RX Family

SD Mode SD Memory Card Driver Firmware Integration Technology

Introduction

This application note describes the SD Mode SD Memory Card driver which uses Firmware Integration Technology (FIT). This driver controls SD Memory Cards in SD mode using the SD Host Interface (SDHI) module included in Renesas Electronics RX Family microcontrollers. In this document, this driver is referred to as the SD Memory Card driver.

When developing host devices that are compliant with the SD Specifications, the user must enter into the SD Host/Ancillary Product License Agreement (SD HALA).

Please refer for details to SD Association homepage.

https://www.sdcard.org/

Target Device

- RX Family

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 (R01AN2063EJ)
- RX Family DTC Module Using Firmware Integration Technology (R01AN1819EJ)
- RX Family CMT Module Using Firmware Integration Technology (R01AN1856EJ)
- RX Family LONGQ Module Using Firmware Integration Technology (R01AN1889EJ)
- RX Family SDHI Module Using Firmware Integration Technology (R01AN3852EJ)

Contents

1. Overview

1.1 SD Memory Card driver

By using this product in conjunction with the lower-layer SDHI FIT module, which is available free of charge, it is possible to control an SD Memory Card. By using this product in conjunction with a separately supplied FAT file system, files can be accessed on an SD Memory Card.

Note that the SDHI control software that is common to SD memory and SDIO is referred to as the SD Card driver.

The SD Memory Card 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 SD Memory Card driver

Table 1.1 and Table 1.2 list this driver function.

Table 1.1 SDHI Functions

Item	Function
Conforming standard	Conforms to the SD Specifications Part 1 Physical Layer Simplified Specification Version 6.00.
SDHI control driver	Block type device driver with 512-byte/sector
Target SD Cards	SD Memory Card
SD Card operating voltage	Only 2.7-3.6 V (high voltage) operation, with 3.3 V signal levels, is supported.
SD Card bus interface	SD mode (4-bit) is supported.
Number of SD Cards that can be controlled	One device/channel
SD Card speed mode	The Default Speed mode are supported. This SD Card driver discriminates the speed mode and initializes the card
SD Card memory Capacity	Standard Capacity SD Memory Cards (SDSC) and High Capacity SD Memory Cards (SDHC, SDXC) are supported.
SD Card memory control objects	Only a user area is supported. Protected area control is not supported.
SD Card detection function	Only detection using the CD pin is supported.

Table 1.2 Microcontroller Functions

Item	Function
Target microcontroller	RX Family microcontrollers that include the SDHI
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 SD Memory Card driver.

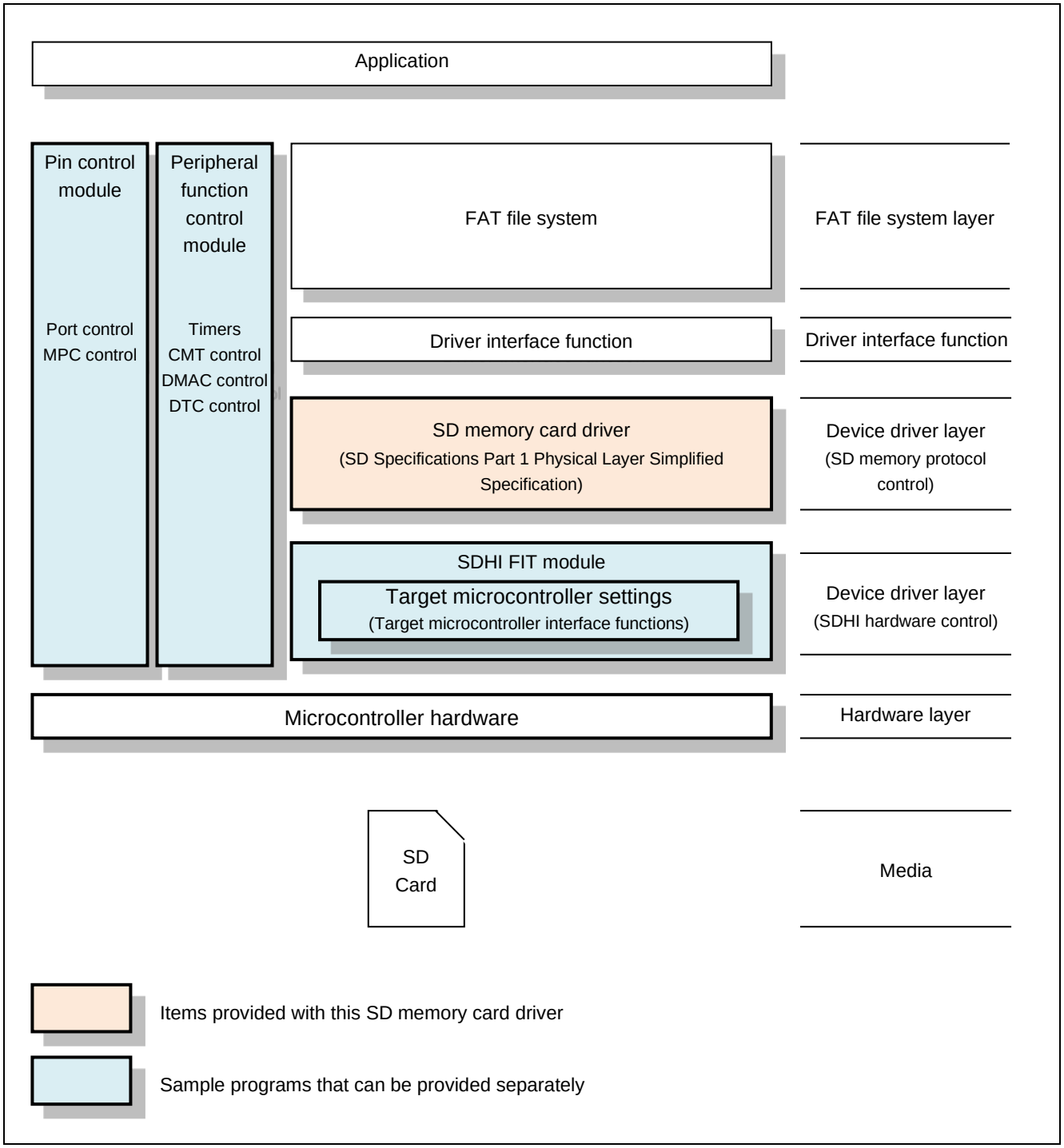


Figure 1.1 Application Structure

(1) FAT File System

This is the software used for SD memory file management. A FAT file system must be provided separately. Please obtain it from the following as necessary.

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

(2) Driver Interface Functions

This is the software that implements the layer that connects the Renesas Electronics FAT file system API with the SD Memory Card 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) SD Memory Card Driver

This software implements the SD Specifications Part 1 Physical Layer Simplified Specification SD memory protocol control.

(4) SDHI FIT Module

This software controls the SDHI hardware. It also includes microcontroller-dependent target microcontroller interface functions and interrupt setting files.

(5) 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.

(6) Pin Control Module (Sample Program)

This is the pin control software used for SDHI control. The microcontroller resources used consist of the port control (SDHI function control and SD Card power supply port control) and MPC control (SDHI 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 RX Family MCU 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 SD Memory Card driver uses the SD Specifications Part 1 Physical Layer Simplified Specification protocol. The table below lists the library functions.

Table 1.4 shows the API Functions for this driver.

Table 1.3 API Functions

Function	Functional Overview
R_SDC_SD_Open()	Driver open processing
R_SDC_SD_Close()	Driver close processing
R_SDC_SD_GetCardDetection()	Insertion verification processing
R_SDC_SD_Initialize()	Initialization processing
R_SDC_SD_End()	End processing
R_SDC_SDMEM_Read()	Read processing *1
R_SDC_SDMEM_ReadSoftwareTrans()	Read processing (software transfers)
R_SDC_SDMEM_ReadSoftwareTransSingleCmd()	Read processing (CMD17 single software transfers)
R_SDC_SDMEM_Write()	Write processing *1
R_SDC_SDMEM_WriteSoftwareTrans()	Write processing (software transfers)
R_SDC_SD_Control()	Driver control processing SDC_SD_SET_STOP command
R_SDC_SD_GetModeStatus()	Mode status information processing
R_SDC_SD_GetCardStatus()	Card status information processing
R_SDC_SD_GetCardInfo()	Register information processing
R_SDC_SDMEM_GetSpeed()	Speed class information processing
R_SDC_SD_CdInt()	Insertion/removal interrupt setup (including registration of the insertion/removal interrupt callback functions)
R_SDC_SD_IntCallback()	Protocol status interrupt callback function registration processing
R_SDC_SD_GetErrCode()	Driver error code processing
R_SDC_SD_GetBuffRegAddress()	SD buffer register address processing
R_SDC_SD_1msInterval()	Interval timer count processing
R_SDC_SD_SetDmacDtcTransFlg()	DMAC/DTC transfer complete flag setting processing
R_SDC_SD_SetLogHdlAddress()	LONGQ module handler address setup processing *2
R_SDC_SD_Log()	Error log processing *2
R_SDC_SD_GetVersion()	Driver version information processing

Notes: 1. When DMAC transfers or DTC transfers are set as the data transfer for the operating mode during initialization 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 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 SD Card using a Renesas Starter Kit (RSK) is described below.

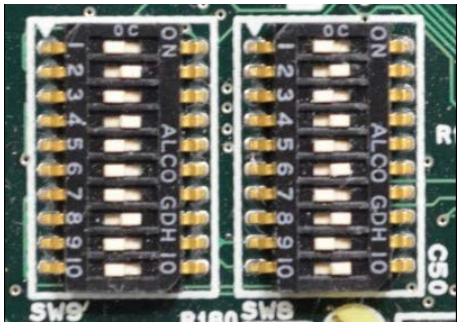
1.4.1.1 Hardware Settings

Settings must be made on the RSK for each target microcontroller. In the case of the RSK for RX231, the PMOD SD Card conversion board must be obtained separately.

(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



(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

No settings are necessary.

(4) RSK for RX231

Install the PMOD SD Card conversion board in PMOD2 on the RSK for RX231 board.

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_sdc_sdmem_rx_vX.XX.zip, r_sdc_sdmem_rx_vX.XX.xml, and r_sdc_sdmem_rx_vX.XX_extend.mdf to the folder containing the e^2 studio FIT modules (normally C:\Renesas\e2_studio\FITModules).
3. Refer to Renesas e^2 studio Smart Configurator User Guide (R01AN0451), and add r_bsp, r_sdc_sdmem_rx, r_sdhi_rx, r_cmt_rx, r_dmaca_rx, r_dtc_rx, and r_longq to your project.
4. Copy the sample program r_sdc_sdmem_rx_demo_main* to the src folder of your project.
5. Make settings to the configuration options of the sample program. For how to make these settings, see 5.3, Configuration Overview.

Note: * Contained in the FITDemos\rx65n_1mb_rsk.zip\src folder in the product package.

1.4.2 Basic Control

1.4.2.1 Supported Commands

This SD Memory Card driver uses the following commands.

The table below lists the SD Card commands, the User's Manual: Hardware and the status of support in this SD Memory Card driver.

Table 1.4 Commands Supported by this SD Memory Card Driver

(○: Supported, ×: Not supported)

Command	Microcontrollers Supported	This Driver	Remarks
CMD0	○	○	Used in SD memory initialization
CMD2	○	○	Used in SD memory initialization
CMD3	○	○	Used in SD memory initialization
CMD4	○	○	Used in SD memory initialization
CMD5	○	×	Unused
CMD6	○	×	Unused
CMD7	○	○	Used in SD memory initialization
CMD8	○	○	Used in SD memory initialization
CMD9	○	○	Used in SD memory initialization
CMD10	○	×	Unused
CMD11	○	×	Unused
CMD12	○	○	Used for SD memory read/write processing
CMD13	○	○	Used for SD memory read/write processing
CMD15	○	×	Unused
CMD16	○	○	Used in SD memory initialization
CMD17	○	○	Used for SD memory read/write processing
CMD18	○	○	Used for SD memory read/write processing
CMD20	○	×	Unused
CMD24	○	○	Used for SD memory read/write processing
CMD25	○	○	Used for SD memory read/write processing
CMD27	○	×	Unused
CMD28	○	×	Unused
CMD29	○	×	Unused
CMD30	○	×	Unused
CMD32	○	×	Unused
CMD33	○	×	Unused
CMD38	○	×	Unused
CMD42	○	×	Unused

Command	Microcontrollers Supported	This Driver	Remarks
CMD52	○	×	Unused
CMD53	○	×	Unused
CMD55	○	○	Used in SD memory initialization
CMD56	○	×	Unused
ACMD6	○	○	Used in SD memory initialization
ACMD13	○	○	Used in SD memory initialization
ACMD22	○	○	Used for SD memory write processing
ACMD23	○	○	Used for SD memory write processing
ACMD41	○	○	Used in SD memory initialization
ACMD42	○	○	Used in SD memory initialization
ACMD51	○	○	Used in SD memory initialization

1.4.2.2 Relation Between Data Buffers and Data in the SD Card

This SD Card 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.

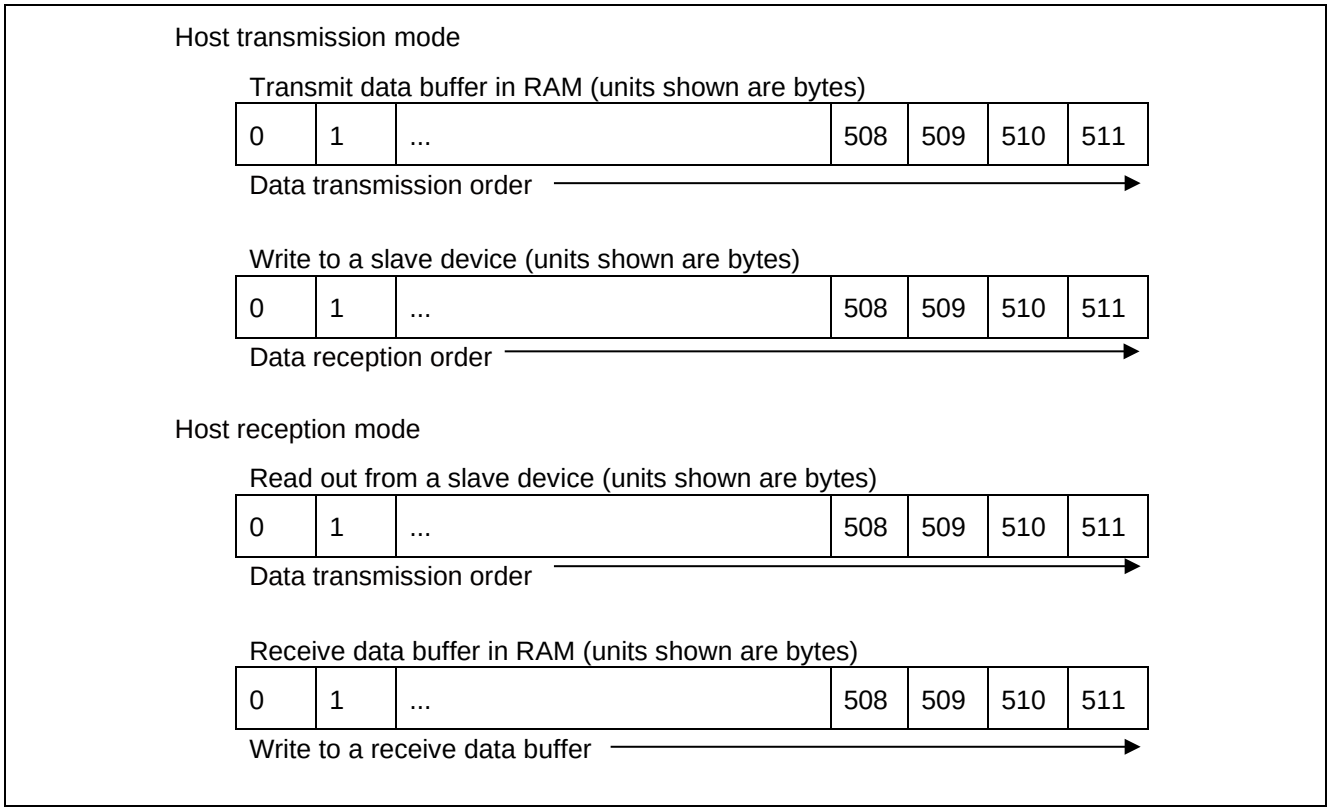


Figure 1.2 Transmission Data Storage

1.4.2.3 Operating Voltage Settings When Initialization

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

For an SD Card, call the R_SDC_SD_End() function and after it reaches the initializable state, the SD Card should be removed. After that, reinsert the card, set the operating voltage again, and perform initialization processing again.

For an SD module, call the R_SDC_SD_End() function and after it reaches the initializable state, stop supply of power to the SD module. After that, restart supply voltage supply to the SD module, set the operating voltage again, and perform the initialization processing again.

1.4.2.4 Stopping SDHI_CLK

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

1.4.2.5 SDHI Status Verification

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

Status Verification Methods

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

The follow status items can be verified.

- SD Card insertion/removal detection
- SD protocol
- SDHI interrupt

Table 1.6 lists the status items verified by the SD Card driver library functions.

Table 1.5 Status Items Verified

Type	Status	Remarks
SD Card insertion/removal (Interrupt enable/disable setting with the R_SDC_SD_CdInt() function)	SD Card inserted/removed state	Detection is possible with the R_SDC_SD_GetCardDetection() function.
SD protocol (Interrupt enable setting with the R_SDC_SD_Initialize() 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 SD Card insertion verification method, interrupts (the SDC_SD_MODE_HWINT setting) should also be selected for SD protocol status verification with the R_SDC_SD_Initialize() function.

Note that the R_SDHI_IntHandlerX() functions (where X is the channel number) of the SDHI FIT module are already registered in the system as the interrupt handlers for the SDHI interrupts.

SD Card Insertion Verification by Software Poling and Interrupt

The SD Card insertion state can be verified using the R_SDC_SD_GetCardDetection() function regardless of the enabled/disabled setting of the SD Card insertion interrupt.

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

SD Protocol Status Verification Using Software Poling

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

When software poling is set up, use the r_sdc_sd_int_wait() function, within that function call the SD status register 1 and 2 acquisition processing (the r_sdc_sd_get_intstatus() function), and check the values of the SD status registers 1 and 2 (SDSTS1 and SDSTS2).

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

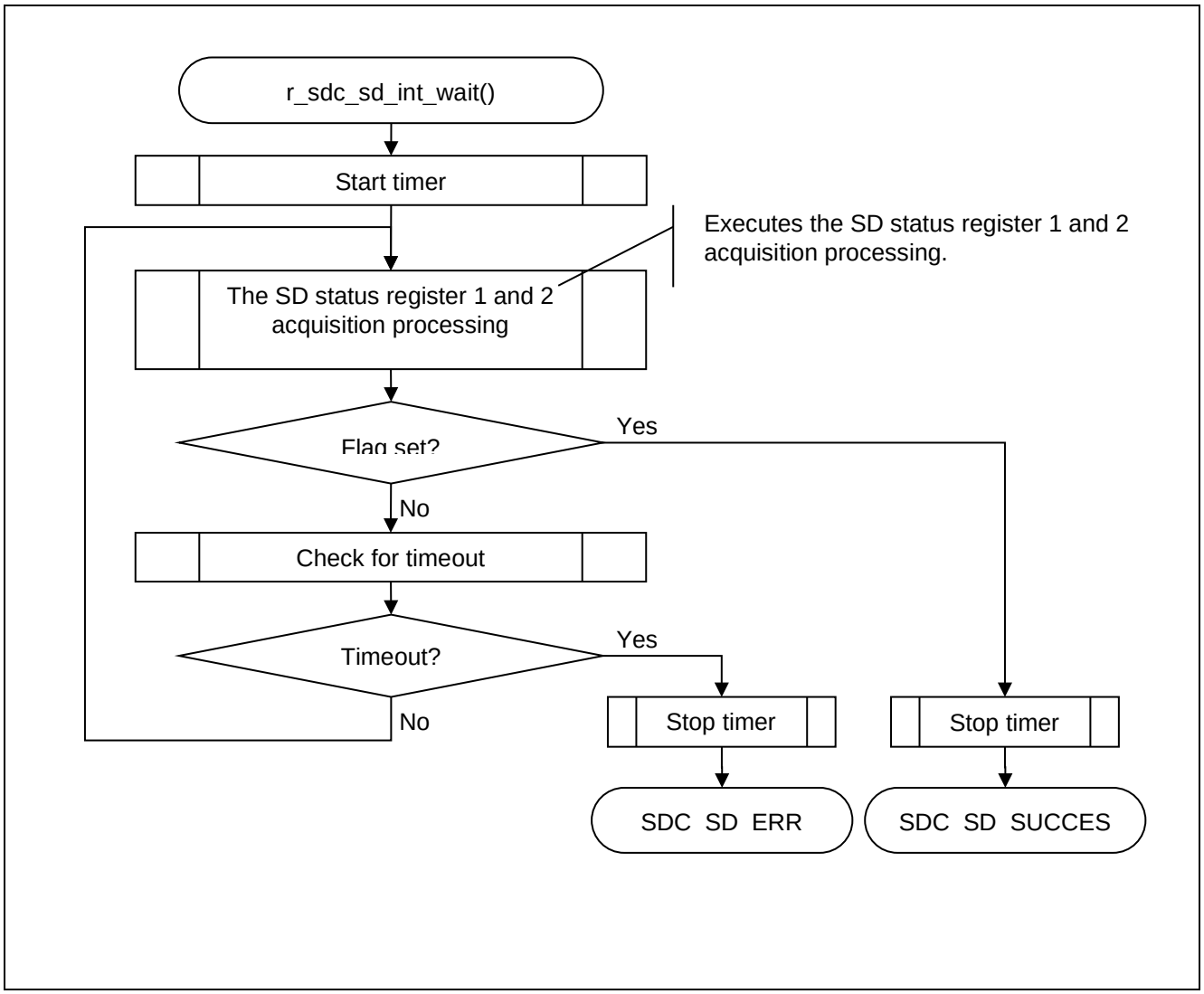


Figure 1.3 SD Protocol Status Verification Using Software Poling

SD Protocol Status Verification Using Interrupts

When interrupts (SDC_SD_MODE_HWINT) are set as the SD protocol status verification method with the R_SDC_SD_Initialize() 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 SD protocol status interrupt callback function with the R_SDC_SD_IntCallback() function.

When interrupt waiting is set up, applications use the r_sdc_sd_int_wait() function, and within this function acquire the information in SD status registers 1 and 2 (by calling the r_sdc_sd_get_intstatus() function), to verify the interrupt occurrence state.

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

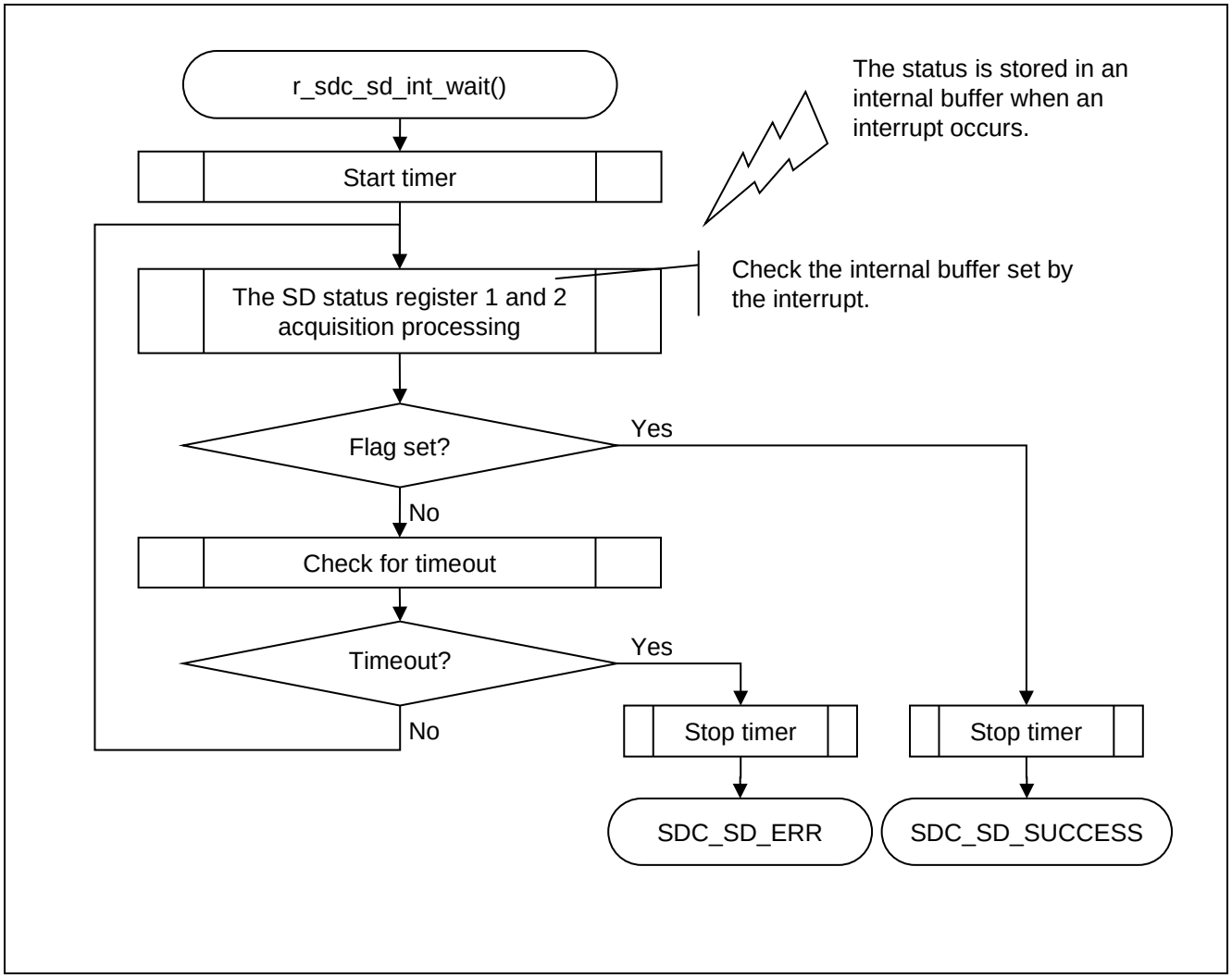


Figure 1.4 SD Protocol Status Verification Using Interrupts

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 SD Card and initialize the card again. See section 4.1, 4.2, for details on the processing related to SD Card insertion and removal. For the SD module, 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 SD Card driver, before removing and reinserting the SD 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 SD Card 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 SD memory card 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_SDC_SD_Open() function, call the following functions in the order shown. See section 3.22, R_SDC_SD_SetLogHdlAddress(), for a setup example.

1. R_LONGQ_Open()
2. R_SDC_SD_SetLogHdlAddress()

Set Up R_SDC_SD_Log()

Call this function to terminate error acquisition. See section 3.23, R_SDC_SD_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_SDC_SD_1msInterval() at 1 ms intervals. Note, however, that this is not required when the r_sdc_sd_int_wait() and r_sdc_sd_wait() functions in r_sdc_sd_config.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 SD Card 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.

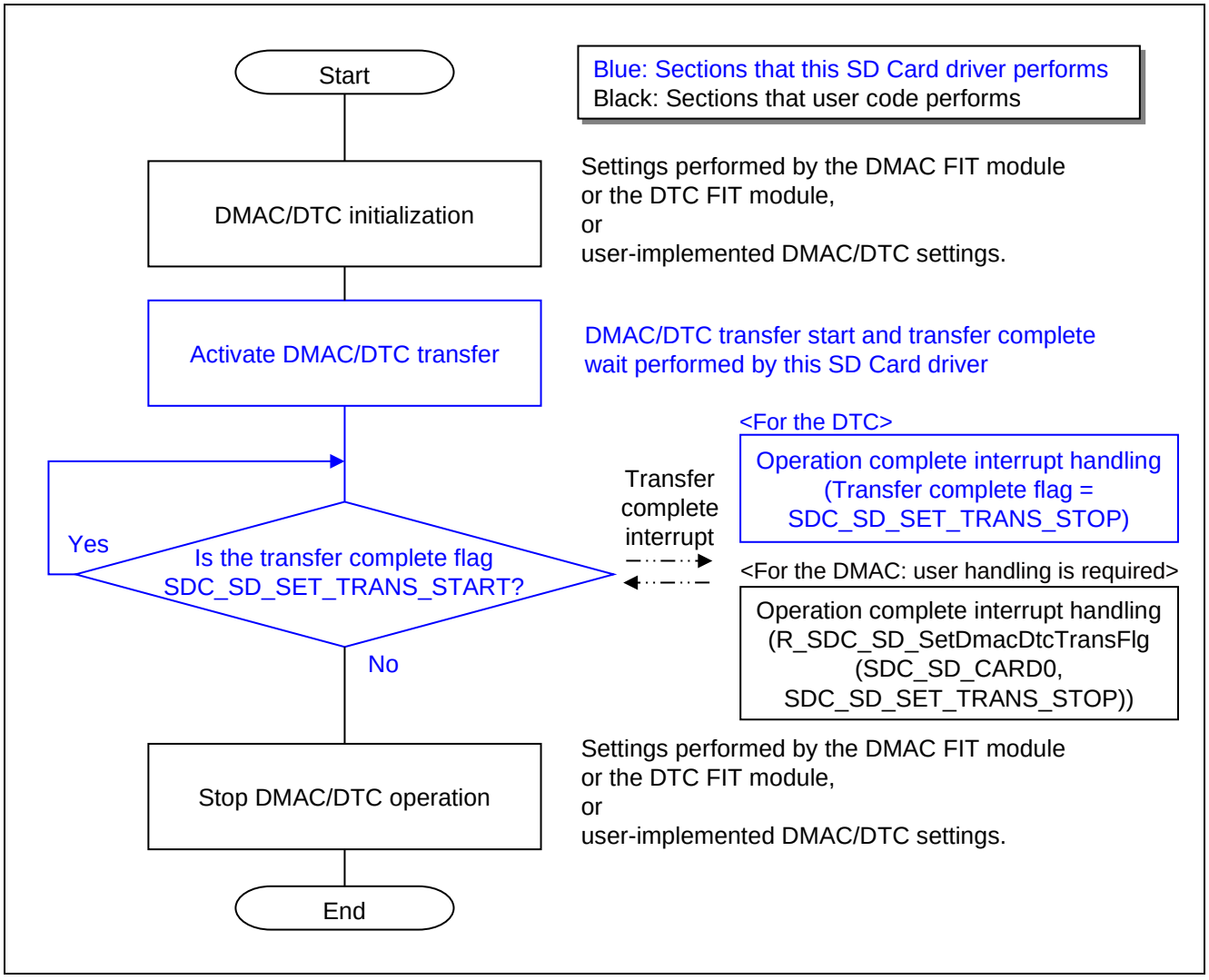
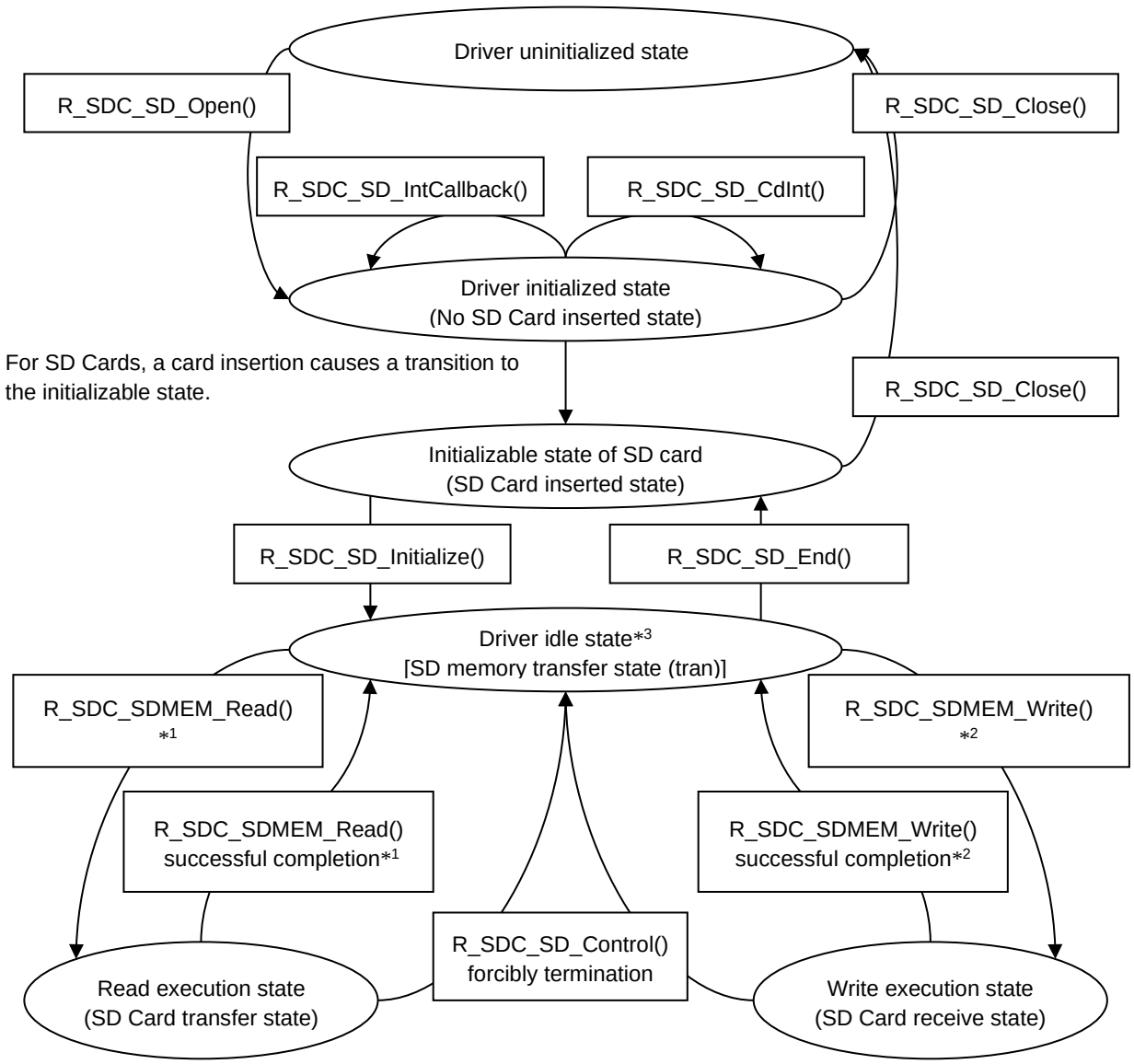


Figure 1.5 Processing for DMAC/DTC Transfer Setup

1.5 State Transition Diagram

Figure 1.6 shows the state transition diagram for this driver.



Notes: 1. The SD Card transfer state transition command processing, which includes the R_SDC_SDMEM_ReadSoftwareTrans() function, is similar.

2. The SD Card receive state transition command processing, which includes the R_SDC_SDMEM_WriteSoftwareTrans() function, is similar.

3. [ ] indicates the state of the SD memory card as described in SD Specifications, Part 1: Physical Layer Simplified Specification.

Figure 1.6 SD Memory Card 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 SD Card 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 SD Card Power Supply

After an SD Card is inserted, the power supply voltage stipulated by the SD Card specifications must be applied. See the Power Scheme section in the SD Specifications Part 1 Physical Layer Simplified Specification.

In particular, when reinserting control of the SD Card after removal of the SD Card or reentering control after turning off the SD Card, refer to the regulations on the voltage value and voltage sustain period, provide control timing for cutting / reentering by the system.

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

After the operating voltage is reached, call the R_SDC_SD_Initialize() initialize processing function. Applications must adjust the time between the application of the power supply voltage and the point when the operating voltage is reached.

Also, the application program must provide the wait time processing for the time from the point when supply of the supply voltage is stopped until a voltage that allows removal of the SD Card is reached.

1.6.3 Software Write Protection

This SD Card driver does not support software protection state control functions.

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:

- SDHI

2.2 Software Requirements

This SD Card driver depends on the following packages.

- r_bsp(Rev. 5.20 or more)
- r_sdhi_rx
- 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)
- r_longq_rx (Only when an error log acquisition function 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 SDHI_CFG_MODE_INT is set to SDHI_MODE_HWINT, SDHI interrupts are enabled. Put the system in the interrupts permitted state before the SD Memory Card driver's open processing, R_SDC_SD_Open(), is called. For details regarding SDHI interrupts, refer to the application note for the SDHI FIT module.

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

Table 2.1 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 2.1 SD Card Driver Library Functions that may be Called from Within an Interrupt Handler

Function	Functional Overview	Remarks
R_SDC_SD_Control()	Driver control processing	SDC_SD_SET_STOP (Forced stop request command)

2.5 Header Files

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

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

```
#include "r_sdc_sd_rx_if.h"
```

2.6 Integer Types

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

2.7 Configuration Overview

This SD Card driver configuration options are set in r_sdc_sd_rx_config.h. When using Smart Configurator, the configuration option can be set on software component configuration screen. The setting value is automatically reflected in r_sdc_sd_rx_config.h when adding modules to user project. The option names and setting values are listed in the table below.

Configuration options in *r_sdc_sd_rx_config.h*	
#define SDC_SD_CFG_STATUS_CHECK_MODE SDC_SD_MODE_HWINT Note: The default value is SDC_SD_MODE_HWINT "status verification: hardware interrupt".	This definition can be used as the argument p_sdc_sd_config->mode to the R_SDC_SD_Initialize() initialization processing function. Refer to the section "Initialization processing function R_SDC_SD_Initialize()" and define p_sdc_sd_config->mode.
#define SDC_SD_CFG_TRANSFER Note: The default value is SDC_SD_MODE_SW " data transfer type: software".	This definition can be used as the argument p_sdc_sd_config->mode to the R_SDC_SD_Initialize() initialization processing function. Refer to the section "Initialization processing function R_SDC_SD_Initialize()" and define p_sdc_sd_config->mode.
#define SDC_SD_CFG_ERROR_LOG_ACQUISITION Note: The default value is "0".	This must be set "1" when an error log acquisition function using the LONGQ FIT module is used. When this function is used, the LONGQ FIT module must be included.

2.8 Code Size

The sizes of ROM, RAM and maximum stack usage associated with this module are listed below. Information is listed for a single representative device of the RX200 Series, and RX600 Series, respectively.

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_sdc_sdmem_rx rev.3.00

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 "lang = c99" 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

ROM, RAM and Stack Code Sizes

Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
RX231	ROM	2,636 bytes	2,079 bytes	4,460 bytes	3,396 bytes	3,760 bytes	2,932 bytes
	RAM	12 bytes		12 bytes		8 bytes	
	STACK *1	164 bytes		-		124 bytes	
RX65N	ROM	5,595 bytes	4,424 bytes	9,588 bytes	7,324 bytes	7,777 bytes	6,147 bytes
	RAM	40 bytes		40 bytes		32 bytes	
	STACK *1	188 bytes		-		148 bytes	

Note 1. The sizes of maximum usage stack of Interrupts functions is included.

2.9 Parameters

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

(1) e_enum_sdc_sd_cmd Structure Definition

```
enum e_enum_sdc_sd_cmd
{
    SDC_SD_SET_STOP,
    SDC_SD_SET_BUFFER
} enum_sdc_sd_cmd_t;
```

(2) e_enum_sdc_sd_trans Structure Definition

```
enum e_enum_sdc_sd_trans
{
    SDC_SD_SET_TRANS_STOP,
    SDC_SD_SET_TRANS_START
} enum_sdc_sd_trans_t;
```

(3) sdc_sd_cmd_t Structure Definition

```
typedef struct
{
    uint32_t    cmd;
    uint32_t    mode;
    uint8_t     *p_buff;
    uint32_t    size;
}sdc_sd_cmd_t;
```

(4) sdc_sd_cfg_t Structure Definition

```
typedef struct
{
    uint32_t    mode;
    uint32_t    voltage;
}sdc_sd_cfg_t;
```

(5) sdc_sd_access_t Structure Definition

```
typedef struct
{
    uint8_t     *p_buff;
    uint32_t    lbn;
    int32_t     cnt;
    uint32_t    mode;
    uint32_t    write_mode;
}sdc_sd_access_t;
```

(6) sdc_sd_card_status_t Structure Definition

```
typedef struct
{
    uint32_t card_sector_size;
    uint32_t prot_sector_size;
    uint8_t  write_protect;
    uint8_t  csd_structure;
    uint8_t  speed_mode;
    uint8_t  io_speed_mode;
    uint8_t  rsv[3];
}sdc_sd_card_status_t;
```

(7) sdc_sd_card_reg_t Structure Definition

```
typedef struct
{
    uint32_t    sdio_ocr[1];
    uint32_t    ocr[1];
    uint32_t    cid[4];
    uint32_t    csd[4];
    uint32_t    dsr[1];
    uint32_t    rca[2];
    uint32_t    scr[2];
    uint32_t    sdstatus[4];
    uint32_t    swtich_func_status[5];
}sdc_sd_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_sdc_sd_rx_if.h along with the API function prototype declarations.

If an error occurs during processing, these SD Card driver library functions return an error code in their return value. Also, the error code can be acquired using the R_SDC_SD_GetErrCode() function after execution of the R_SDC_SD_Initialize(), R_SDC_SDMEM_Read(), and R_SDC_SDMEM_Write() functions*.

Table 2.3 lists the error codes. Note that values not listed in the table are reserved for future expansion.

Note: * The R_SDC_SDMEM_ReadSoftwareTrans(), R_SDC_SDMEM_ReadSoftwareTransSingleCmd(), and R_SDC_SDMEM_WriteSoftwareTrans() functions operate similarly.

Table 2.2 Error Codes

Macro Definition	Value	Meaning	
SDC_SD_SUCCESS_LOCKED_CARD	1	Successful operation	Successful operation but the SD Card was locked.
SDC_SD_SUCCESS	0	Successful operation	Successful operation
SDC_SD_ERR	-1	General error	R_SDC_SD_Open() function not yet executed, argument parameter error, and other errors
SDC_SD_ERR_WP	-2	Write protect error	Write to an SD Card in the write protected state
SDC_SD_ERR_RO	-3	Read only error	Read only error
SDC_SD_ERR_RES_TOE	-4	Response timeout error	A response to a command was not received within 640 clock cycles (SDHI clock).

Macro Definition	Value	Meaning	
SDC_SD_ERR_CARD_TOE	-5	Card timeout error	Card busy state timeout, data reception timeout after read command, or CRC status reception timeout after write command*. Note: * Depends on the setting of bits 7 to 4 (TOP) in the card access option register (SDOPT).
SDC_SD_ERR_END_BIT	-6	End bit error	No end bit was detected.
SDC_SD_ERR_CRC	-7	CRC error	The host detected a CRC error.
SDC_SD_ERR_CARD_RES	-8	Card response error	
SDC_SD_ERR_HOST_TOE	-9	Host timeout error	Error in the r_sdc_sd_int_wait() function.
SDC_SD_ERR_CARD_ERASE	-10	Card erase error	R1 response card status error (ERASE_SEQ_ERROR or ERASE_PARAM)
SDC_SD_ERR_CARD_LOCK	-11	Card lock error	R1 response card status error (CARD_IS_LOCKED)
SDC_SD_ERR_CARD_UNLOCK	-12	Card unlock error	R1 response card status error (LOCK_UNLOCK_FAILED)
SDC_SD_ERR_CARD_CRC	-13	Card CRC error	R1 response card status error (COM_CRC_ERROR)
SDC_SD_ERR_CARD_ECC	-14	Card ECC error	R1 response card status error (CARD_ECC_FAILED)
SDC_SD_ERR_CARD_CC	-15	Card CC error	R1 response card status error (CC_ERROR)
SDC_SD_ERR_CARD_ERROR	-16	Card error	R1 response card status error (ERROR)
SDC_SD_ERR_CARD_TYPE	-17	Unsupported card	The card was recognized as an unsupported card
SDC_SD_ERR_NO_CARD	-18	No card inserted error	No card is inserted
SDC_SD_ERR_ILL_READ	-19	Illegal read error	An illegal SD buffer register read method was used
SDC_SD_ERR_ILL_WRITE	-20	Illegal write error	An illegal SD buffer register write method was used
SDC_SD_ERR_AKE_SEQ	-21	Card AKE error	R1 response card status error (AKE_SEQ_ERROR)
SDC_SD_ERR_OVERWRITE	-22	Card OVERWRITE error	R1 response card status error (CSD_OVERWRITE)
SDC_SD_ERR_CPU_IF	-30	Target microcontroller interface function error	Target microcontroller interface function error (other than the r_sdc_sd_int_wait() function)
SDC_SD_ERR_STOP	-31	Forced stop error	Forced stop state due to the R_SDC_SD_Control() function
SDC_SD_ERR_CSD_VER	-50	Version error	CSD register version error
SDC_SD_ERR_FILE_FORMAT	-52	File format error	CSD register file format error
SDC_SD_ERR_ILL_FUNC	-60	Function number error	Error that occurs when the function number requested is invalid
SDC_SD_ERR_IFCOND_VOLT	-71	Interface condition: voltage error	The interface condition voltage is incorrect

Macro Definition	Value	Meaning	
SDC_SD_ERR_IFCOND_ECHO	-72	Interface condition: echo back error	The interface condition echo back pattern is incorrect
SDC_SD_ERR_OUT_OF_RANGE	-80	Argument range error	R1 response card status error (OUT_OF_RANGE)
SDC_SD_ERR_ADDRESS_ERROR	-81	Address error	R1 response card status error (ADDRESS_ERROR)
SDC_SD_ERR_BLOCK_LEN_ERRO R	-82	Block length error	R1 response card status error (BLOCK_LEN_ERROR)
SDC_SD_ERR_ILLEGAL_COMMAN D	-83	Illegal command error	R1 response card status error (ILLEGAL_COMMAND)
SDC_SD_ERR_CMD_ERROR	-86	Command index error	Internal error (The transmission command index and the reception command index differ)
SDC_SD_ERR_CBSY_ERROR	-87	Command error	SDHI internal error (Command busy)
SDC_SD_ERR_NO_RESP_ERROR	-88	No response error	SDHI internal error (Response could not be received)
SDC_SD_ERR_ADDRESS_BOUNDA RY	-89	Buffer address error	Argument buffer address error The address does not fall on a 4-byte boundary
SDC_SD_ERR_UNSUPPORTED_TY PE	-97	Unsupported SDIO access error	Unsupported SDIO access error
SDC_SD_ERR_API_LOCK	-98	API lock error	Error that occurs when an API is called while another is being called
SDC_SD_ERR_INTERNAL	-99	Internal error	Internal driver error

2.11 Callback Function

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

For information regarding how to register callback functions, see "3.16, R_SDC_SD_CdInt()" and "3.17, R_SDC_SD_IntCallback()".

For the timing at which the callback function occurs, see "1.4.2.5 SDHI 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_SDC_SD_Open()

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

Format

```
sdc_sd_status_t R_SDC_SD_Open(
  uint32_t card_no,
  uint32_t card_no,
  void *p_sdc_sd_workarea
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

channel

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

**p_sdc_sd_workarea*

Pointer to a working area on a 4-byte boundary (area size: 200 bytes)

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*
SDC_SD_ERR_CPU_IF	*Target microcontroller interface error*
SDC_SD_ERR_ADDRESS_BOUNDARY	*Argument buffer address error*

Properties

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

Description

Gets the SDHI channel resource controlled by the argument card_no and initializes the SD Card driver specified by the argument channel and the SDHI FIT module. Also, this function exclusively acquires that SDHI channel resource.

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

Example

```
uint32_t       g_sdc_sd_work[200/sizesof(uint32_t)];

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

if (R_SDC_SD_Open(SDC_SD_CARD0, &g_sdc_sd_work) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

The pins must be set up before this function is called. See section 4.1, SD Card Insertion and Power-On Timing, for details.

If this function does not complete successfully, do not call any library functions other than R_SDC_SD_GetVersion(), R_SDC_SD_Log(), or R_SDC_SD_SetLogHdlAddress().

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

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

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

3.2 R_SDC_SD_Close()

This function releases the resources being used by the SD Card driver.

Format

```
sdc_sd_status_t R_SDC_SD_Close(
  uint32_t card_no
)
```

Parameters

card_no

SD Card number	The number of the SD Card used (numbering starts at 0)

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

This function terminates all SD Card driver processing and releases the resources for the SDHI channel specified in the argument card_no.

That SDHI 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_SDC_SD_Open() function is not used after this function has been executed. This are may be used for other purposes.

Reentrant

This function is reentrant with respect to other channels.

Example

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

if (R_SDC_SD_Close(SDC_SD_CARD0) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

The pins must be set up after this function is called. See section 4.2, SD Card Removal and Power-Off Timing, for details. Before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

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

3.3 R_SDC_SD_GetCardDetection()

This function verifies the SD Card insertion state.

Format

```
sdc_sd_status_t R_SDC_SD_GetCardDetection(
  uint32_t card_no
)
```

Parameters

card_no

SD Card number	The number of the SD Card used (numbering starts at 0)

Return Values

SDC_SD_SUCCESS	*The SDHI_CD pin was at the low level or card detection was invalid.*
SDC_SD_ERR	*The SDHI_CD pin was at the high level.*

Properties

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

Description

This function verifies the SD Card insertion state.

<When SDHI_CFG_CHx_CD_ACTIVE == 1 (card detection enabled)>

If the SDHI_CD pin is low, this function returns SDC_SD_SUCCESS.

If the SDHI_CD pin is high, this function returns SDC_SD_ERR.

<When SDHI_CFG_CHx_CD_ACTIVE == 0 (card detection disabled)>

This function will always return SDC_SD_SUCCESS.

Example

```
if (R_SDC_SD_GetCardDetection(SDC_SD_CARD0) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

To enable card detection, set #define SDHI_CFG_CHx_CD_ACTIVE to 1 in the SDHI FIT module.

When using the card insertion detection function, pin setting is necessary after this function is executed. See section 4.1, SD Card Insertion and Power-On Timing, for details. Before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

When using with card removal detection, pin setting is required before this function is executed. See section 4.2, SD Card Removal and Power-Off Timing, for details.

The SDHI_CD pin, which is connected to the SD Card socket CD pin, is used as the SD Card insertion detection pin.

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

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

3.4 R_SDC_SD_Initialize()

This function initializes the SD Card and transitions the state from the SD Card initialization state to the driver idle state.

Format

```
sdc_sd_status_t R_SDC_SD_Initialize(
  uint32_t card_no,
  sdc_sd_cfg_t *p_sdc_sd_config,
  uint32_t init_type
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_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, SD Card Driver Operating Mode (mode).

voltage: The power supply voltage

Specify the voltage supplied to the SD Card (For the setting values, see the macro definitions in Table 3.2, Supply Voltages (voltage). An SD Card 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 SD Card Driver Operating Mode (mode)

Type	Macro Definition	Value (Bits)	Definition
SD Protocol Status Verification	SDC_SD_MODE_POLL	0x0000	Software polling
	SDC_SD_MODE_HWINT	0x0001	Hardware interrupt
Data transfer type	SDC_SD_MODE_SW	0x0000	Software transfers
	SDC_SD_MODE_DMA*1*4	0x0002	DMAC transfers*3
	SDC_SD_MODE_DTC*2*4	0x0004	DTC transfers*3
Media support type	SDC_SD_MODE_MEM	0x0000	SD Memory Card/SD memory
SD bus support type	SDC_SD_MODE_4BIT	0x0200	SD mode 4-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 functions used.
4. Do not set up SDC_SD_MODE_DMA and SDC_SD_MODE_DTC at the same time.

Table 3.2 Supply Voltage (voltage)

Supply Voltage [V]	Macro Definition	Value (Bits)
2.7-2.8	SDC_SD_VOLT_2_8	0x00008000
2.8-2.9	SDC_SD_VOLT_2_9	0x00010000
2.9-3.0	SDC_SD_VOLT_3_0	0x00020000
3.0-3.1	SDC_SD_VOLT_3_1	0x00040000
3.1-3.2	SDC_SD_VOLT_3_2	0x00080000
3.2-3.3	SDC_SD_VOLT_3_3	0x00100000
3.3-3.4	SDC_SD_VOLT_3_4	0x00200000
3.4-3.5	SDC_SD_VOLT_3_5	0x00400000
3.5-3.6	SDC_SD_VOLT_3_6	0x00800000

init_type: Initialization type

Specify the initialization target. As the value, use the media support type macro definition listed in Table 3.1, SD Card Driver Operating Mode (mode).

Return Values

```
SDC_SD_SUCCESS              Successful operation
SDC_SD_SUCCESS_LOCKED_CARD  Successful operation, and, furthermore,
                            SD Card 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_sdc_sd_rx_if.h.

Description

This function performs SD Card initialize processing. Applications should call this function after detecting an SD Card.

When the card is recognized as an SD Card, the internal pulled-up function of CD/DAT3 pin in SD Card will be disabled.

When the return value is SDC_SD_SUCCESS, the SD Card will have transitioned to the transfer state (tran) and SD Card read and write access will be possible. When the return value is SDC_SD_SUCCESS_LOCKED_CARD, the SD Card will have transitioned to the transfer state (tran), but SD Card read and write access will not be possible. In this case, SDC_SD_ERR_CARD_LOCK will be registered as the error code. A locked SD Card only accepts certain commands. To read or write to a locked SD Card, the SD Card must first be set to the unlocked state and then must be initialized again.

Example

```
sdc_sd_cfg_t      sdc_sd_config;

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

sdc_sd_config.mode = SDC_SD_CFG_DRIVER_MODE;
sdc_sd_config.voltage = SDC_SD_VOLT_3_3;
if (R_SDC_SD_Initialize(SDC_SD_CARD0, &sdc_sd_config, SDC_SD_MODE_MEM) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

This SD Card driver sets Default Speed mode when initializing.

The pins must be set up before executing this function. See section 4.1, SD Card Insertion and Power-On Timing, for details. Also, before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

If this function returns an error, after setting the hardware to the SD Card initialization possible state by calling the R_SDC_SD_End() function, perform the initialization processing again.

After initialization completes, perform end processing before running reinitialization processing.

When voltage in p_sdc_sd_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_SDC_SD_CdInt() function is used, set SDC_SD_MODE_HWINT as the p_sdc_sd_config mode status.

The SDHI only supports 3.3 V signal levels. Therefore, this SD Card driver sets the S18R bit in the command arguments to 0 during SD Card initialization and issues either an ACMD41 command.

3.5 R_SDC_SD_End()

This function clears the value of the work area and transitions from the driver idle state to the SD Card initialization possible state. Running this function does not change the state of the SD Card.

Format

```
sdc_sd_status_t R_SDC_SD_End(
  uint32_t card_no,
  uint32_t end_type
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

end_type

End type

Specify the end target. As the value, use the macro definition of a mode supported by the media listed in Table 3.1, SD Card Driver Operating Mode (mode), in the description of the R_SDC_SD_Initialize() function.

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

This function performs SD Card end processing.

For SD Cards, it switches to the SD Card removable state. Note that even if this function has been called and the SD Card initialized state has been cleared, the SD Card insertion interrupt and the SD Card insertion verification interrupt callback function remain enabled.

Example

```
if (R_SDC_SD_End(SDC_SD_CARD0, SDC_SD_MODE_MEM) != SDC_SD_SUCCESS)
{
   /* Error */
}
/* ==== Please add the processing to set the pins. ==== */
```

Special Notes

If the SD Card is removed after this function has been called, the pins must be set up. See section 4.2, SD Card Removal and Power-Off Timing, for details. Also, before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

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

3.6 R_SDC_SDMEM_Read()

This function performs read processing.

Format

```
sdc_sd_status_t R_SDC_SDMEM_Read(
  uint32_t card_no,
  sdc_sd_access_t *p_sdc_sd_access
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_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)

write_mode: Write mode (Does not need to be set)

Return Values

SDC_SD_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_sdc_sd_rx_if.h.

Description

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

If SD Card removal is detected at the start of this function's execution, processing is interrupted and processing is terminated and an error (SDC_SD_ERR_STOP) is returned.

If a forced stop request due to an R_SDC_SD_Control() function SDC_SD_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 (SDC_SD_ERR_STOP) is returned.

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

Up to 2 blocks: READ_SINGLE_BLOCK command (CMD17)

3 blocks or more: READ_MULTIPLE_BLOCK command (CMD18)

Reentrant

This function is reentrant with respect to other channels.

Example

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

sdc_sd_access_t   sdc_sd_access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn    = 0x10000000;
sdc_sd_access.cnt    = TEST_BLOCK_CNT;

if(R_SDC_SDMEM_Read(SDC_SD_CARD0, &sdc_sd_access) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

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

This function disables the SD Card insertion/removal interrupt during the data transfer period for DMAC or DTC transfers.

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_SDC_SDMEM_ReadSoftwareTrans()

This function performs read processing (software transfers).

Format

```
sdc_sd_status_t R_SDC_SDMEM_ReadSoftwareTrans(
  uint32_t card_no,
  sdc_sd_access_t *p_sdc_sd_access
)
```

Parameters

card_no

SD Card number The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_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)

write_mode: Write mode (Does not need to be set)

Return Values

SDC_SD_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_sdc_sd_rx_if.h.

Description

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

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

If SD Card removal is detected at the start of this function’s execution, processing is interrupted and processing is terminated and an error (SDC_SD_ERR_STOP) is returned.

If a forced stop request due to an R_SDC_SD_Control() function SDC_SD_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 (SDC_SD_ERR_STOP) is returned.

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

Up to 2 blocks: READ_SINGLE_BLOCK command (CMD17)

3 blocks or more: READ_MULTIPLE_BLOCK command (CMD18)

Reentrant

This function is reentrant with respect to other channels.

Example

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

sdc_sd_access_t   sdc_sd_access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn    = 0x10000000;
sdc_sd_access.cnt    = TEST_BLOCK_CNT;

if(R_SDC_SDMEM_ReadSoftwareTrans(SDC_SD_CARD0, &sdc_sd_access) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, driver open processing must be performed by the R_SDC_SD_Open() function and initialization by the R_SDC_SD_Initialize() 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_SDC_SDMEM_ReadSoftwareTransSingleCmd()

This function performs read processing (CMD17 single software transfers).

Format

```
sdc_sd_status_t R_SDC_SDMEM_ReadSoftwareTransSingleCmd(
  uint32_t card_no,
  sdc_sd_access_t *p_sdc_sd_access
)
```

Parameters

card_no

SD Card number The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_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)

write_mode: Write mode (Does not need to be set)

Return Values

SDC_SD_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_sdc_sd_rx_if.h.

Description

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

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

If SD Card removal is detected at the start of this function's execution, processing is interrupted and processing is terminated and an error (SDC_SD_ERR_STOP) is returned.

If a forced stop request due to an R_SDC_SD_Control() function SDC_SD_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 (SDC_SD_ERR_STOP) is returned.

Only the CMD17 command is used for reading out block data.

Reentrant

This function is reentrant with respect to other channels.

Example

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

sdc_sd_access_t   sdc_sd_access;
uint32_t       g_test_r_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn    = 0x10000000;
sdc_sd_access.cnt    = TEST_BLOCK_CNT;

if(R_SDC_SDMEM_ReadSoftwareTransSingleCmd(SDC_SD_CARD0, &sdc_sd_access) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

3.9 R_SDC_SDMEM_Write()

This function performs write processing.

Format

```
sdc_sd_status_t R_SDC_SDMEM_Write(
  uint32_t card_no,
  sdc_sd_access_t *p_sdc_sd_access
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_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)

write_mode: Write mode

Set this parameter to one of the macro definitions shown in Table 3.3, SD Card Driver Write Mode (write_mode).

Return Values

SDC_SD_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_sdc_sd_rx_if.h.

Description

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

If SD Card removal is detected at the start of this function's execution, processing is interrupted and processing is terminated and an error (SDC_SD_ERR_STOP) is returned.

If a forced stop request due to an R_SDC_SD_Control() function SDC_SD_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 (SDC_SD_ERR_STOP) is returned.

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

Up to 2 blocks: WRITE_SINGLE_BLOCK command (CMD24)

3 blocks or more: WRITE_MULTIPLE_BLOCK command (CMD25)

Table 3.3 SD Card Driver Write Mode (write_mode)

Type	Macro Definition	Value (Bits)
Write with pre-erase	SDC_SD_WRITE_WITH_PREERASE	0x00000000
Normal write	SDC_SD_WRITE_OVERWRITE	0x00000001

Reentrant

This function is reentrant with respect to other channels.

Example

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

sdc_sd_access_t   sdc_sd_access;
uint32_t       g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff = (uint8_t *)&g_test_w_buff[0];
sdc_sd_access.lbn    = 0x10000000;
sdc_sd_access.cnt    = TEST_BLOCK_CNT;
sdc_sd_access.write_mode= SDC_SD_WRITE_OVERWRITE;

if(R_SDC_SDMEM_Write(SDC_SD_CARD0, &sdc_sd_access) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

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

If an SD Card removal occurs during write processing when SDC_SD_WRITE_WITH_PREERASE is specified, there is a higher probability that SD memory data will be lost than when SDC_SD_WRITE_OVERWRITE is specified.

This function disables the SD Card insertion/removal interrupt during the data transfer period for DMAC or DTC transfers.

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_SDC_SDMEM_WriteSoftwareTrans()

This function performs write processing (software transfers).

Format

```
sdc_sd_status_t R_SDC_SDMEM_WriteSoftwareTrans(
  uint32_t card_no,
  sdc_sd_access_t *p_sdc_sd_access
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_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)

write_mode: Write mode

Set this parameter to one of the macro definitions shown in Table 3.4, SD Card Driver Write Mode (write_mode).

Return Values

SDC_SD_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_sdc_sd_rx_if.h.

Description

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

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

If SD Card removal is detected at the start of this function's execution, processing is interrupted and processing is terminated and an error (SDC_SD_ERR_STOP) is returned.

If a forced stop request due to an R_SDC_SD_Control() function SDC_SD_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 (SDC_SD_ERR_STOP) is returned.

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

Up to 2 blocks: WRITE_SINGLE_BLOCK command (CMD24)

3 blocks or more: WRITE_MULTIPLE_BLOCK command (CMD25)

Table 3.4 SD Card Driver Write Mode (write_mode)

Type	Macro Definition	Value (Bits)
Write with pre-erase	SDC_SD_WRITE_WITH_PREERASE	0x00000000
Normal write	SDC_SD_WRITE_OVERWRITE	0x00000001

Reentrant

This function is reentrant with respect to other channels.

Example

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

sdc_sd_access_t   sdc_sd_access;
uint32_t       g_test_w_buff[(TEST_BLOCK_CNT*BLOCK_NUM)/sizeof(uint32_t)];

sdc_sd_access.p_buff = (uint8_t *)&g_test_w_buff[0];
sdc_sd_access.lbn    = 0x10000000;
sdc_sd_access.cnt    = TEST_BLOCK_CNT;
sdc_sd_access.write_mode= SDC_SD_WRITE_OVERWRITE;

if(R_SDC_SDMEM_WriteSoftwareTrans(SDC_SD_CARD0, &sdc_sd_access) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

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

If an SD Card removal occurs during write processing when SDC_SD_WRITE_WITH_PREERASE is specified, there is a higher probability that SD memory data will be lost than when SDC_SD_WRITE_OVERWRITE is specified.

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.

3.11 R_SDC_SD_Control()

This function performs driver control processing.

Format

```
sdc_sd_status_t R_SDC_SD_Control(
  uint32_t card_no,
  sdc_sd_cmd_t *p_sdc_sd_cmd
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

p_sdc_sd_cmd

Control information structure

cmd: Command macro definition

mode: Mode

**p_buff: Transfer buffer pointer*

size: Transfer size

Return Values

SDC_SD_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_sdc_sd_rx_if.h.

Description

This is an SD Card control utility.

Table 3.5, Commands, lists the commands that can be controlled. These commands are described individually starting on the following page.

Table 3.5 Commands

Command macro Definition cmd	Mode mode	Transfer Content *p_buff	Transfer Size size	Control Performed
SDC_SD_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.

Reentrant

This function is reentrant with respect to other channels.

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_SDC_SD_Open() function and initialization by the R_SDC_SD_Initialize() function.

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

(a) SDC_SD_SET_STOP

This function forcibly stops read/write processing.

Return Values

SDC_SD_SUCCESS	*Successful operation*

Description

This function requests a forced stop and transitions the SD Card driver to the forced stop state.

This function can be called from within an interrupt handler when an application program wants to stop processing.

<For SD Memory>

When there is a data transfer to or from the SD Card in progress, a CMD12 is issued to transition the SD Card 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 SD Card 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 SD Card 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 SD Card still in the forced stop request state. Therefore, an error end (SDC_SD_ERR_STOP) is returned when the R_SDC_SDMEM_Read() function or R_SDC_SDMEM_Write() function* is called.

Example

```
sdc_sd_cmd_t      sdc_sd_cmd;

sdc_sd_cmd.cmd = SDC_SD_SET_STOP;
sdc_sd_cmd.mode = 0;
sdc_sd_cmd.p_buff = 0;
sdc_sd_cmd.size = 0;

if (R_SDC_SD_Control(SDC_SD_CARD0, &sdc_sd_cmd) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

When a forced stop is performed during write processing, the SD Card 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 SD Card.
(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_SDC_SDMEM_Read() or R_SDC_SDMEM_Write()* function.
(2) When the R_SDC_SDMEM_Read() or R_SDC_SDMEM_Write()* 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_SDC_SDMEM_ReadSoftwareTrans(), R_SDC_SDMEM_ReadSoftwareTransSingleCmd(), and R_SDC_SDMEM_WriteSoftwareTrans() functions operate similarly.

3.12 R_SDC_SD_GetModeStatus()

This function acquires the transfer mode status.

Format

```
sdc_sd_status_t R_SDC_SD_GetModeStatus(
  uint32_t card_no,
  uint8_t *p_mode
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_mode*

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

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

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

Reentrant

This function is reentrant with respect to other channels.

Example

```
uint8_t * p_mode;

if(R_SDC_SD_GetModeStatus(SDC_SD_CARD0, p_mode) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

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

3.13 R_SDC_SD_GetCardStatus()

This function acquires the card status information.

Format

```
sdc_sd_status_t R_SDC_SD_GetCardStatus(
  uint32_t card_no,
  sdc_sd_card_status_t *p_sdc_sd_card_status
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_card_status*

Card status information structure pointer

card_sector_size: User area block count

prot_sector_size: Protected area block count

write_protect: Write protect information (See Table 3.6, Write Protect Information (write_protect).)

media_type: Media type (See Table 3.7, Media Type (media_type).)

csd_structure: CSD information

0: Standard Capacity card (SDSC)

1: High Capacity card (SDHC, SDXC)

speed_mode: Reserved

io_speed_mode: Reserved

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

This function gets the card status information of the SD Card and stores it in a card status information structure.

Table 3.6 Write Protect Information (write_protect)

Macro Definition	Value (Bits)	Definition
SDC_SD_WP_OFF	0x00	Write protect released state
SDC_SD_WP_HW	0x01	Hardware write protect state
SDC_SD_WP_TEMP	0x02	The TEMP_WRITE_PROTECT bit in the CSD register is set.
SDC_SD_WP_PERM	0x04	The PERM_WRITE_PROTECT bit in the CSD register is set.
SDC_SD_WP_ROM	0x10	SD ROM

Table 3.7 Media Type (media_type)

Macro Definition	Value (Bits)	Definition
SDC_SD_MEDIA_UNKNOWN	0x00	Unknown
SDC_SD_MEDIA_SDMEM	0x20	SD Memory Card/SD memory
SDC_SD_MEDIA_SDIO	0x01	SDIO Card/SDIO
SDC_SD_MEDIA_COMBO	0x21	SD Combo card (Logical OR of SD memory and SDIO)

Reentrant

This function is reentrant with respect to other channels.

Example

```
sdc_sd_card_status_t    sdc_sd_card_status;

if (R_SDC_SD_GetCardStatus(SDC_SD_CARD0, &sdc_sd_card_status) !=
SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

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

3.14 R_SDC_SD_GetCardInfo()

This function acquires the SD Card register information.

Format

```
sdc_sd_status_t R_SDC_SD_GetCardInfo(
  uint32_t card_no,
  sdc_sd_card_reg_t *p_sdc_sd_card_reg
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_sdc_sd_card_reg*

SD Card register information structure pointer

sdio_ocr[1]: SDIO OCR information
ocr[1]: SD memory OCR information
cid[4]: SD memory CID information
csd[4]: SD memory CSD information
dsr[1]: SD memory DSR information
rca[2]: SDIO and SD memory RCA information
scr[2]: SD memory SCR information
sdstatus[4]: SD memory SD status information
switch_func_status[5]: Reserved

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

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

Reentrant

This function is reentrant with respect to other channels.

Example

```
sdc_sd_card_reg_t    sdc_sd_card_reg;

if (R_SDC_SD_GetCardInfo(SDC_SD_CARD0, &sdc_sd_card_reg) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

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

3.15 R_SDC_SDMEM_GetSpeed()

This function acquires the SD Card Speed Class information and the performance move information.

Format

```
sdc_sd_status_t R_SDC_SDMEM_GetSpeed(
  uint32_t card_no,
  uint8_t *p_clss,
  uint8_t *p_move
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_clss*

Pointer to a Speed Class information structure (1 byte) (See Table 3.8, Speed Class Information.)

**p_move*

Pointer to a performance move information structure (1 byte) (See Table 3.9, Performance Move Information.)

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

Gets the Speed Class information and Performance Move information of the SD card.

Table 3.8 Speed Class Information

Macro Definition	Value (Bits)	Definition
SDC_SD_SPEED_CLASS_0	0x00	Speed Class 0
SDC_SD_SPEED_CLASS_2	0x01	Speed Class 2
SDC_SD_SPEED_CLASS_4	0x02	Speed Class 4
SDC_SD_SPEED_CLASS_6	0x03	Speed Class 6
SDC_SD_SPEED_CLASS_10	0x04	Speed Class 10
(No corresponding definitions)	0x05 to 0xFF	Reserved

Note: The values and definitions are the same as those in the SD Specifications Part 1 Physical Layer Simplified Specification.

Table 3.9 Performance Move Information

Value	Definition
0x00	Sequential write
0x01	1 MB/sec.
0x02	2 MB/sec.
...	...
0xFE	254 MB/sec.
0xFF	Infinity

Note: The values and definitions are the same as those in the SD Specifications Part 1 Physical Layer Simplified Specification.

Reentrant

This function is reentrant with respect to other channels.

Example

```
uint8_t     clss;
uint8_t     move;

if (R_SDC_SDMEM_GetSpeed(SDC_SD_CARD0, &clss, &move) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

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

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

3.16 R_SDC_SD_CdInt()

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

Format

```
sdc_sd_status_t R_SDC_SD_CdInt(
  uint32_t card_no,
  int32_t enable,
  sdc_sd_status_t (*callback)(int32_t)
)
```

Parameters

card_no

SD Card number The number of the SD Card used (numbering starts at 0)

enable: Specifies enable/disable of the SD Card insertion interrupt.

When SDC_SD_CD_INT_ENABLE is specified, the SD Card insertion interrupt is enabled.

When SDC_SD_CD_INT_DISABLE is specified, the SD 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 SD Card is inserted.

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

0: SDC_SD_CD_INSERT (A falling edge on the SDHI_CD pin was detected)

1: SDC_SD_CD_REMOVE (A rising edge on the SDHI_CD pin was detected)

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

This function sets up the SD 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 SD Card insertion interrupt occurs.

Note that the SD Card insertion state can be verified with the R_SDC_SD_GetCardDetection() function regardless of the enabled/disabled state of the SD Card insertion interrupt.

Reentrant

This function is reentrant with respect to other channels.

Example

```
/* Callback function */
sdc_sd_status_t r_sdc_sd_cd_callback(int32_t cd)
{
    if(cd & SDC_SD_CD_INSERT)
    {
        /* sdcard in */
    }
    else
    {
        /* sdcard out */
    }
    return SDC_SD_SUCCESS;
}


/* main */
void main(void)
{
   if (R_SDC_SD_CdInt(SDC_SD_CARD0, SDC_SD_CD_INT_ENABLE,
r_sdc_sd_cd_callback) != SDC_SD_SUCCESS)
   {
      /* Error */
   }
}
```

Special Notes

To enable card detection, set #define SDHI_CFG_CHx_CD_ACTIVE to 1 in the SDHI FIT module.

Before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

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

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

3.17 R_SDC_SD_IntCallback()

This function registers an SD protocol status interrupt callback function.

Format

```
sdc_sd_status_t R_SDC_SD_IntCallback(
  uint32_t card_no,
  sdc_sd_status_t (*callback)(int32_t)
)
```

Parameters

card_no

SD Card number — The number of the SD Card 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_SDC_SD_Initialize() function is executed.

The value 0 is always stored in (int32_t).

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

This function registers an SD 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 SD protocol status.

Reentrant

This function is reentrant with respect to other channels.

Example

```
/* Callback function */
sdc_sd_status_t r_sdc_sd_callback(int32_t channel)
{
   /* User program */

   return SDC_SD_SUCCESS;
}

if (R_SDC_SD_IntCallback(SDC_SD_CARD0, r_sdc_sd_callback) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

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

The callback function registered by this function differs from the SD Card insertion interrupt callback function.

The callback function registered by this function is not called when an SD Card insertion interrupt occurs.

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

3.18 R_SDC_SD_GetErrCode()

This function acquires the driver error codes.

Format

```
sdc_sd_status_t R_SDC_SD_GetErrCode(
  uint32_t card_no
)
```

Parameters

card_no

SD Card number	The number of the SD Card used (numbering starts at 0)

Return Values

Error code	*See the error code documentation.*

Properties

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

Description

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

Note: * The R_SDC_SDMEM_ReadSoftwareTrans(), R_SDC_SDMEM_ReadSoftwareTransSingleCmd(), and R_SDC_SDMEM_WriteSoftwareTrans() functions operate similarly.

Reentrant

This function is reentrant with respect to other channels.

Example

```
sdc_sd_cfg_t      sdc_sd_config;
sdc_sd_status_t   error_code = SDC_SD_SUCCESS;

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

sdc_sd_config.mode = SDC_SD_CFG_DRIVER_MODE;
sdc_sd_config.voltage = SDC_SD_VOLT_3_3;
if (R_SDC_SD_Initialize(SDC_SD_CARD0, &sdc_sd_config) != SDC_SD_SUCCESS)
{
   /* Error */
   error_code = R_SDC_SD_GetErrCode(SDC_SD_CARD0);
}
```

Special Notes

Before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

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

3.19 R_SDC_SD_GetBuffRegAddress()

This function acquires the address of the SD buffer register.

Format

```
sdc_sd_status_t R_SDC_SD_GetBuffRegAddress(
  uint32_t card_no,
  uint32_t *p_reg_buff
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

**p_reg_buff*

SD buffer register address pointer

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Properties

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

Description

This function acquires the address of the SD buffer register and stores it in a buffer.

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

Reentrant

This function is reentrant with respect to other channels.

Example

```
uint32_t    reg_buff = 0;

if (R_SDC_SD_GetBuffRegAddress(SDC_SD_CARD0, &reg_buff) != SDC_SD_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, driver open processing must be performed by the R_SDC_SD_Open() function.

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

3.20 R_SDC_SD_1msInterval()

This function increments the SD Card driver's internal timer counter.

Format

```
void R_SDC_SD_1msInterval(
  void
)
```

Parameters

None

Return Values

None

Properties

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

Description

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

Reentrant

This function is not reentrant.

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_SDC_SD_1msInterval();
   }
}

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_sdc_sd_int_wait() and r_sdc_sd_wait() functions in r_sdc_sd_config.c.

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

3.21 R_SDC_SD_SetDmacDtcTransFlg()

This function sets the DMAC/DTC transfer complete flag.

Format

```
sdc_sd_status_t R_SDC_SD_SetDmacDtcTransFlg(
  uint32_t card_no,
  uint32_t flg
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

flg

DMAC/DTC transfer complete flag — SDC_SD_SET_TRANS_STOP

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error (channel error)*

Properties

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

Description

This function sets the DMAC/DTC transfer complete flag.

Table 3.10, 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 SDC_SD_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 SDC_SD_SET_TRANS_STOP in the SDHI SBFAI interrupt handler.

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

Table 3.10 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_SDC_SD_SetDmacDtcTransFlg() 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_SDC_SD_SetDmacDtcTransFlg( ) 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

Reentrant

This function is reentrant with respect to other channels.

Example

<When the DMAC transfer completes successfully>

```
void r_dmaca_callback(void)
{
    volatile dmaca_return_t    ret_dmaca;
    dmaca_stat_t               p_stat_dmaca;

    /* check DMA end */
    /*** DMACA transfer end check ***/
    ret_dmaca = R_DMACA_Control(DMACA_CH0, DMACA_CMD_STATUS_GET,
(dmaca_stat_t*)&p_stat_dmaca);
    if (DMACA_SUCCESS != ret_dmaca)
    {
        return;
    }

    if (true == (p_stat_dmaca.dtif_stat))
    {
        ret_dmaca = R_DMACA_Control(DMACA_CH0, DMACA_CMD_DTIF_STATUS_CLR,
(dmaca_stat_t*)&p_stat_dmaca);
        R_SDC_SD_SetDmacDtcTransFlg(SDC_SD_CARD0, SDC_SD_SET_TRANS_STOP);
    }

    if (true == (p_stat_dmaca.esif_stat))
    {
        ret_dmaca = R_DMACA_Control(DMACA_CH0, DMACA_CMD_ESIF_STATUS_CLR,
(dmaca_stat_t*)&p_stat_dmaca);
    }

    return;
}

<When the DMAC transfer terminates with an error>
#define TEST_BLOCK_CNT  (4)
#define BLOCK_NUM       (512)

sdc_sd_access_t   sdc_sd_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);
sdc_sd_access.p_buff = (uint8_t *)&g_test_r_buff[0];
sdc_sd_access.lbn    = 0x10000000;
sdc_sd_access.cnt    = TEST_BLOCK_CNT;
sdc_sd_access.rw_mode= SDHI_PRE_DEF;

if(R_SDC_SDMEM_Read(SDC_SD_CARD0, &sdc_sd_access) != SDC_SD_SUCCESS)
{
   /* Error */
   R_SDC_SD_SetDmacDtcTransFlg(SDC_SD_CARD0, SDC_SD_SET_TRANS_STOP);
}
```

Special Notes

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

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

3.22 R_SDC_SD_SetLogHdlAddress()

This function sets the LONGQ FIT module handler address.

Format

```
sdc_sd_status_t R_SDC_SD_SetLogHdlAddress(
  uint32_t user_long_que
)
```

Parameters

user_long_que

LONGQ FIT module handler address

Return Values

SDC_SD_SUCCESS	*Successful operation*

Properties

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

Description

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

Reentrant

This function is reentrant with respect to other channels.

Example

```
#define SDC_SD_USER_LONGQ_MAX          (8)
#define SDC_SD_USER_LONGQ_BUFSIZE      (SDC_SD_USER_LONGQ_MAX * 4)
#define SDC_SD_USER_LONGQ_IGN_OVERFLOW (1)

uint32_t             g_sdc_sd_user_longq_buf[SDC_SD_USER_LONGQ_BUFSIZE];
static longq_hdl_t   p_sdc_sd_user_long_que;
longq_err_t          err = LONGQ_SUCCESS;
uint32_t             user_long_que = 0;

err = R_LONGQ_Open(g_sdc_sd_user_longq_buf,
             SDC_SD_USER_LONGQ_BUFSIZE,
             SDC_SD_USER_LONGQ_IGN_OVERFLOW,
             &p_sdc_sd_user_long_que);
if (LONGQ_SUCCESS != err)
{
   /* Error */
}
user_long_que = (uint32_t)p_sdc_sd_user_long_que;
if (R_SDC_SD_SetLogHdlAddress(user_long_que) != SDC_SD_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_SDC_SD_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_SDC_SD_GetErrCode() function.

3.23 R_SDC_SD_Log()

This function acquires an error log.

Format

```
uint32_t R_SDC_SD_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_sdc_sd_rx_if.h.

Description

This function acquires an error log.

To terminate error log acquisition, call this function.

Reentrant

This function is reentrant with respect to other channels.

Example

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

sdc_sd_cfg_t      sdc_sd_config;

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

sdc_sd_config.mode = SDC_SD_CFG_DRIVER_MODE;
sdc_sd_config.voltage = SDC_SD_VOLT_3_3;
if (R_SDC_SD_Initialize(SDC_SD_CARD0, &sdc_sd_config) != SDC_SD_SUCCESS)
{
   /* Error */
   R_SDC_SD_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_SDC_SD_GetErrCode() function.

3.24 R_SDC_SD_GetVersion()

This function acquires the version information for the driver.

Format

```
uint32_t R_SDC_SD_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_sdc_sd_rx_if.h.

Description

This function returns the driver version information.

Reentrant

This function is reentrant.

Example

```
uint32_t version;
version = R_SDC_SD_GetVersion();
```

Special Notes

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

4. Pin Setting

To use the lower-layer SDHI 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[1].

Refer to 4.1, SD Card Insertion and Power-On Timing, and 4.2, SD Card Removal and Power-Off Timing, and create appropriate program code to provide this processing.

[1] For details of the pin setting function, refer to RX Family SDHI Module Using Firmware Integration Technology (R01AN3852EJ).

4.1 SD Card Insertion and Power-On Timing

Figure 4.1 and Table 4.1 show the control procedure. Perform the SD Card insertion procedure after successful operation of the R_SDC_SD_Open() function and when the supply of power voltage to the SD Card is in the halted state and the SDHI output pin is in the L output state.

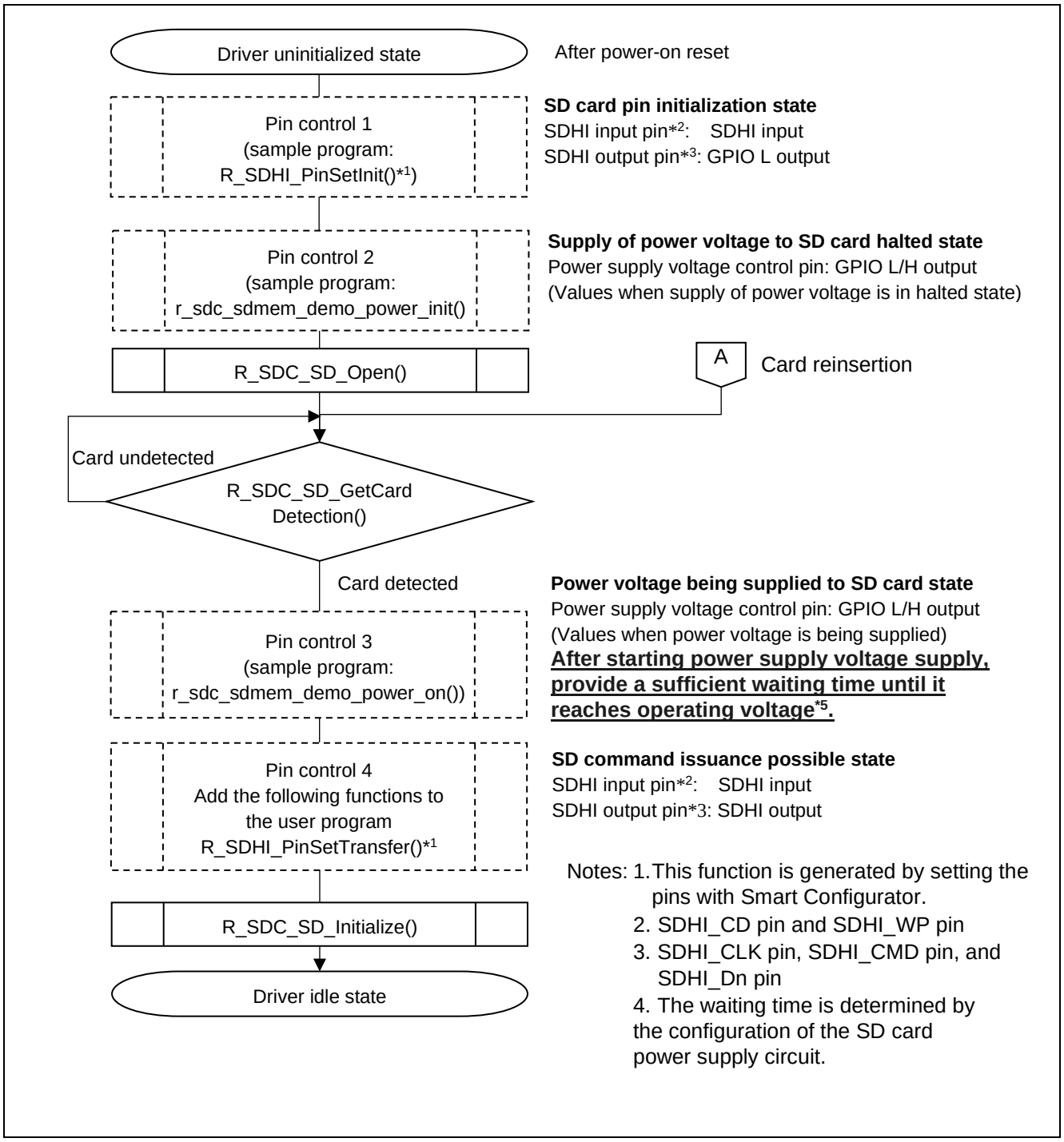


Figure 4.1 SD Card Insertion and Power-On Timing

Table 4.1 User Setting Method at SD Card Insertion

Processing	Target Pin	Pin Settings	Subsequent Pin State
Pin control 1	SDHI input pin*1	PMR setting: General I/O port PCR setting: Input pull-up resistor disabled*3 PDR setting: Input MPC setting: SDHI PMR setting: Peripheral module	SDHI input (SD Card detection possible state)
	SDHI 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	SDHI input pin*1	MPC setting: SDHI PMR setting: Peripheral module	SDHI input
	SDHI output pin*2	MPC setting: SDHI PMR setting: Peripheral module	SDHI output (SD command issuance possible state)

Notes: 1. SDHI_CD pin and SDHI_WP pin
2. SDHI_CLK pin, SDHI_CMD pin, and SDHI_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.

4.2 SD Card Removal and Power-Off Timing

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

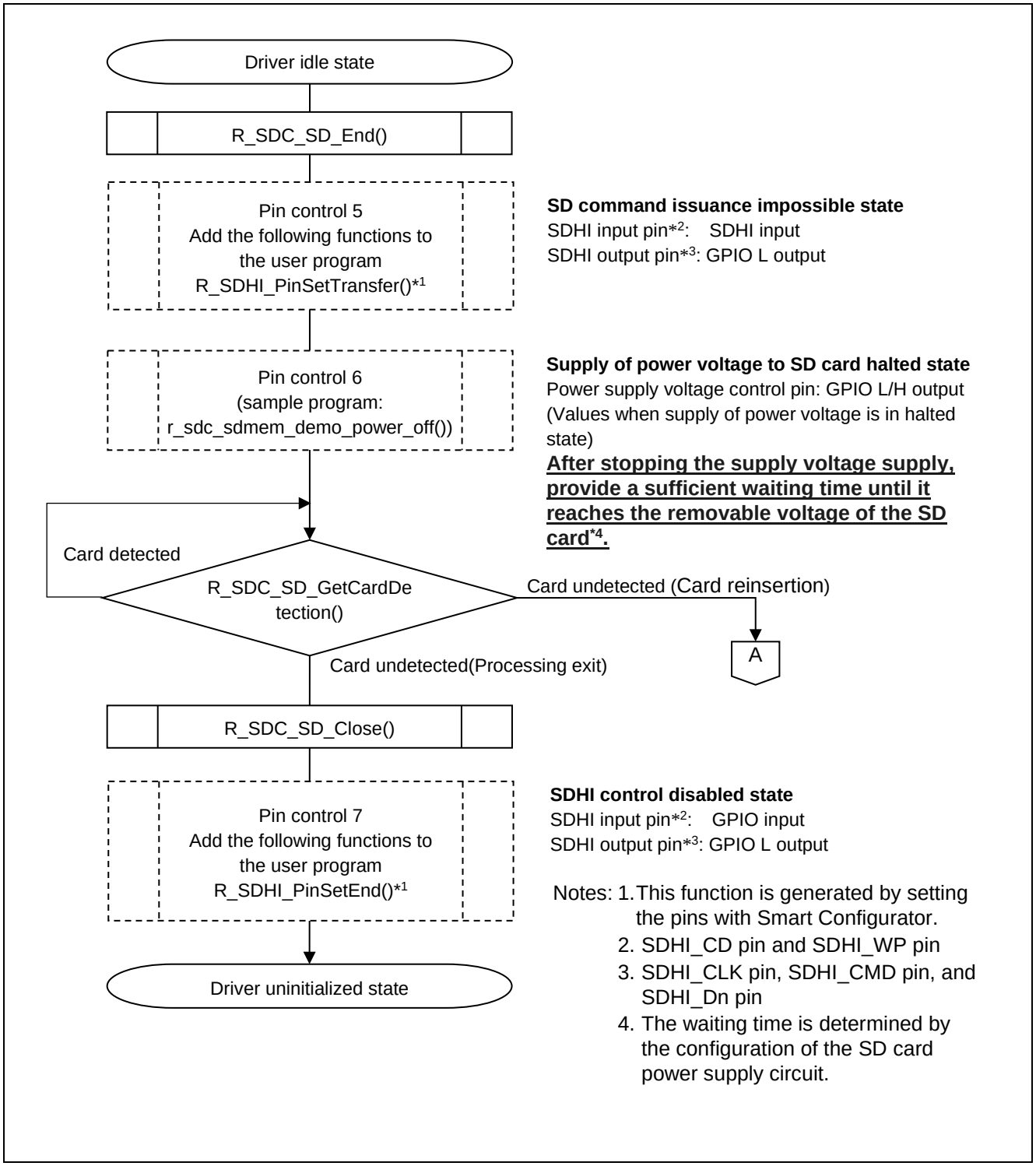


Figure 4.2 SD Card Removal and Power-Off Timing

Table 4.2 User Setting Method at SD Card Removal

Processing	Target Pin	Pin Settings	Subsequent Pin State
Pin control 5	SDHI input pin*1	MPC setting: SDHI PMR setting: Peripheral module	SDHI input
	SDHI 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	SDHI input pin*1	PMR setting: General I/O port MPC setting: Hi-z	GPIO input
	SDHI output pin*2	PMR setting: General I/O port MPC setting: Hi-z	GPIO L output

Notes: 1. SDHI_CD pin and SDHI_WP pin
2. SDHI_CLK pin, SDHI_CMD pin, and SDHI_Dn pin

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 4.1, SD Card Insertion and Power-On Timing and 4.2, SD Card Removal and Power-Off Timing, as well as SD Card read and write processing.

5.2 State Transition Diagram

Figure 5.1 shows the state transition diagram for this driver.

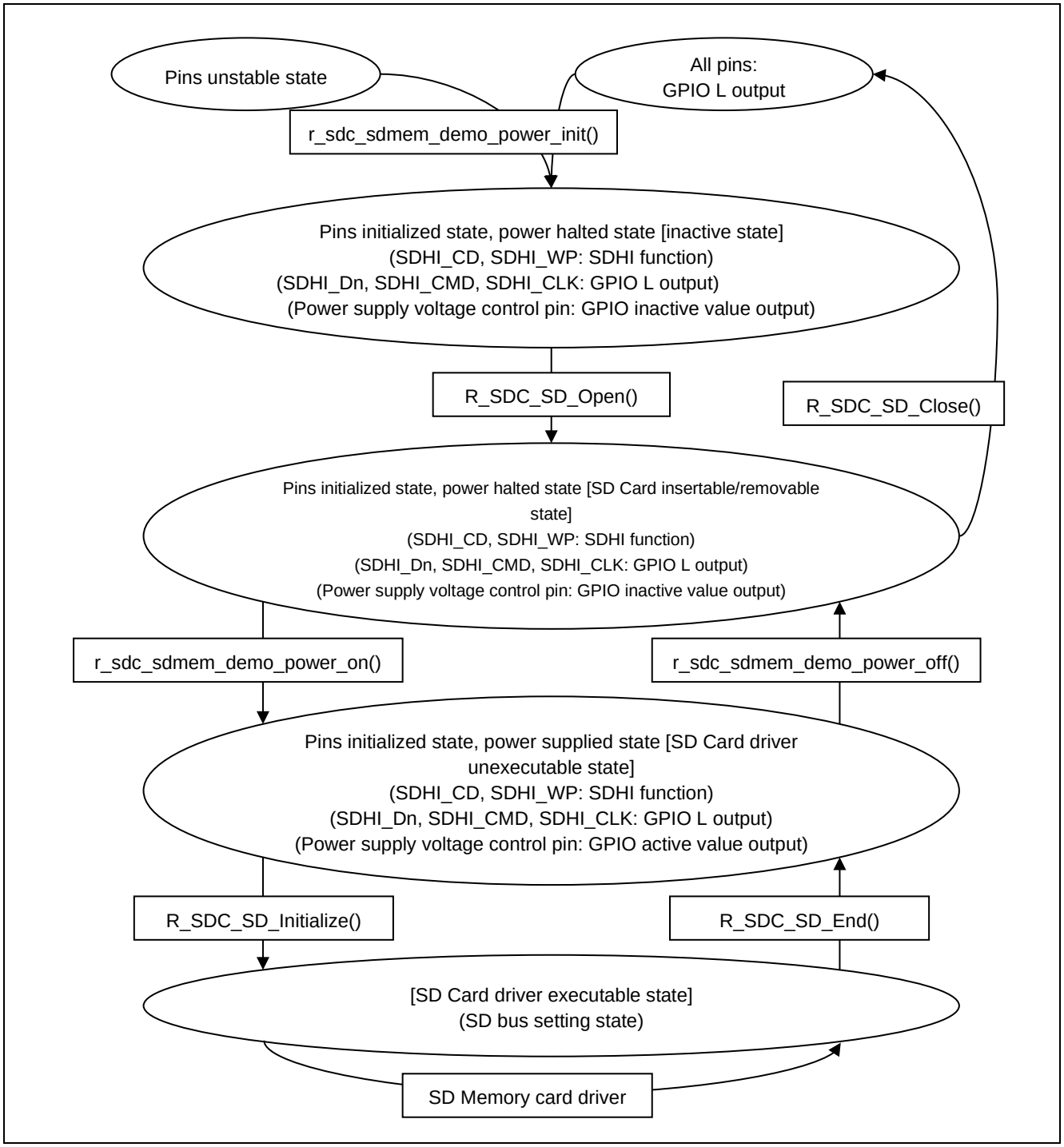


Figure 5.1 State Transition Diagram

5.3 Configuration Overview

The sample program configuration options are set in the file r_sdc_sdmem_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_sdc_sdmem_rx_demo_pin_config.h*	
#define SDC_SD_CFG_POWER_PORT_NONE Note: The default value is "disabled".	This definition is used when an SD Card is used. If SD Card power supply control is not required, enable this definition. If SD Card power supply control is required, disable this definition.
#define SDC_SD_CFG_POWER_HIGH_ACTIVE (1) Note: The default value is "1 (high level supplied)".	This definition is used when an SD Card is used and furthermore SD Card power supply control is required. When set to 1, a high level is supplied to the port that controls the SD Card power supply circuit to enable the SD Card power supply circuit. When set to 0, a low level is supplied to the port that controls the SD Card power supply circuit to enable the SD Card power supply circuit.
#define SDC_SD_CFG_POWER_ON_WAIT (100) Note: The default value is "100 (100 ms wait)".	This definition is used when an SD Card is used. Set this definition to the wait time after power supply is started to the SD 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 SDC_SD_CFG_POWER_OFF_WAIT (100) Note: The default value is "100 (100 ms wait)".	This definition is used when an SD Card is used. Set this definition to the wait time after power supply to the SD Card power supply circuit is stopped until the voltage at which SD 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 R_SDC_SD_CFG_POWER_CARDx_PORT Note: "x" in CARDx indicates an SD Card number (x = 0)	Set these definitions to the port number for the power supply voltage control pin allocated for SD Card number x. Surround each setting value with single quotation marks ' '.
#define R_SDC_SD_CFG_POWER_CARDx_BIT Note: "x" in CARDx indicates an SD Card number (x = 0)	Set these definitions to the bit number for the power supply voltage control pin allocated for SD Card number 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_sdc_sdmem_demo_power_init()	Initializes the power supply voltage control pin settings
r_sdc_sdmem_demo_power_on()	Starts supply of power supply voltage
r_sdc_sdmem_demo_power_off()	Stops supply of power supply voltage
r_sdc_sdmem_demo_softwaredelay()	Performs delay

(1) r_sdc_sdmem_demo_power_init()

This function initializes the SDHI pin settings used by the SD Memory Card driver. It also initializes the setting of the power voltage control pin of the SD Card.

Format

```
sdc_sd_status_t r_sdc_sdmem_demo_power_init(
uint32_t card_no
)
```

Parameters

card_no

SD Card number	The number of the SD Card used (numbering starts at 0)

Return Values

SDC_SD_SUCCESS	*Successful operation*

Description

Initializes the setting of the power voltage control pin of the SD 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_sdc_sdmem_demo_power_on()

This function controls the power supply voltage control pin of the SD Card and starts power supply.

Format

```
sdc_sd_status_t r_sdc_sdmem_demo_power_on(
  uint32_t card_no
)
```

Parameters

card_no

SD Card number	The number of the SD Card used (numbering starts at 0)

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Description

Controls the power supply voltage control pins of the SD Card, and starts the supply of power from the power supply. Then, after the time specified by SDC_SD_CFG_POWER_ON_WAIT in r_sdc_sdmem_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 r_sdc_sdmem_demo_softwaredelay() function in order to wait until the operating voltage is reached. Set the wait time using SDC_SD_CFG_POWER_ON_WAIT in section 6.8.1, Compile Time Settings.

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

(3) r_sdc_sdmem_demo_power_off()

This function controls the power supply voltage control pins of the SD Card, and stops the supply of power from the power supply.

Format

```
sdc_sd_status_t r_sdc_sdmem_demo_power_off(
  uint32_t card_no
)
```

Parameters

card_no

SD Card number	The number of the SD Card used (numbering starts at 0)

Return Values

SDC_SD_SUCCESS	*Successful operation*
SDC_SD_ERR	*General error*

Description

Controls the power supply voltage control pins of the SD Card, and stops the supply of power from the power supply. Then, after the time specified by SDC_SD_CFG_POWER_OFF_WAIT in r_sdc_sdmem_rx_demo_pin_config.h has elapsed, returns the result.

Special Notes:

After stopping the supply of power supply voltage, executes the r_sdc_sdmem_demo_softwaredelay() function in order to wait until the removable voltage is reached. Set the wait time using SDC_SD_CFG_POWER_OFF_WAIT in section 6.8.1, Compile Time Settings.

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

(4) r_sdc_sdmem_demo_softwaredelay()

This function is used when waiting for a particular time.

Format

```
bool r_sdc_sdmem_demo_softwaredelay(
  uint32_t delay,
  sdhi_delay_units_t units
)
```

Parameters

delay

Timeout time (Units: set with the units)

units

Microseconds: SDC_SD_DELAY_MICROSECS
Milliseconds: SDC_SD_DELAY_MILLISECS
Seconds: SDC_SD_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
SD Card power on power supply voltage stabilization time	The wait time until the operating voltage is reached after power supply is started to the SD Card power supply circuit <100 ms> Note:The wait time can be modified with SDC_SD_CFG_POWER_ON_WAIT.
SD Card power off voltage turn-off time	The wait time until the SD Card removable voltage is reached after supply is stopped to the SD Card power supply circuit <100 ms> Note:The wait time can be modified with SDC_SD_CFG_POWER_OFF_WAIT.

5.5 Replacing Wait Time Processing with Operating System Processing

The r_sdc_sdmem_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).

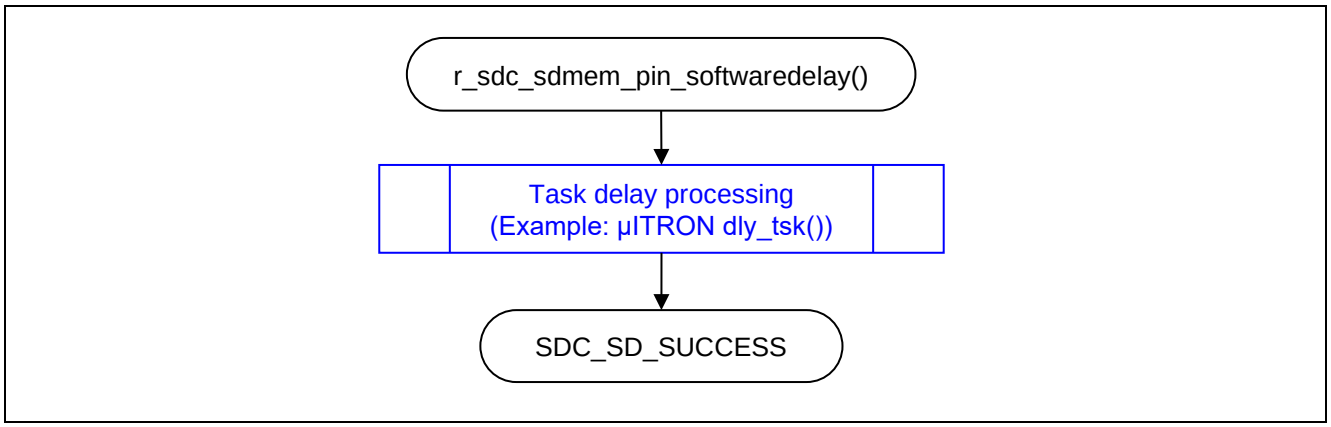


Figure 5.2 Wait Example Using Operating System Task Delay Processing

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

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V6.3.0
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.2.02
Board used	Renesas Starter Kit for RX64M (product No.:R0K50564MSxxxBE) Renesas Starter Kit for RX71M (product No.:R0K50571MSxxxBE) Renesas Starter Kit for RX231 (product No.:R0K505231SxxxBE) Renesas Starter Kit for RX65N (product No.:RTK500565NSxxxxxBE) Renesas Starter Kit for RX65N-2MB (product No.:RTK50565N2SxxxxxBE)

Table 6.2 Confirmed Operation Environment (Rev. 3.00)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.7.0 IAR Embedded Workbench for Renesas RX 4.13.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.02.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.201904 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 4.13.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.3.00
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxx)

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_sdc_sd_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 SD Memory: Notes on Power Consumption Settings (XPS Setting when an ACMD41 Command is Issued) in SDXC Card Default Speed Mode

This SD Memory Card driver issues an ACMD41 command with the command argument XPC set to 0 during SDXC card initialization. As a result, regardless of the VDD power supply Capacity of the external power supply circuit, the SD Memory Card will operate with a maximum power consumption of 0.36 W (maximum current drain 100 mA, 3.6 V) in Default Speed mode for SDXC cards as well as SDSC and SDHC cards.

6.4 Replacing Wait Processing with Operating System Processing

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

Table 6.2 Target Microcontroller Interface Functions

Function	Functional Overview
r_sdc_sd_int_wait()	Status interrupt wait processing
r_sdc_sd_int_mem_wait()	Status interrupt wait processing (SD memory control)
r_sdc_sd_int_err_mem_wait()	Status interrupt wait processing (SD memory error control)
r_sdc_sd_wait()	Wait time processing

(1) r_sdc_sd_int_wait()*

This function is used when waiting for a status interrupt.

Format

```
sdc_sd_status_t r_sdc_sd_int_wait(
  uint32_t card_no,
  int32_t time
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

time

Timeout time (units ms)

Return Values

SDC_SD_SUCCESS	*Successful operation (Interrupt request generation)*
SDC_SD_ERR	*General error*

Description

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

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

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

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

This function calls SD status register 1 and 2 acquisition processing (the r_sdc_sd_get_intstatus() function) internally to determine whether or not an interrupt has occurred.

Note: * The r_sdc_sd_int_mem_wait() and r_sdc_sd_int_err_mem_wait() functions operate similarly.

Special Notes

The response reception wait time during communication with the SD Card 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_sdc_sd_int_wait() function themselves.

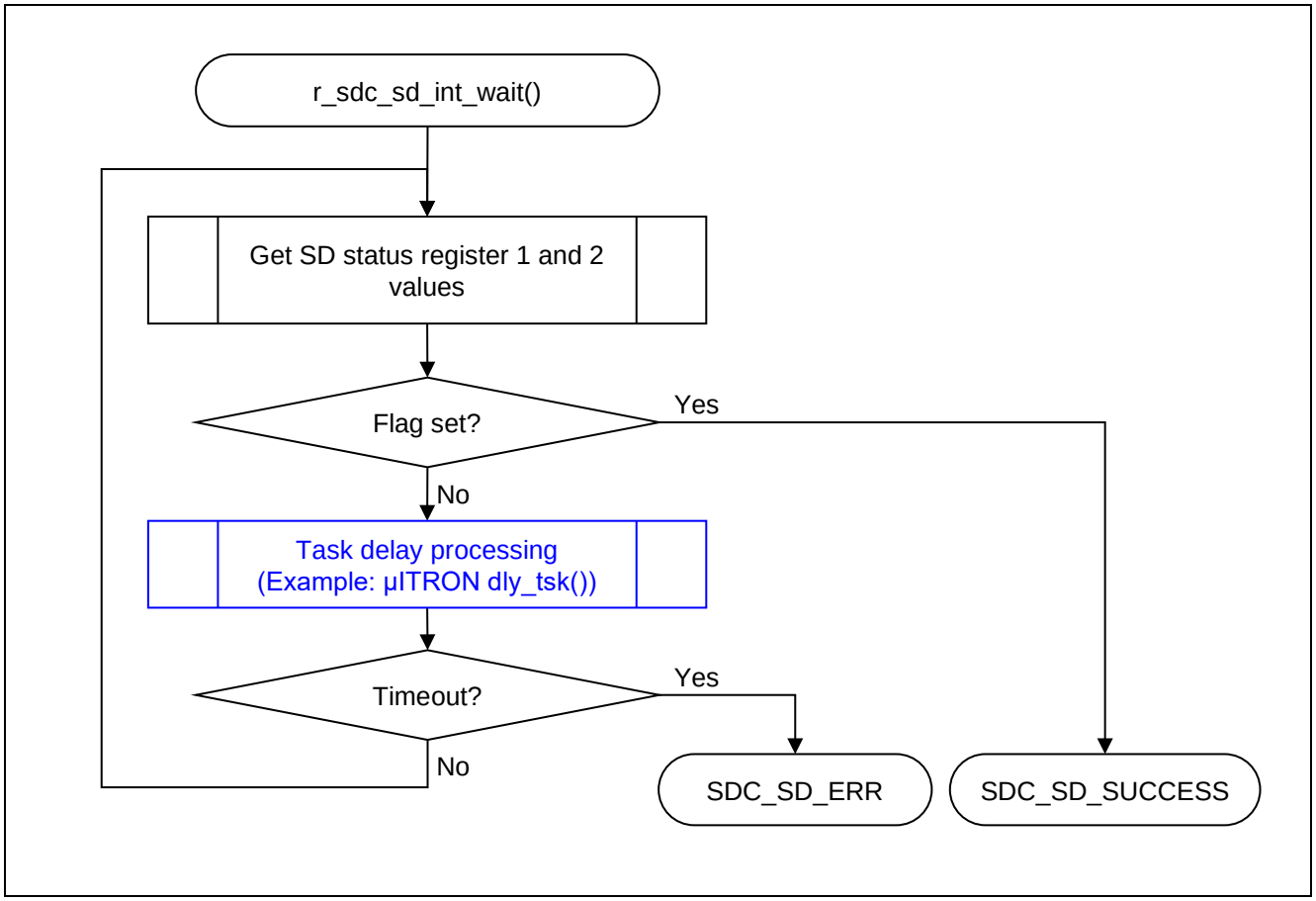


Figure 6.1 SD Protocol Status Verification Example Using Operating System Task Delay Processing

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_sdc_sd_int_wait() function SD status register 1 and 2 acquisition processing (the r_sdc_sd_get_intstatus() function) with event flag set wait processing and furthermore add wakeup processing to the SD protocol status interrupt callback function.

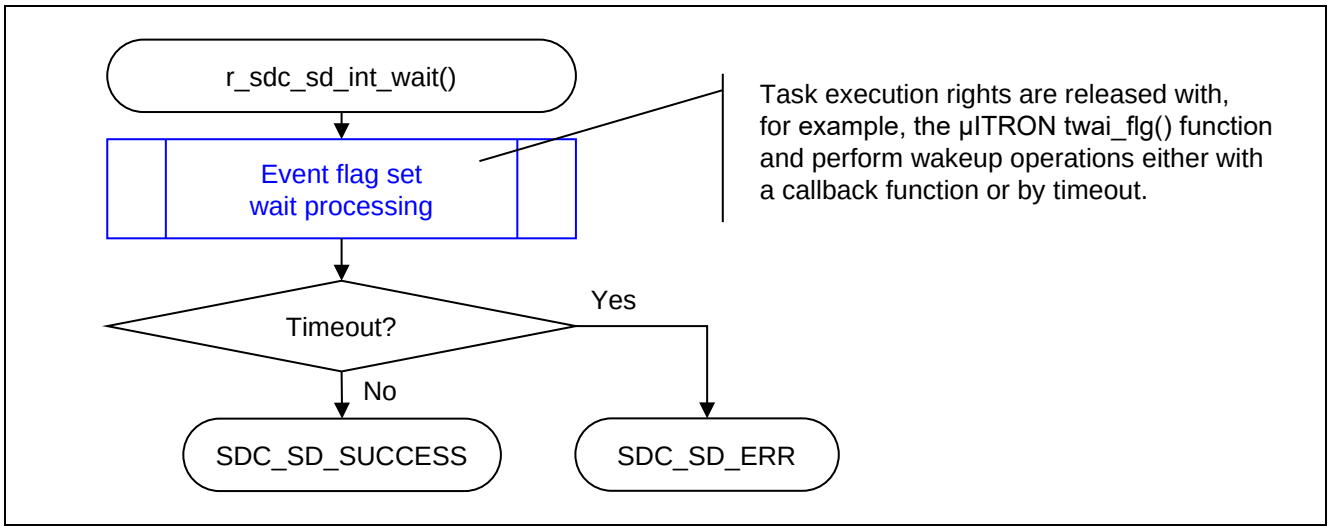


Figure 6.2 SD Protocol Status Verification Example Using Operating System Wait Task Processing

(2) r_sdc_sd_wait()

This function is used when waiting for a particular time.

Format

```
sdc_sd_status_t r_sdc_sd_wait(
  uint32_t card_no,
  int32_t time
)
```

Parameters

card_no

SD Card number — The number of the SD Card used (numbering starts at 0)

time

Timeout time (units ms)

Return Values

SDC_SD_SUCCESS	*Successful operation (Interrupt request generation)*
SDC_SD_ERR	*General error*

Description

This function performs wait time processing.

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

Special Notes

Table 6.5 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 6.3 Wait Time Processing not Associated with Status Verification

Type	Description
The 74 clock cycle wait during SD Card initialization	Card identification mode: A wait of 74 clock cycles <3 ms> for initialization (A minimum of 2 ms and a maximum of 3 ms must be assured)
Ready state transition detection during SD Card initialization	Card identification mode: Wait for the ready state <5 ms> (maximum: 1 second) For SD memory, an ACMD41 command is issued at 5 ms intervals, repeated a maximum of 200 times.
Wait time after a CMD9 command has been issued to the SD memory	Data transfer mode: response wait time <3 ms> after a CMD9 command has been issued to the SD memory Wait time after issuing a CMD9 command to the SD memory (A minimum of 2 ms and a maximum of 3 ms must be assured)

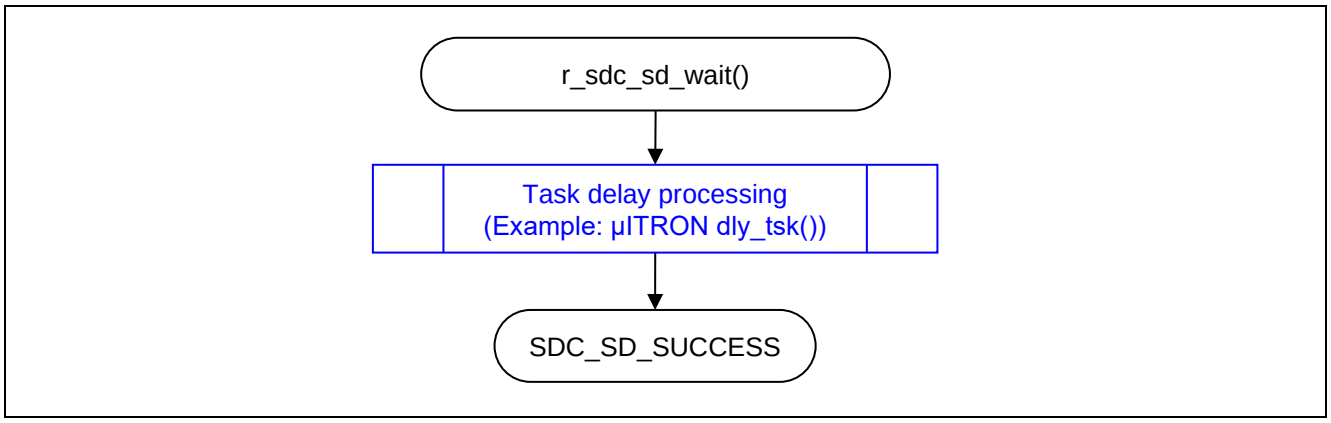


Figure 6.3 Wait Example Using Operating System Task Delay Processing

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

This module reflects the content of the following technical updates.

- TN-RX*-A195A/E
- TN-RX*-A196A/E
- TN-RX*-A197A/E

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

Revision History

Rev.	Date	Description	
		Page	Summary
2.01	Feb 28, 2018	-	First edition issued. SD Mode SD Memory Card Driver Software RTM0RX0000DSDD0 Ver.2.00 User's Manual (R01UW0135) changed to the application note.
2.02	Jun 29, 2018	-	Corresponded to SD Specifications Part1 Physical Layer Simplified Specification.
3.00	Feb. 10. 2020	-	Supported the following compilers. - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX Fix type mismatch with "r_sdhi_rx". Add "WAIT_LOOP" to the code.

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

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. 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/.

© 2020 Renesas Electronics Corporation. All rights reserved.