571MSxxxBE) Renesas Starter Kits for RX65N (product No.:RTK500565NSxxxxxBE) Renesas Starter Kits for RX65N-2MB (product No.:RTK50565N2SxxxxxBE)*

Table 6.2 Operation Confirmation Environment (Ver. 1.04)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.3.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Version of the module	Ver.1.04

Table 6.3 Operation Confirmation Environment (Ver. 1.05)

Item	Contents
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	Big endian/little endian
Version of the module	Ver.1.05
Board used	Renesas Starter Kit+ for RX64M (product No.:R0K50564Mxxxxxx)

Table 6.4 Operation Confirmation Environment (Ver. 1.06)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 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	Big endian/little endian
Version of the module	Ver.1.06
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 6.5 Operation Confirmation Environment (Ver. 1.07)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 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	Big endian/little endian
Version of the module	Ver.1.07
Board used	Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxxx)

6.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file “platform.h”.

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”

- Using e^2 studio:

  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

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

(2) Q: I have added the FIT module to the project and built it. Then I got the error: This MCU is not supported by the current r_mmcif_rx module.

A: The FIT module you added may not support the target device chosen in your project. Check the supported devices of added FIT modules.

6.3 Replacing Wait Processing with Operating System Processing

The handling of status interrupts generated by this MMCIF driver can be replaced with operating system processing. The table below lists the details of the related functions.

Table 6.6 Target Microcontroller Interface Functions

Function	Functional Overview
r_mmcif_dev_int_wait()	Status interrupt wait processing
r_mmcif_dev_wait()	Wait time processing

(1) r_mmcif_dev_int_wait()*

This function is used when waiting for a status interrupt.

Format

```
mmc_status_t r_mmcif_dev_int_wait(
  uint32_t channel,
  int32_t time
)
```

Parameters

channel

Channel number — The number of the MMCIF channel used (numbering starts at 0)

time

Timeout time (units ms)

Return Values

MMC_SUCCESS	*Successful operation (Interrupt request generation)*
MMC_ERR	*General error*

Description

This function performs the interrupt wait processing used for protocol communication with the MMC.

When an interrupt request is verified, this function returns MMC_SUCCESS.

If no interrupt is detected within the interrupt wait time given by the argument time, it returns MMC_ERR.

The interrupt wait processing is already included as processing that uses interrupts.

This function calls interrupt status flag register acquisition processing (the r_mmcif_get_intstatus() function) internally to determine whether or not an interrupt has occurred.

Special Notes

The response reception wait time during communication with the MMC and the data transfer completion wait time can be allocated to other processing.

Figure 6.1 shows a usage example in which the operating system invoking task delay processing (in this example, the µITRON dly_tsk() function) is used. Note, however, that users must code the required calls to the r_mmcif_dev_int_wait() function themselves.

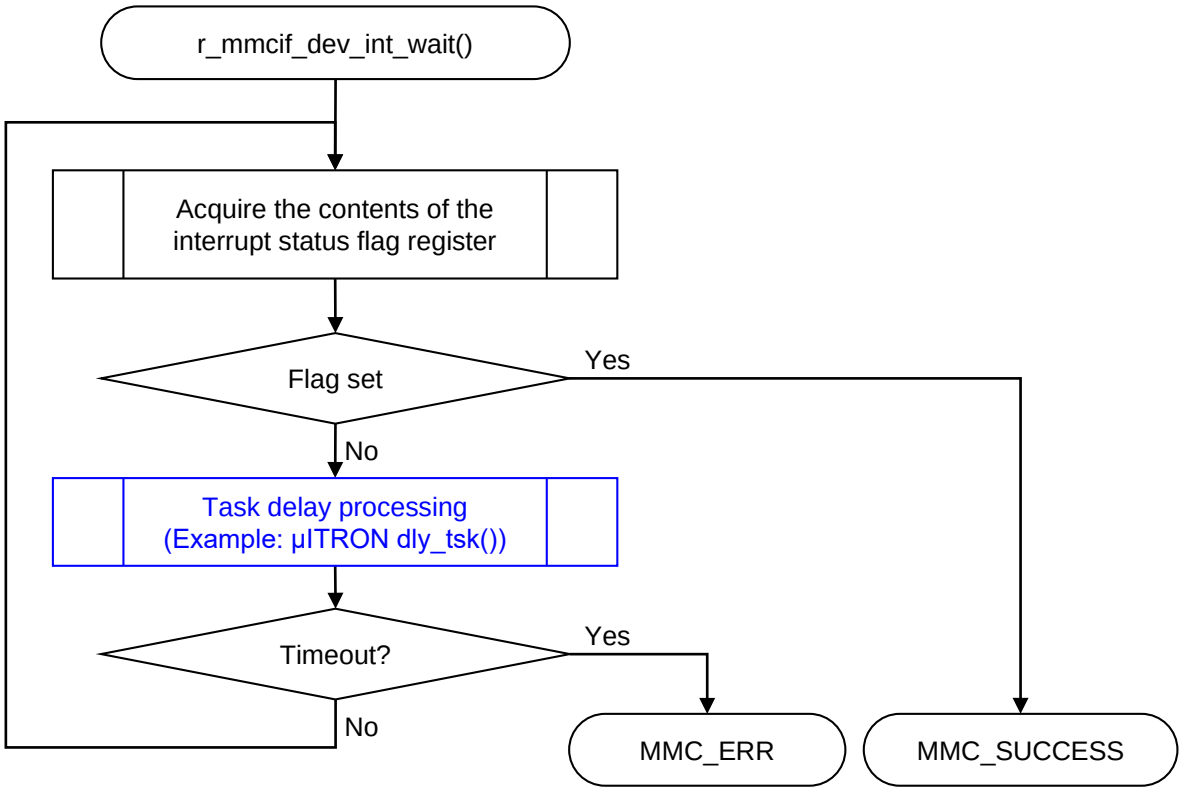


Figure 6.1 MMC Protocol Status Verification Example Using Operating System Task Delay Processing

Figure 6.2 shows a usage example in which the operating system event flag set wait processing is used. If this functionality is used, the user must replace the r_mmcif_dev_int_wait() function interrupt status flag register acquisition processing (the r_mmcif_get_intstatus() function) with event flag set wait processing and furthermore add wakeup processing to the MMC protocol status interrupt callback function.

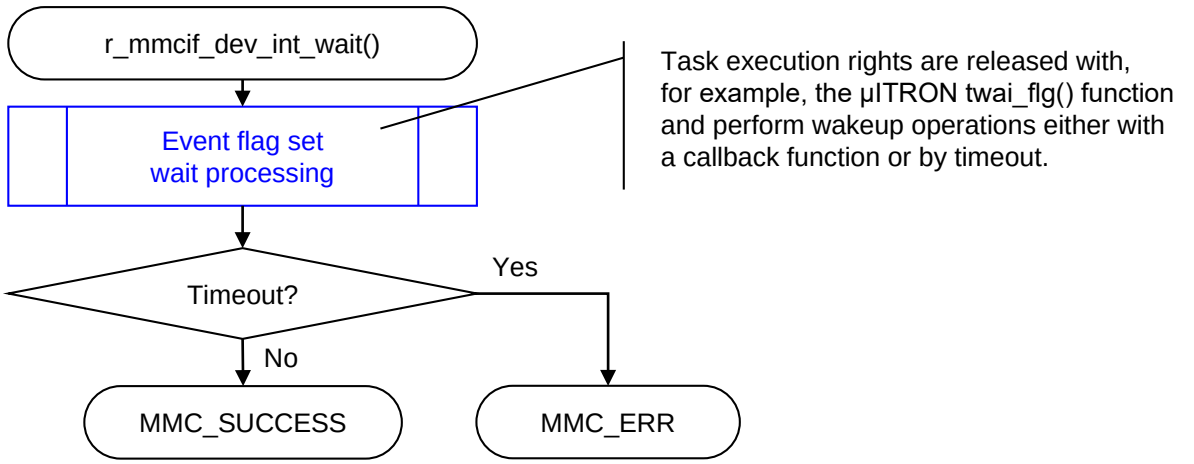


Figure 6.2 MMC Protocol Status Verification Example Using Operating System Wait Task Processing

(2) r_mmcif_dev_wait()

This function is used when waiting for a particular time.

Format

```
mmc_status_t r_mmcif_dev_wait(
  uint32_t channel,
  int32_t time
)
```

Parameters

channel

Channel number — The number of the MMCIF channel used (numbering starts at 0)

time

Timeout time (units ms)

Return Values

MMC_SUCCESS	*Successful operation (Interrupt request generation)*
MMC_ERR	*General error*

Description

This function performs wait time processing.

MMC_SUCCESS is returned when the timeout time specified in the argument time has elapsed.

Special Notes

Table 6.7 lists the wait time processing not associated with status verification. Since this function only provides a wait for a specified time, it can be replaced with the operating system task delay processing (example: the µITRON dly_tsk() function).

Table 6.7 Wait Time Processing not Associated with Status Verification

Type	Description*
The 74 clock cycle wait during MMC initialization	Card identification mode: A wait of 74 clock cycles <3 ms> for MMC initialization (A minimum of 2 ms and a maximum of 3 ms must be assured)
Ready state transition detection during MMC initialization	Card identification mode: A wait <5 ms> for transition to the MMC ready state (maximum: 1 second) For MMC, a CMD1 command is issued at 5 ms intervals, repeated a maximum of 200 times.
Ready state transition detection on an error during mount, read, or write processing	Data transfer mode: A wait <1 ms> for transition to the MMC ready state after a CMD13 command has been issued for mount, read, or write processing (A 1 ms wait is repeated multiple times until a timeout occurs.)

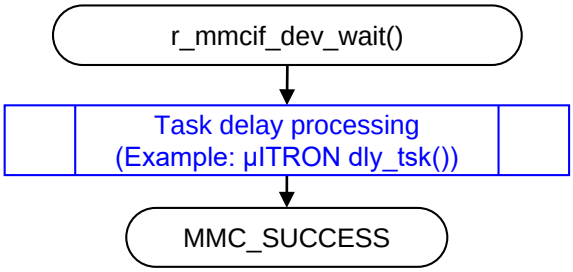


Figure 6.3 Wait Example Using Operating System Task Delay Processing

7. Reference Documents

User’s Manual: Hardware
The latest versions 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

Not applicable technical update for this module.

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
1.03	Mar 31, 2018	-	First edition issued. MMC Mode MMCIF Driver Software RTM0RX0000DMMC Ver.1.02 User's Manual (R01UW0118) changed to the application note.
1.04	Feb 01, 2019	87	Added Table 6.2 Operation Confirmation Environment (Ver. 1.04)
		-	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
1.05	May 20, 2019	-	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		-	Deleted Table 1.3.
		-	Deleted Table 6.3, 6.4 Operation Confirmation Environment.
		1	Added Target Compilers.
		1	Deleted R01AN1723, R01AN1826 and R20AN0451 from Related Documents.
		21	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		24	2.8 Code Size, amended.
		88	Added Table 6.3 Operation Confirmation Environment (Ver.1.05).
1.06	Jul 30, 2019	-	Changes associated with RX72M.
		24	Changed Section 2.8 Code Size.
		30	Added Section 2.13 "for", "while" and "do while" statements.
		31-67	Delete "Reentrant" item on the API description page.
		90	Added Table 6.4 Operation Confirmation Environment (Ver.1.06).
1.07	Nov 22, 2019	-	Changes associated with RX66N and RX72N.
		9	Added RX72N hardware setting in 1.4.1 Quick Start Guide.
		21	Added RX66N and RX72N device in 2.4 Interrupt Vector.
		24	2.8 Code Size, amended.
		65-66	Changed "Special Notes" in 3.22 R_MMCIF_Set_LogHdlAddress() and 3.23 R_MMCIF_Log().
		90	Added Table 6.5 Operation Confirmation Environment (Ver.1.07).

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

Trademarks

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

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/.