RX Family

SDHI Module Using Firmware Integration Technology

Summary

This application note describes an SDHI module using Firmware Integration Technology (FIT). This module is a device driver that controls the on-chip SD host interface (SDHI) of RX Family microcontrollers from Renesas Electronics. The module is referred to below as the SDHI FIT module.

Target Devices

RX231 Group, RX23W Group

RX64M Group, RX65N Group, RX651 Group, RX66N Group and RX671 Group

RX71M Group, RX72M Group and RX72N Group

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 Operation Confirmation Environment".

Related Documents

RX Family Board Support Package Module Using Firmware Integration Technology (R01AN1685)

RX Family SD Mode SD Memory Card Driver Firmware Integration Technology(R01AN4233)

RX Family DMA Controller DMACA Control Module Firmware Integration Technology (R01AN2063)

RX Family DTC Module Using Firmware Integration Technology (R01AN1819)

RX Family CMT Module Using Firmware Integration Technology (R01AN1856)

RX Family LONGQ Module Using Firmware Integration Technology (R01AN1889)

Contents

1. Overview

1.1 SDHI FIT Module

The SDHI FIT module can be added to projects as an API. Refer to 2.12, Adding the FIT Module to Your Project, for instructions for adding the SDHI FIT module to your project.

1.2 Overview of SDHI FIT Module

By using the API functions provided by the SDHI FIT module in combination with your own software, you can control an SD memory card or SDIO in SD mode.

Note that SD memory card and SDIO drivers designed to work in combination with the SDHI FIT module are available separately. To obtain these, visit the following webpage:

SD card drivers for RX Family microcontrollers: https://www.renesas.com/driver/rtm0rx0000dsdd

1.2.1 Overview of Functions

The functions of the SDHI FIT module are listed below.

Table 1.1 List of Functions

Item	Function
Clock supply	Supports SDHI clock supply/halt setting.
SD bus	Supports SD mode (1-bit/4-bit) setting.
Interrupt control	Supports settings to enable/disable interrupts used by the SDHI. Supports clearing of interrupt flags used by the SDHI.
Callback functions	Supports calling a designated callback function when one of the following interrupts occurs: • Card access interrupt (CACI) • SDIO access interrupt (SDACI) • Card detect interrupt (CDETI) • SD buffer access interrupt (SBFAI)
SDHI register set/get contents	Supports SDHI register set/get contents.
Endianness	Supports operation in little-endian or big-endian mode.
Other functions	Supports Firmware Integration Technology (FIT).

1.3 API Overview

Table 1.2 lists the API functions provided by the SDHI FIT module.

Table 1.2 API Functions

Function	Function Description
R_SDHI_Open	Module open processing
R_SDHI_Close	Module close processing
R_SDHI_IntHandler0	Interrupt handler
R_SDHI_IntCallback	Callback function registration processing for card access interrupt (CACI) and card detect interrupt (CDETI)
R_SDHI_IntSDBuffCallBack	Callback function registration processing for SD buffer access interrupt (SBFAI)
R_SDHI_IntSdioCallback	Callback function registration processing for SDIO access interrupt (SDACI)
R_SDHI_EnableIcuInt	ICU controller interrupt enable processing for SDHI
R_SDHI_DisableIcuInt	ICU controller interrupt disable processing for SDHI
R_SDHI_SetIntMask	SD interrupt enable processing
R_SDHI_ClearIntMask	SD interrupt disable processing
R_SDHI_ClearSdstsReg	SD status register clear processing
R_SDHI_SetSdioIntMask	SDIO interrupt mask register setting processing
R_SDHI_ClearSdioIntMask	SDIO interrupt mask register clear processing
R_SDHI_ClearSdiostsReg	SDIO status register clear processing
R_SDHI_SetClock	SD clock supply/halt processing
R_SDHI_SetBus	SD bus setting processing
R_SDHI_GetResp	Command response acquisition processing
R_SDHI_OutReg	SDHI register setting processing
R_SDHI_InReg	SDHI register get contents processing
R_SDHI_CDLayout	SDHI_CD pin in use/not in use confirmation processing
R_SDHI_WPLayout	SDHI_WP pin in use/not in use confirmation processing
R_SDHI_GetWP	SDHI_WP pin state acquisition processing
R_SDHI_GetSpeedType	Speed mode acquisition processing
R_SDHI_GetBuffRegAddress	SD buffer register address acquisition processing
R_SDHI_GetVersion	Module version information acquisition processing
R_SDHI_SetLogHdlAddress	LONGQ module handler address setting processing*1
R_SDHI_Log	Error log acquisition processing*1

Note 1. Separate LONGQ FIT module required.

1.4 Processing Example

1.4.1 Application Structure

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

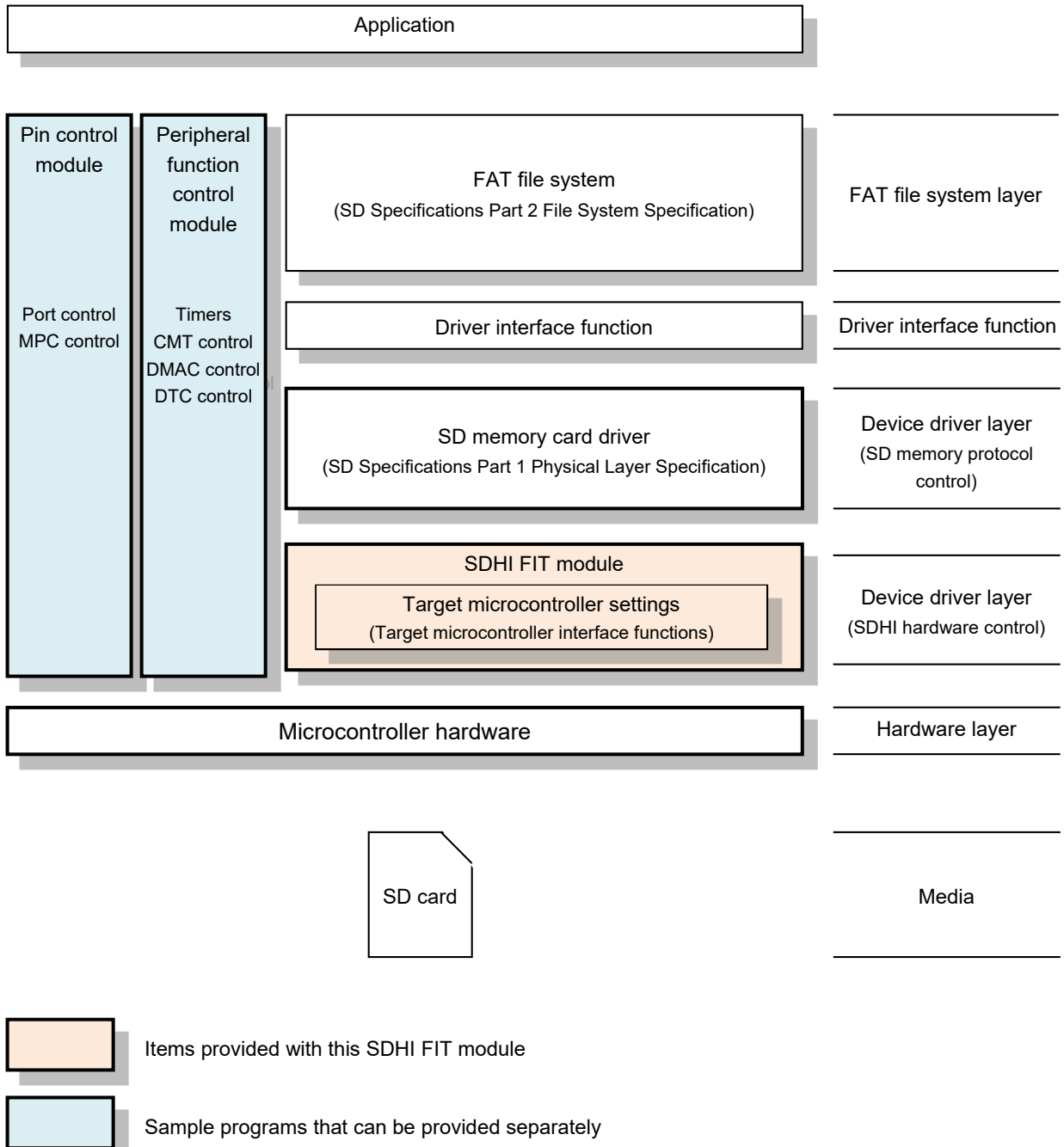


Figure 1.1 Application Structure

(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/software-tool/fat-file-system-m3s-tfat-tiny-rx-family

(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 webpage 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 Specification SD memory protocol control and SDHI low-level access control.

(4) SDHI FIT Module

The module described in this application note. It also includes target microcontroller interface functions and interrupt setting files compatible with specific microcontroller products.

(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 Refer to “Related Documents” on the first page of this application note, and obtain the relevant items.

(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.

2. API Information

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

2.1 Hardware Requirements

The microcontroller used must support the following function.

- SDHI

2.2 Software Requirements

The SDHI FIT module is dependent on the following FIT module:

- Board support package (r_bsp) Rev.5.20 or higher

2.3 Supported Tool Chain

The SDHI FIT module has been confirmed to operate with the toolchain listed in 6.1, Operation Confirmation Environment.

2.4 Interrupt Vectors

If the macro definition SDHI_CFG_MODE_INT is set to SDHI_MODE_HWINT, SDHI interrupts are enabled. Table 2.1 lists the interrupt vectors used by the SDHI FIT module.

Table 2.1 Interrupt Vectors

Device	Interrupt Vectors
RX64M RX65N RX66N RX671 RX71M RX72M RX72N	SD buffer access interrupt (SBFAI) (vector number: 44) GROUPBL1 interrupt (vector number: 111) • Card detect interrupt (CDETI) (group interrupt source number: 3) • Card access interrupt (CACI) (group interrupt source number: 4) • SDIO access interrupt (SDACI) (group interrupt source number: 5)
RX231 RX23W	SD buffer access interrupt (SBFAI) (vector number: 40) • Card detect interrupt (CDETI) (vector number: 41) • Card access interrupt (CACI) (vector number: 42) • SDIO access interrupt (SDACI) (vector number: 43)

2.5 Header Files

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

2.6 Integer Types

The SDHI FIT module is coded in ANSI C99. These types are defined in stdint.h.

2.7 Compile Time Settings

The configuration option settings of the SDHI FIT module are contained in r_sdhi_rx_config.h.

The following table lists the names of the options and descriptions of their setting values.

Configuration options in *r_sdhi_rx_config.h*	
#define SDHI_CFG_CHx_INCLUDED (1) Note: Channel 0 default value: “1 (Enabled)” Note: “x”: Channel number	Select whether the corresponding channel will be used. If “disabled” is selected, the code for processing the corresponding channel is omitted. If “enabled” is selected, the code for processing the corresponding channel is included. If the microcontroller supports multiple channels, it is necessary to add definitions for several channels.
#define SDHI_CFG_CHx_CD_ACTIVE (1) Note: Default value: “1 (Enabled)” Note: “x”: Channel number	If it is not necessary to assign an SDHI_CD pin, it can be removed individually from control by the SDHI FIT module. To assign a pin to the SDHI function so that it can be controlled, set this option to (1). To remove it from the control of the SDHI FIT module, set this option to (0). When a pin is removed from the control of the SDHI FIT module it can be used for a different purpose (as a general I/O port, for example). When functional control by the SDHI FIT module is removed, the functionality for specifying the pin to be used is also disabled. Therefore, to use a pin for a different purpose, it is also necessary to specify the pin to be used separately. Settings are required for each channel used.
#define SDHI_CFG_CHx_WP_ACTIVE (1) Note: Default value: “1 (Enabled)” Note: “x”: Channel number	If it is not necessary to assign an SDHI_WP pin, it can be removed individually from control by the SDHI FIT module. To assign a pin to the SDHI function so that it can be controlled, set this option to (1). To remove it from the control of the SDHI FIT module, set this option to (0). When a pin is removed from the control of the SDHI FIT module it can be used for a different purpose (as a general I/O port, for example). When functional control by the SDHI FIT module is removed, the functionality for specifying the pin to be used is also disabled. Therefore, to use a pin for a different purpose, it is also necessary to specify the pin to be used separately. Settings are required for each channel used.
#define SDHI_CFG_CHx_INT_LEVEL (10) /* SDHI channel x interrupt level */ Note: Default value: “(10)”	Specify the level of the card detect interrupt (CDETI), card access interrupt (CACI), and SDIO access interrupt (SDACI).
#define SDHI_CFG_CHx_INT_LEVEL_DMADTC (10) /* SDHI channel x DMA/DTC interrupt level */ Note: Default value: “(10)”	Specify the level of the SD buffer access interrupt (SBFAI). This interrupt level is relevant when writing data to the SD buffer, or when reading data from the SDHI buffer, using the DMAC or DTC.
#define SDHI_CFG_DIV_HIGH_SPEED SDHI_DIV_2 Note: Default value: “SDHI_DIV_2” (×1/2)*1	The clock frequency definition for high-speed mode. Specify the PCLKB division ratio by setting the SDHI clock frequency select bits (CLKSEL[7:0]). Refer to the hardware manual of the microcontroller regarding setting values, and specify a value of SDHI_DIV_1 to SDHI_DIV_512 (×1 to ×1/512). For example, for PCLKB = 60 MHz, high-speed mode clock frequency = 30 MHz, the setting value would be SDHI_DIV_2 (×1/2).

#define SDHI_CFG_DIV_DEFAULT_SPEED SDHI_DIV_4 Note: Default value: “SDHI_DIV_4” (×1/4)*1	The clock frequency definition for default-speed mode. The setting method is the same as that for high-speed mode, above. For example, for PCLKB = 60 MHz, default-speed mode clock frequency = 15 MHz, the setting value would be SDHI_DIV_4 (×1/4).
#define SDHI_CFG_DIV_INIT_SPEED SDHI_DIV_1024 Note: Default value: “SDHI_DIV_256” (×1/256)*1	The clock frequency definition for card-recognition mode. The setting method is the same as that for high-speed mode, above. For example, for PCLKB = 60 MHz, card-recognition mode clock frequency = 234 kHz, the setting value would be SDHI_DIV_256 (×1/256).
#define SDHI_CFG_SDOPT_CTOP (0x000eul) /* CD time out count */ Note: Default value: “0x000eul”	The timeout card detect timeout setting. Set the timeout value in the card detect timeout counter (CTOP) bits (3 to 0) in the card access option register (SDOPT).
#define SDHI_CFG_SDOPT_TOP (0x00e0ul) /* response time out count */ Note: Default value: “0x00e0ul”	The command response timeout setting. Set the SRBSYTO[3:0] timeout value in the timeout counter (TOP) bits (7 to 4) in the card access option register (SDOPT).
#define SDHI_CFG_PARAM_CHECKING_ENABLE (1) Note: Default value: “1 (Enabled)”	Set argument check to enable or disable. (0): Disable (1): Enable
/* #define SDHI_CFG_LONGQ_ENABLE */ Note: Default value: “Disabled”	Set this definition if the error log acquisition function using the LONGQ FIT module will be used. To use this function, it is necessary to use the debugging module (a dedicated module with this definition enabled) and to add the LONGQ FIT module.

Note 1. SDHI_DIV_n (n represents an integer-value division ratio) indicates the PCLK division ratio of the SDHI. Depending on the electrical characteristics of the microcontroller, it may not be possible to use the maximum transfer frequency defined in SD Specifications, Part 1, Physical Layer Specification. Refer to the hardware manual of the microcontroller to determine the maximum allowable transfer frequency setting.

2.8 Code Sizes

The table below lists code sizes of the SDHI FIT module. As representative examples, one device each is listed from the RX200 Series, RX600 Series, and RX700 Series.

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

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

Module Revision: r_sdhi_rx rev2.07

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00

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

GCC for Renesas RX 8.03.00.202002

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

IAR C/C++ Compiler for Renesas RX version 4.14.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

Table 2.2 Code Sizes

ROM, RAM and Stack Code Sizes (note 1, 2)							
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	1,793 bytes	1,508 bytes	4,036 bytes	3,388 bytes	3,048 bytes	2,620 bytes
	RAM	28 bytes		28 bytes		32 bytes	
	Max. size of user stack	68 bytes		-		60 bytes	
	Max. size of interrupt stack	48 bytes		-		64 bytes	
RX65N	ROM	1,867 bytes	1,582 bytes	4,132 bytes	3,468 bytes	3,096 bytes	2,695 bytes
	RAM	28 bytes		28 bytes		32 bytes	
	Max. size of user stack	88 bytes		-		68 bytes	
	Max. size of interrupt stack	36 bytes		-		68 bytes	
RX71M	ROM	1,864 bytes	1,579 bytes	4,132 bytes	3,468 bytes	3,104 bytes	2,680 bytes
	RAM	28 bytes		28 bytes		32 bytes	
	Max. size of user stack	88 bytes		-		68 bytes	
	Max. size of interrupt stack	36 bytes		-		68 bytes	

Note 1. The necessary memory size differs depending on factors such as the version and compile options of the C compiler.
Note 2. The values shown apply to little-endian bit order. Depending on the endianness, the above memory sizes may differ.

2.9 Arguments

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

2.10 Return Values

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

2.11 Callback Functions

The SDHI FIT module calls the callback function specified by the user when the Card detect interrupt (CDETI), Card access interrupt (CACI), or SDIO access interrupt (SDACI) occurs.

For the method of registering callback functions, refer to R_SDHI_IntCallback(), R_SDHI_IntSDBuffCallback(), and R_SDHI_IntSdioCallback().

2.12 Adding the FIT Module to Your Project

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

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

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

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

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

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

2.13 “for”, “while” and “do while” 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

R_SDHI_Open()

This function is run first when utilizing the API functions provided by the SDHI FIT module.

Format

```
sdhi_status_t R_SDHI_Open(
  uint32_t channel,
)
```

Parameters

channel

Channel number	SDHI channel number to be used (starting from 0)

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Obtains the SDHI channel resource specified by the argument channel, and initializes the SDHI FIT module and SDHI channel. Also, takes exclusive possession of the SDHI channel resource.

Example

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

if (R SDHI Open(SDHI CH0) != SDHI SUCCESS)
{
   /* Error */
}
```

Special Notes

The value after initialization of the swap control register (SDSWAP) depends on the endian setting.

Little endian: 0x00000000 (Swap write/read: Disable)
Big endian: 0x000000c0 (Swap write/read: Enable)

Pin settings must be entered before running this function. Refer to 4, Pin Settings.

If this function does not complete successfully, library functions other than R_SDHI_GetVersion(), R_SDHI_Log(), and R_SDHI_SetLogHdlAddress() cannot be used.

The pin states remain unchanged before and after this function is run.

R_SDHI_Close()

This function releases the resource currently in use by the SDHI FIT module.

Format

```
sdhi_status_t R_SDHI_Close(
  uint32_t channel
)
```

Parameters

channel

Channel number	SDHI channel number to be used (starting from 0)

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Ends all processing by the SDHI FIT module, and releases the SDHI channel resource specified by the argument channel.

The SDHI channel is then set to the module stop state.

Insertion-extraction interrupts are disabled after this function runs.

Example

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

if (R_SDHI_Close(SDHI_CH0) != SDHI_SUCCESS)
{
   /* Error */
}
```

Special Notes

Pin settings must be entered before running this function. Refer to 4, Pin Settings. Before running this function, initialization processing by the R_SDHI_Open() function is required.

The pin states remain unchanged before and after this function is run.

R_SDHI_IntHandler0()

This function is the interrupt handler.

Format

```
void R_SDHI_IntHandler0(
  void *vect
)
```

Parameters

**vect*

vector table

Return Values

None

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

This is the interrupt handler of the SDHI FIT module.

It is incorporated into the system as a processing routine for interrupt sources supported by the SDHI.

When a callback function for the card access interrupt (CACI) and card detect interrupt (CDETI), and a callback function for the SDIO access interrupt (SDACI) have been registered, the appropriate callback function is called by this function.

Example

No settings are required because the function is incorporated into the system.

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_IntCallback()

This function registers the callback function for the card access interrupt (CACI) and card detect interrupt (CDETI).

Format

```
sdhi_status_t R_SDHI_IntCallback(
  uint32_t channel,
  sdhi_status_t (*callback)(uint32_t, uint32_t)
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

*(*callback)(uint32_t, uint32_t): Callback function to be registered*

If set to a null pointer, no callback function is registered.
The first argument (uint32_t) contains the value of SD status register 1 (SDSTS1).
The second argument (uint32_t) contains the value of SD status register 2 (SDSTS2).

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Registers the callback function for the card access interrupt (CACI) and card detect interrupt (CDETI). The callback function registered by this function is called as a subroutine of the interrupt handler when an interrupt is generated by a change in the SD protocol status.

Example

```
/* Callback function */
sdhi_status_t r_sdhi_callback(uint32_t sdsts1, uint32_t sdsts2)
{
   /* Do nothing */

   return SDHI_SUCCESS;
}

if (R_SDHI_IntCallback(SDHI_CH0, r_sdhi_callback) != SDHI_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

The callback function registered by this function is different from the callback functions of the SD buffer access interrupt (SBFAI) and SDIO access interrupt (SDACI). Therefore, this callback function is not called when the above-mentioned interrupts occur.

R_SDHI_IntSDBuffCallback()

This function registers the callback function for the SD buffer access interrupt (SBFAI).

Format

```
sdhi_status_t R_SDHI_IntSDBuffCallback(
  uint32_t channel,
  sdhi_status_t (*callback)(void *)
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

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

If set to a null pointer, no callback function is registered.
The value of (void *) is always 0.

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Registers the callback function for the SD buffer access interrupt (SBFAI). The callback function registered by this function is called as a subroutine of the DTC's data transfer end interrupt handler when an interrupt is generated at DTC transfer end.

Example

```
/* Callback function */
sdhi_status_t r_sdhi_sdbuff_callback(void * vect)
{
   /* Do nothing */

   return SDHI_SUCCESS;
}

if (R_SDHI_IntSDBuffCallback(SDHI_CH0, r_sdhi_sdbuff_callback) !=
SDHI_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

The callback function registered by this function is different from the callback function of the card access interrupt (CACI) and card detect interrupt (CDETI), and from the callback function of the SDIO access interrupt (SDACI). Therefore, this callback function is not called when the above-mentioned interrupts occur.

R_SDHI_IntSdioCallback()

This function registers the callback function for the SDIO access interrupt (SDACI).

Format

```
sdhi_status_t R_SDHI_IntSdioCallback(
  uint32_t channel,
  sdhi_status_t (*callback)(uint32_t)
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

*(*callback)(uint32_t): Callback function to be registered*

If set to a null pointer, no callback function is registered.

The first argument (uint32_t) contains the value of the SDIO status register (SDIOSTS).

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Registers the callback function for the SDIO access interrupt (SDACI).

The callback function registered by this function is called as a subroutine of the interrupt handler when an SDHI SDIO interrupt is generated.

Example

```
/* Callback function */
sdhi_status_t r_sdhi_sdio_callback(int32_t sdiosts)
{
   /* Do nothing */

   return SDHI_SUCCESS;
}

if (R_SDHI_IntSdioCallback(SDHI_CH0, r_sdhi_sdio_callback) != SDHI_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

The callback function registered by this function is different from the callback function of the card access interrupt (CACI) and card detect interrupt (CDETI), and from the callback function of the SD buffer access interrupt (SBFAI). Therefore, this callback function is not called when the above-mentioned interrupts occur.

R_SDHI_EnableIcuInt()

Enables ICU controller interrupts*1 for the SDHI.

Note 1. The following interrupt is enabled. [SD buffer access interrupt (SBFAI), Card detect interrupt (CDETI), Card access interrupt (CACI), SDIO access interrupt (SDACI)]

Format

```
sdhi_status_t R_SDHI_EnableIcuInt(
  uint32_t channel,
  uint32_t select
)
```

Parameters

channel

Channel number SDHI channel number to be used (starting from 0)

select

Specify interrupt arguments using values for the macro definitions shown in the table below or OR operations.

Macro Definition	Value (Bit)	Processing
SDHI_HWINT_ACCESS_CD	0x0001	Card detect interrupt (CDETI) setting Card access interrupt (CACI) setting SDIO access interrupt (SDACI) setting
SDHI_HWINT_BUFFER	0x0010	SD buffer access interrupt (SBFAI) setting

Return Values

SDHI_SUCCESS *Successful operation*
SDHI_ERR *General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Makes settings to the ICU controller registers.

Makes settings to the SDHI's interrupt source property register (IPR). The setting values are defined by #define SDHI_CHx_INT_LEVRL and #define SDHI_CFG_CHx_INT_LEVEL_DMADTC.

Sets the SDHI interrupt request enable register (IEN) to enable interrupts.

Example

```
/* Enable all SDHI ICU interrupt */
R_SDHI_EnableIcuInt(SDHI_CH0, SDHI_HWINT_ACCESS_CD | SDHI_HWINT_BUFFER);

/* Enable only SD buffer access ICU interrupt */
R_SDHI_EnableIcuInt(SDHI_CH0, SDHI_HWINT_BUFFER);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_DisableIcuInt()

Disables ICU controller interrupts*1 for the SDHI.

Note 1. The following interrupt is disabled. [SD buffer access interrupt (SBFAI), Card detect interrupt (CDETI), Card access interrupt (CACI), SDIO access interrupt (SDACI)]

Format

```
sdhi_status_t R_SDHI_DisableIcuInt(
  uint32_t channel,
  uint32_t select
)
```

Parameters

channel

Channel number SDHI channel number to be used (starting from 0)

select

Specify interrupt arguments using values for the macro definitions shown in the table below or OR operations.

Macro Definition	Value (Bit)	Processing
SDHI_HWINT_ACCESS_CD	0x00000001	Card detect interrupt (CDETI) setting Card access interrupt (CACI) setting SDIO access interrupt (SDACI) setting
SDHI_HWINT_BUFFER	0x00000010	SD buffer access interrupt (SBFAI) setting

Return Values

SDHI_SUCCESS *Successful operation*
SDHI_ERR *General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Makes settings to the ICU controller registers.

Clears the SDHI interrupt source priority register (IPR) to 0.

Sets the SDHI interrupt request enable register (IEN) to disable interrupts.

Example

```
/* Disable all SDHI ICU interrupt */
R_SDHI_DisableIcuInt(SDHI_CH0, SDHI_HWINT_ACCESS_CD | SDHI_HWINT_BUFFER);

/* Disable only SD buffer access ICU interrupt */
R_SDHI_DisableIcuInt(SDHI_CH0, SDHI_HWINT_BUFFER);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_SetIntMask()

This function controls the SD interrupt mask registers to enable SD interrupts.

Format

```
sdhi_status_t R_SDHI_SetIntMask(
  uint32_t channel,
  uint32_t mask1,
  uint32_t mask2
)
```

Parameters

channel

Channel number SDHI channel number to be used (starting from 0)

mask1

SD interrupt mask register 1 (SDIMSK1) control

To enable an interrupt, set the target bit to 1.

To not change the interrupt setting, clear the target bit to 0.

However, setting to the Read Only bit is invalid.

Bit	Symbol	Bit Name	R/W
b0	RSPENDM	Response End Interrupt Request Mask	R/W
b1	-	Reserved	R
b2	ACENDM	Access End Interrupt Request Mask	R/W
b3	SDCDRMM	SDHI_CD Removal Interrupt Request Mask	R/W
b4	SDCDRMM	SDHI_CD Insertion Interrupt Request Mask	R/W
b7-b5	-	Reserved	R
b8	SDD3RMM	SDHI_D3 Removal Interrupt Request Mask	R/W
b9	SDD3INM	SDHI_D3 Insertion Interrupt Request Mask	R/W
b31-b10	-	Reserved	R

mask2

SD interrupt mask register 2 (SDIMSK2) control

To enable an interrupt, set the target bit to 1.

To not change the interrupt setting, clear the target bit to 0.

However, setting to the Read Only bit is invalid.

Bit	Symbol	Bit Name	R/W
b0	CMDEM	Command Error Interrupt Request Mask	R/W
b1	CRCEM	CRC Error Interrupt Request Mask	R/W
b2	ENDEM	End Bit Error Interrupt Request Mask	R/W
b3	DTTOM	Data Timeout Interrupt Request Mask	R/W
b4	ILWM	SDBUFR Register Illegal Write Interrupt Request Mask	R/W
b5	ILRM	SDBUFR Register Illegal Read Interrupt Request Mask	R/W
b6	RSPTOM	Response Timeout Interrupt Request Mask	R/W
b7	-	Reserved	R
b8	BREM	BRE Interrupt Request Mask	R/W
b9	BWEM	BWE Interrupt Request Mask	R/W
b14-b10	-	Reserved	R
b15	ILAM	Illegal Access Error Interrupt Request Mask	R/W
b31-b16	-	Reserved	R

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Controls SD interrupt mask register 1 (SDIMSK1) and SD interrupt mask register 2 (SDIMSK2) to enable SD interrupts.

Example

```
#define SDHI_SDIMSK1_DATA_TRNS    (0x0004u)    /* Command sequence end */
#define SDHI_SDIMSK2_BWE          (0x8a7fu)    /* Write enable and All errors*/

R_SDHI_SetIntMask(SDHI_CH0, SDHI_SDIMSK1_DATA_TRNS, SDHI_SDIMSK2_BWE);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

After calling this function, call the R_SDHI_EnableIcuInt() function to enable SDHI ICU controller interrupts. If not enabled, no SD interrupts will be generated.

R_SDHI_ClearIntMask()

This function controls the SD interrupt mask registers to disable SD interrupts.

Format

```
sdhi_status_t R_SDHI_ClearIntMask(
  uint32_t channel,
  uint32_t mask1,
  uint32_t mask2
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

mask1

SD interrupt mask register 1 (SDIMSK1) control
To disable an interrupt, set the target bit to 1.
To not change the interrupt setting, clear the target bit to 0.
However, setting to the Read Only bit is invalid.
For details of the SDIMSK1 register, refer to "R_SDHI_SetIntMask()".

mask2

SD interrupt mask register 2 (SDIMSK2) control
To disable an interrupt, set the target bit to 1.
To not change the interrupt setting, clear the target bit to 0.
However, setting to the Read Only bit is invalid.
For details of the SDIMSK2 register, refer to "R_SDHI_SetIntMask()".

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Controls SD interrupt mask register 1 (SDIMSK1) and SD interrupt mask register 2 (SDIMSK2) to disable SD interrupts.

Example

```
#define SDHI_SDIMSK1_DATA_TRNS    (0x0004u)    /* Command sequence end */
#define SDHI_SDIMSK2_BWE          (0x8a7fu)    /* Write enable and All errors*/

R_SDHI_ClearIntMask(SDHI_CH0, SDHI_SDIMSK1_DATA_TRNS, SDHI_SDIMSK2_BWE);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

Before calling this function, call the R_SDHI_DisableIcuInt() function to disable SDHI ICU controller interrupts. If not disabled, SD interrupts may be generated at unintended times.

R_SDHI_ClearSdstsReg()

This function controls the SD status registers to clear interrupt flags.

Format

```
sdhi_status_t R_SDHI_ClearSdstsReg(
  uint32_t channel,
  uint32_t clear_sdsts1,
  uint32_t clear_sdsts2
)
```

Parameters

channel

Channel number SDHI channel number to be used (starting from 0)

clear_sdsts1

SD status register 1 (SDSTS1) control

To 0 clear an interrupt flag, set the target bit to 1.
To not change the interrupt flag, clear the target bit to 0.
However, setting to the Read Only bit is invalid.

Bit	Symbol	Bit Name	R/W
b0	RSPEND	Response End Detection Flag	R/W
b1	-	Reserved	R
b2	ACEND	Access End Detection Flag	R/W
b3	SDCDRM	SDHI_CD Removal Flag	R/W
b4	SDCDIN	SDHI_CD Insertion Flag	R/W
b5	SDCDMON	SDHI_CD Pin Monitor Flag	R
b6	-	Reserved	R
b7	SDWPMON	SDHI_WP Pin Monitor Flag	R
b8	SDD3RM	SDHI_D3 Removal Flag	R/W
b9	SDD3IN	SDHI_D3 Insertion Flag	R/W
b10	SDD3MON	SDHI_D3 Pin Monitor Flag	R
b31-b11	-	Reserved	R

clear_sdsts2

SD status register 2 (SDSTS2) control

To 0 clear an interrupt flag, set the target bit to 1.
To not change the interrupt flag, clear the target bit to 0.
However, setting to the Read Only bit and b12 (Reserved bit) is invalid.

Bit	Symbol	Bit Name	R/W
b0	CMDE	Command Error Detection Flag	R/W
b1	CRCE	CRC Error Detection Flag	R/W
b2	ENDE	End Bit Error Detection Flag	R/W
b3	DTO	Data Timeout Detection Flag	R/W
b4	ILW	SDBUFR Illegal Write Access Detection Flag	R/W
b5	ILR	SDBUFR Illegal Read Access Detection Flag	R/W
b6	RSPTO	Response Timeout Detection Flag	R/W
b7	SDD0MON	SDHI_D0 Pin Status Flag	R
b8	BRE	SDBUFR Read Enable Flag	R/W
b9	BWE	SDBUFR Write Enable Flag	R/W
b10	-	Reserved	R
b11	-	Reserved (*1)	R

b12	-	Reserved	R
b13	SDCLKCREN	SDCLKCR Write Enable Flag	R
b14	CBSY	Command Sequence Status Flag	R
b15	ILA	Illegal Access Error Detection Flag	R/W
b31-b16	-	Reserved	R

Note1 Writes "1" to the target bit irrespective of the setting value.

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Clears interrupt flags in SD status register 1 (SDSTS1) and SD status register 2 (SDSTS2).

Example

```
#define SDHI_SDIMSK1_TRNS_RESP    (0x0005u)
                                  /* Command sequence end and Response end */
#define SDHI_SDIMSK2_CLEAR        (0x837fu)
                                  /* All initialization clear */

R_SDHI_ClearSdstsReg(SDHI_CH0, SDHI_SDIMSK1_TRNS_RESP, SDHI_SDIMSK2_CLEAR);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

Before calling this function, call the R_SDHI_DisableIcuInt() function to disable SDHI ICU controller interrupts. If not disabled, SD interrupts may be generated at unintended times.

R_SDHI_SetSdioIntMask()

This function controls the SDIO interrupt mask registers to enable SDIO interrupts.

Format

```
sdhi_status_t R_SDHI_SetSdioIntMask(
  uint32_t channel,
  uint32_t mask
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

mask

SDIO interrupt mask register (SDIOIMSK) control

To enable an interrupt, set the target bit to 1.

To not change the interrupt setting, clear the target bit to 0.

However, setting to the Read Only bit and b2-b1 (Reserved bit) is invalid.

Bit	Symbol	Bit Name	R/W
b0	IOIRQM	IOIRQ Interrupt Mask Control	R/W
b2-b1	-	Reserved (*1)	R/W
b13-b3	-	Reserved	R
b14	EXPUB52M	EXPUB52 Interrupt Request Mask Control	R/W
b15	EXWTM	EXWT Interrupt Request Mask Control	R/W
b31-b16	-	Reserved	R

Note1 Writes "1" to the target bit irrespective of the setting value.

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Controls the SDIO interrupt mask register (SDIOIMSK) to enable interrupts.

Example

```
#define SDHI_SDIOIMSK_IOIRQ   (0x0001u)   /* Interrupt from IO Card */

R_SDHI_SetSdioIntMask(SDHI_CH0, SDHI_SDIOIMSK_IOIRQ);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

After calling this function, call the R_SDHI_EnableIcuInt() function to enable SDHI ICU controller interrupts. If not enabled, no SDIO interrupts will be generated.

R_SDHI_ClearSdioIntMask()

This function controls the SDIO interrupt mask registers to disable SDIO interrupts.

Format

```
sdhi_status_t R_SDHI_ClearSdioIntMask(
  uint32_t channel,
  uint32_t mask
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

mask

SDIO interrupt mask register (SDIOIMSK) control

To disable an interrupt, set the target bit to 1.

To not change the interrupt setting, clear the target bit to 0.

However, setting to the Read Only bit and b2-b1 (Reserved bit) is invalid.

For details of the SDIOIMSK register, refer to "R_SDHI_SetSdioIntMask()".

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Controls the SDIO interrupt mask register (SDIOIMSK) to disable interrupts.

Example

```
#define SDHI_SDIOIMSK_IOIRQ   (0x0001u)   /* Interrupt from IO Card */

R_SDHI_ClearSdioIntMask(SDHI_CH0, SDHI_SDIOIMSK_IOIRQ);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

Before calling this function, call the R_SDHI_DisableIcuInt() function to disable SDHI ICU controller interrupts. If not disabled, SD interrupts may be generated at unintended times.

R_SDHI_ClearSdiostsReg()

This function controls the SDIO status registers to clear interrupt flags.

Format

```
sdhi_status_t R_SDHI_ClearSdiostsReg(
  uint32_t channel,
  uint32_t clear
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

clear

SDIO status register (SDIOSTS) control

To clear an interrupt flag, set the target bit to 1.

To not change the interrupt flag, clear the target bit to 0.

However, setting to the Read Only bit and b2-b1 (Reserved bit) is invalid.

Bit	Symbol	Bit Name	R/W
b0	IOIRQ	SDIO Interrupt Status Flag	R/W
b2-b1	-	Reserved (*1)	R/W
b13-b3	-	Reserved	R
b14	EXPUB52	EXPUB52 Status Flag	R/W
b15	EXWT	EXWT Status Flag	R/W
b31-b16	-	Reserved	R

Note1 Writes "1" to the target bit irrespective of the setting value.

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Clears interrupt flags in the SDIO status register (SDIOSTS).

Example

```
#define SDHI_SDIOIMSK_IOIRQ   (0x0001u)   /* Interrupt from IO Card */

R_SDHI_ClearSdiostsReg(SDHI_CH0, SDHI_SDIOIMSK_IOIRQ);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

Before calling this function, call the R_SDHI_DisableIcuInt() function to disable SDHI ICU controller interrupts. If not disabled, SD interrupts may be generated at unintended times.

R_SDHI_SetClock()

This function turns the SD clock on and off.

Format

```
sdhi_status_t R_SDHI_SetClock(
  uint32_t channel,
  uint32_t div,
  int32_t enable
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

div

Use the following setting values:

High-speed mode: SDHI_CFG_DIV_HIGH_SPEED
Default-speed mode: SDHI_CFG_DIV_DEFAULT_SPEED
Card-recognition mode: SDHI_CFG_DIV_INIT_SPEED

Refer to 2.7, Compile Time Settings, regarding the definitions of the above.

enable

Use the following setting values:

Clock stopped: SDHI_CLOCK_DISABLE
Clock supplied: SDHI_CLOCK_ENABLE

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Turns the SD clock on and off.

Example

```
/* Supply the clock */
if (R_SDHI_SetClock(SDHI_CH0, SDHI_CFG_DIV_INIT_SPEED, SDHI_CLOCK_ENABLE) !=
SDHI_SUCCESS)
{
   /* Error */
}

/* Stop the clock */
if (R_SDHI_SetClock(SDHI_CH0, 0, SDHI_CLOCK_DISABLE) != SDHI_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_SetBus()

This function makes SD bus settings.

Format

```
sdhi_status_t R_SDHI_SetBus(
  uint32_t channel,
  uint32_t width
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

width

Use the following setting values:

1-bit bus: SDHI_PORT_1BIT

4-bit bus: SDHI_PORT_4BIT

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Controls the SD bus width select bit (SDOPT.WIDTH) to set the SD bus to 1-bit or 4-bit operation.

Example

```
R_SDHI_SetBus(SDHI_CD0, SDHI_PORT_1BIT);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

Do not call this function while a command sequence is running (SDSTS2.CBSY = 1).

R_SDHI_GetResp()

This function gets the response from the SD card.

Format

```
sdhi_status_t R_SDHI_GetResp(
  uint32_t channel,
  sdhi_get_resp_t * p_resp_reg
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

*p_resp_reg

Response register information structure

sdrsp10: Variable stored in response register 10

sdrsp32: Variable stored in response register 32

sdrsp54: Variable stored in response register 54

sdrsp76: Variable stored in response register 76

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Stores the values contained in the response registers (SDRSP10, SDRSP32, SDRSP54, and SDRSP76) in the response register information structure. Divides and stores the contents of the response among register sdrsp10, sdrsp32, sdrsp54, and sdrsp76, according to the response type. Table 3.1 shows the correspondence between the response register information structure and response storage destinations.

Table 3.1 Response Register Information Structure and Response Storage Destinations

Response Type	sdrsp76	sdrsp54	sdrsp32	sdrsp10
R1	—	[39:8]*1	—	[39:8]
R1b	—	[39:8]*1	—	[39:8]
R2	[127:104]	[103:72]	[71:40]	[39:8]
R3	—	—	—	[39:8]
R4	—	—	—	[39:8]
R5	—	—	—	[39:8]
R6	—	—	—	[39:8]
R7	—	—	—	[39:8]

Note 1. Responses to CMD18 or CMD25 are stored in both the SDRSP10 and SDRSP54 registers. Therefore, it is possible to check responses to CMD18 or CMD25 by referencing the value stored in the SDRSP54 register, even if an automatically sent response to CMD12 overwrites the contents of the SDRSP10 register.

Example

```
sdhi_get_resp_t   resp_reg;

R_SDHI_GetResp(channel, &resp_reg);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_OutReg()

This function sets the SDHI registers.

Format

```
sdhi_status_t R_SDHI_OutReg(
  uint32_t channel,
  uint32_t reg,
  uint32_t data
)
```

Parameters

channel

Channel number SDHI channel number to be used (starting from 0)

reg

SDHI base register offset value. Refer to the table below when setting macro definitions.

Register Name	Offset	Macro Definition
Command register (SDCMD)	0x00u	SDHI_SDCMD
Argument register (SDARG)	0x08u	SDHI_SDARG
Data stop register (SDSTOP)	0x10u	SDHI_SDSTOP
Block count register (SDBLKCNT)	0x14u	SDHI_SDBLKCNT
Response register 10 (SDRESP10)	0x18u	SDHI_SDRESP10
Response register 32 (SDRESP32)	0x20u	SDHI_SDRESP32
Response register 54 (SDRESP54)	0x28u	SDHI_SDRESP54
Response register 76 (SDRESP76)	0x30u	SDHI_SDRESP76
SD status register 1 (SDSTS1)	0x38u	SDHI_SDSTS1
SD status register 2 (SDSTS2)	0x3cu	SDHI_SDSTS2
SD interrupt mask register 1 (SDIMSK1)	0x40u	SDHI_SDIMSK1
SD interrupt mask register 2 (SDIMSK2)	0x44u	SDHI_SDIMSK2
SDHI clock control register (SDCLKCR)	0x48u	SDHI_SDCLKCR
Transfer data size register (SDSIZE)	0x4cu	SDHI_SDSIZE
Card access option register (SDOPT)	0x50u	SDHI_SDOPT
SD error status register 1 (SDERSTS1)	0x58u	SDHI_SDERSTS1
SD error status register 2 (SDERSTS2)	0x5cu	SDHI_SDERSTS2
SD buffer register (SDBUFR)	0x60u	SDHI_SDBUFR
SDIO mode control register (SDIOMD)	0x68u	SDHI_SDIOMD
SDIO status register (SDIOSTS)	0x6cu	SDHI_SDIOSTS
SDIO interrupt mask register (SDIOIMSK)	0x70u	SDHI_SDIOIMSK
DMA transfer enable register (SDDMAEN)	0x1b0u	SDHI_SDDMAEN
SDHI software reset register (SDRST)	0x1c0u	SDHI_SDRST
Version register (SDVER)	0x1c4u	SDHI_SDVER
Swap control register (SDSW AP)	0x1e0u	SDHI_SDSWAP

data

Register setting value

Return Values

SDHI_SUCCESS *Successful operation*

SDHI_ERR *General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Sets SDHI registers.

Example

```
R_SDHI_OutReg(SDHI_CD0, SDHI_SDCMD, cmd);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_InReg()

This function gets the value of an SDHI register.

Format

```
sdhi_status_t R_SDHI_InReg(
  uint32_t channel,
  uint32_t reg,
  uint32_t * p_data
)
```

Parameters

channel

Channel number SDHI channel number to be used (starting from 0)

reg

SDHI base register offset value. Refer to the table in R_SDHI_OutReg(), when setting macro definitions.

** p_data*

Pointer to storage destination of acquired register value

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Sets SDHI registers.

Example

```
R_SDHI_InReg(SDHI_CD0, SDHI_SDSTS1, &sdstas1);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_CDLayout()

This function checks whether the SDHI_CD (SD card detection) pin is used.

Format

```
sdhi_status_t R_SDHI_CDLayout(
  uint32_t channel
)
```

Parameters

channel

Channel number	SDHI channel number to be used (starting from 0)

Return Values

SDHI_SUCCESS	*CD pin used.*
SDHI_ERR	*CD pin not used.*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Checks whether the CD pin is used.

Example

```
if (R_SDHI_CDLayout(SDHI_CD0) == SDHI_SUCCESS)
{
   /* User setting */
}
```

Special Notes

None

R_SDHI_WPLayout()

This function checks whether the SDHI_WP (SD write protect) pin is used.

Format

```
sdhi_status_t R_SDHI_WPLayout(
  uint32_t channel
)
```

Parameters

channel

Channel number	SDHI channel number to be used (starting from 0)

Return Values

SDHI_SUCCESS	*WP pin used.*
SDHI_ERR	*WP pin not used.*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Checks whether the WP pin is used.

Example

```
if (R_SDHI_WPLayout(SDHI_CD0) == SDHI_SUCCESS)
{
   /* User setting */
}
```

Special Notes

None

R_SDHI_GetWP()

This function gets the state of the SDHI_WP (SD write protect) pin.

Format

```
sdhi_status_t R_SDHI_GetWP(
  uint32_t channel,
  uint32_t * p_wp
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

** p_wp*

Pointer to storage destination of SDHI_WP pin state

0: SDHI_WP pin level is high.

1: SDHI_WP pin level is low.

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Gets the SDHI_WP pin state.

Example

```
R_SDHI_GetWP(SDHI_CH0, &wp);
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

To execute this function, the terminal setting processing of the SDHI_WP terminal is necessary. For details, refer to "4 Pin Settings".

R_SDHI_GetSpeedType()

This function gets information on the speed modes supported by the target microcontroller.

Format

```
sdhi_status_t R_SDHI_GetSpeedType(
  uint32_t channel
)
```

Parameters

channel

Channel number	SDHI channel number to be used (starting from 0)

Return Values

SDHI_SUCCESS	*Compatible with default-speed and high-speed modes*
SDHI_ERR	*Compatible with default-speed mode*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Gets information on the speed modes supported by the target microcontroller.

Example

```
if (R_SDHI_GetSpeedType(SDHI_CH0) == SDHI_SUCCESS)
{
   /* User setting */
}
```

Special Notes

None

R_SDHI_GetBuffRegAddress()

This function gets the address of the SD buffer register.

Format

```
sdhi_status_t R_SDHI_GetBuffRegAddress(
  uint32_t channel,
  uint32_t *p_reg_buff
)
```

Parameters

channel

Channel number — SDHI channel number to be used (starting from 0)

**p_reg_buff*

Pointer to SD buffer register address

Return Values

SDHI_SUCCESS	*Successful operation*
SDHI_ERR	*General error*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Gets the SD buffer register address and stores it in the buffer.

Used for example when setting data register addresses to be used for DMAC or DTC transfers.

Example

```
uint32_t    reg_buff = 0;

if (R_SDHI_Get_BuffRegAddress(SDHI_CH0, &reg_buff) != SDHI_SUCCESS)
{
   /* Error */
}
```

Special Notes

Before running this function, initialization processing by the R_SDHI_Open() function is required.

R_SDHI_GetVersion()

This function gets the driver version information.

Format

```
uint32_t R_SDHI_GetVersion(
  void
)
```

Parameters

None

Return Values

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

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Returns the driver version information.

Example

```
uint32_t version;
version = R_SDHI_GetVersion();
```

Special Notes

None

R_SDHI_SetLogHdlAddress()

This function gets the handler address of the LONGQ FIT module.

Format

```
sdhi_status_t  R_SDHI_SetLogHdlAddress(
  uint32_t user_long_que
)
```

Parameters

user_long_que

LONGQ FIT module handler address

Return Values

SDHI_SUCCESS	*Successful operation*

Properties

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Sets the handler address of the LONGQ FIT module to point to the SDHI FIT module.

Example

```
#define ERR_LOG_SIZE (16)
#define SDHI_USER_LONGQ_IGN_OVERFLOW    (1)

sdhi_status_t     ret = SDHI_SUCCESS;
uint32_t          MtlLogTbl[ERR_LOG_SIZE];
longq_err_t       err;
longq_hdl_t       p_sdhi_user_long_que;
uint32_t          long_que_hndl_address;


/* Open LONGQ module. */
err = R_LONGQ_Open(&MtlLogTbl[0],
                   ERR_LOG_SIZE,
                   SDHI_USER_LONGQ_IGN_OVERFLOW,
                   &p_sdhi_user_long_que
);

long_que_hndl_address = (uint32_t)p_sdhi_user_long_que;
ret = R_SDHI_SetLogHdlAddress(long_que_hndl_address);
```

Special Notes

Preparatory processing is performed to obtain error logs using the LONGQ FIT module. Perform this processing before calling the R_SDHI_Open() function.

Add the separately available LONGQ FIT module to your project.

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

R_SDHI_Log()

This function gets the error log.

Format

```
uint32_t R_SDHI_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

Prototype declaration contained in r_sdhi_rx_if.h.

Description

Gets the error log.

To end getting the error log, call the function.

Example

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

if (R_SDHI_Open(SDHI_CH0) != SDHI_SUCCESS)
{
   /* Error */
   R_SDHI_Log(USER_DRIVER_ID, USER_LOG_MAX, USER_LOG_ADR_MAX);
}
```

Special Notes

Use the debugging module.

Add the separately available LONGQ FIT module to your project.

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

4. Pin Settings

To use the 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. Refer to Table 4.1 for details.

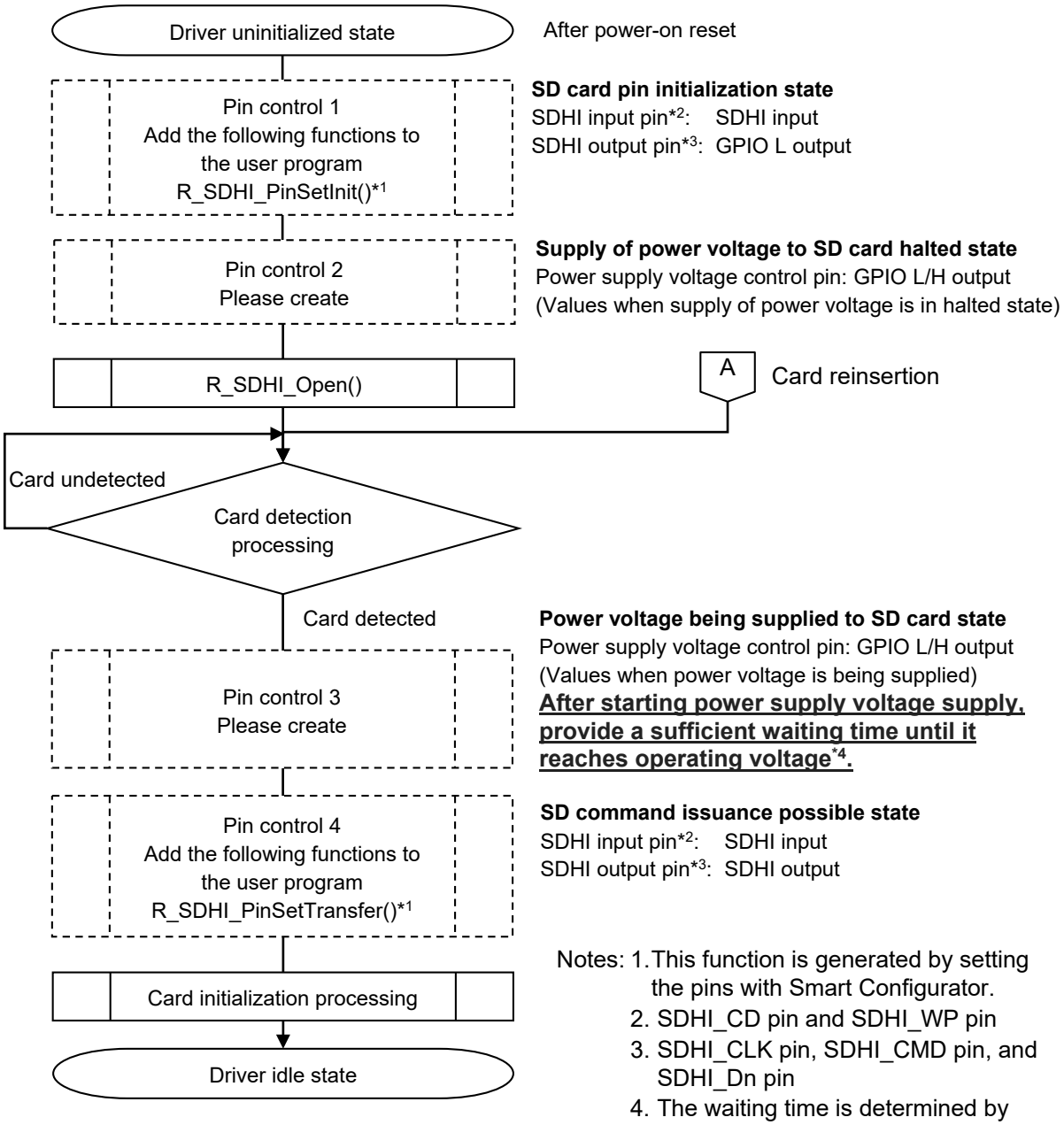
The pin assignment is referred to as the “Pin Setting” in this document. Also, GPIO control is required. Refer to 4.1,
SD Card Insertion and Power-On Timing, and 4.2, SD Card Removal and Power-Off Timing, and create appropriate program code to provide this processing.

Table 4.1 Function Output by the Smart Configurator

Function to be Output	Function
R_SDHI_PinSetInit()	Performs initialization of the SDHI pins. After execution, only SDHI_CD and SDHI_WP pins are valid.
R_SDHI_PinSetTransfer()	Sets SDHI pins to SD command issuance possible state. After execution, all SDHI pins are valid.
R_SDHI_PinSetDetection()	Sets SDHI pins to SD command issuance impossible state. After execution, only SDHI_CD and SDHI_WP pins are valid.
R_SDHI_PinSetEnd()	Sets SDHI pins to SDHI control disabled state. After execution, all SDHI pins are invalid.

4.1 SD Card Insertion and Power-On Timing

Figure 4.1 and Table 4.2 show the control procedure. Perform the SD card insertion procedure after successful operation of the R_SDHI_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.



Notes: 1. This function is generated by setting the pins with Smart Configurator.
2. SDHI_CD pin and SDHI_WP pin
3. SDHI_CLK pin, SDHI_CMD pin, and SDHI_Dn pin
4. The waiting time is determined by the configuration of the SD card power supply circuit.

Figure 4.1 SD Card Insertion and Power-On Timing

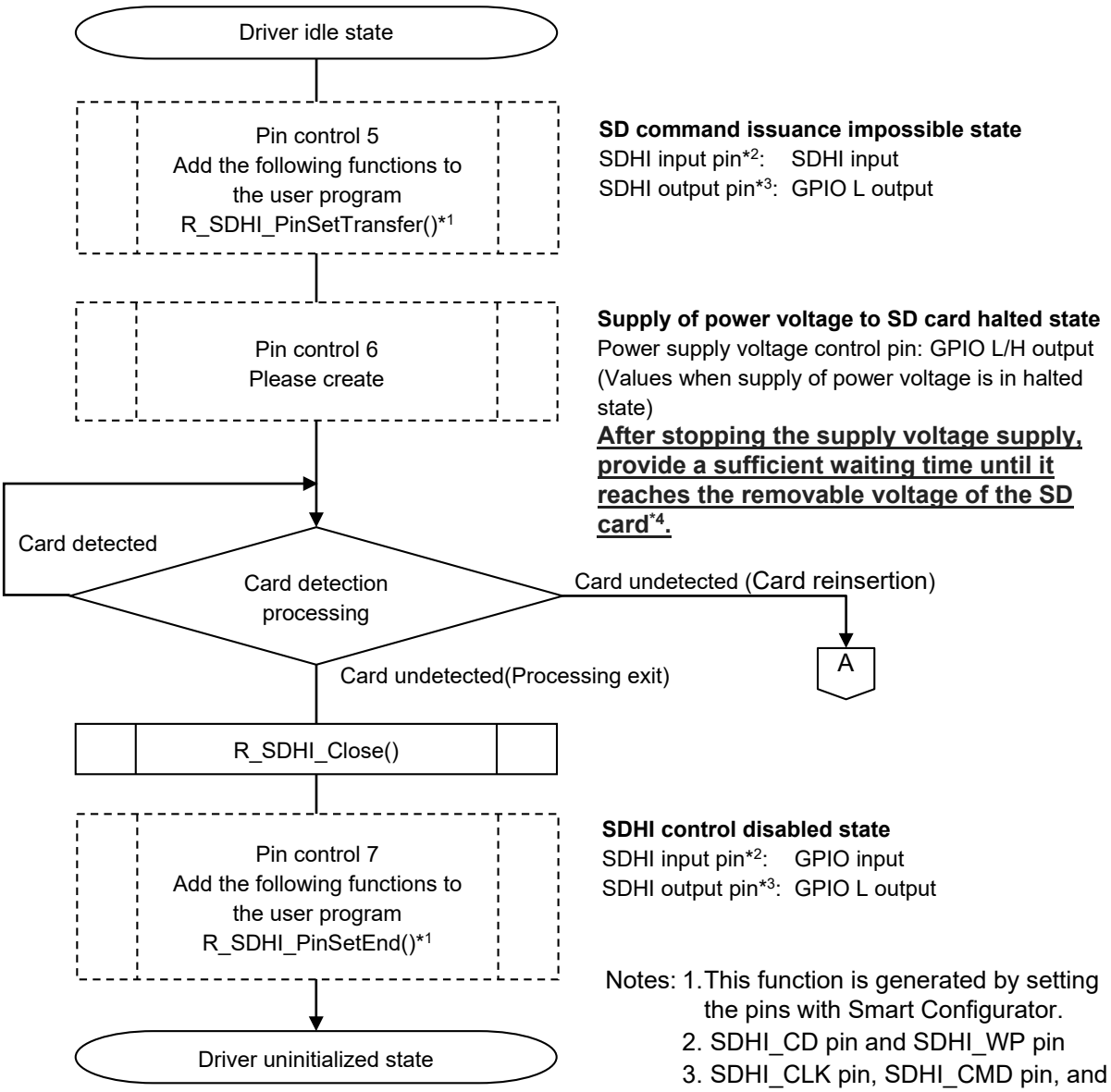
Table 4.2 User Setting Method at SD Card Insertion

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)

Note 1. SDHI_CD pin and SDHI_WP pin
Note 2. SDHI_CLK pin, SDHI_CMD pin, and SDHI_Dn pin
Note 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.
Note 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 4.3 show the control procedure. Perform the SD card removal procedure after successful operation of the R_SDHI_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.



Notes: 1. This function is generated by setting the pins with Smart Configurator.
2. SDHI_CD pin and SDHI_WP pin
3. SDHI_CLK pin, SDHI_CMD pin, and SDHI_Dn pin
4. The waiting time is determined by the configuration of the SD card power supply circuit.

Figure 4.2 SD Card Removal and Power-Off Timing

Table 4.3 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 supplied/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

Note 1. SDHI_CD pin and SDHI_WP pin
Note 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.

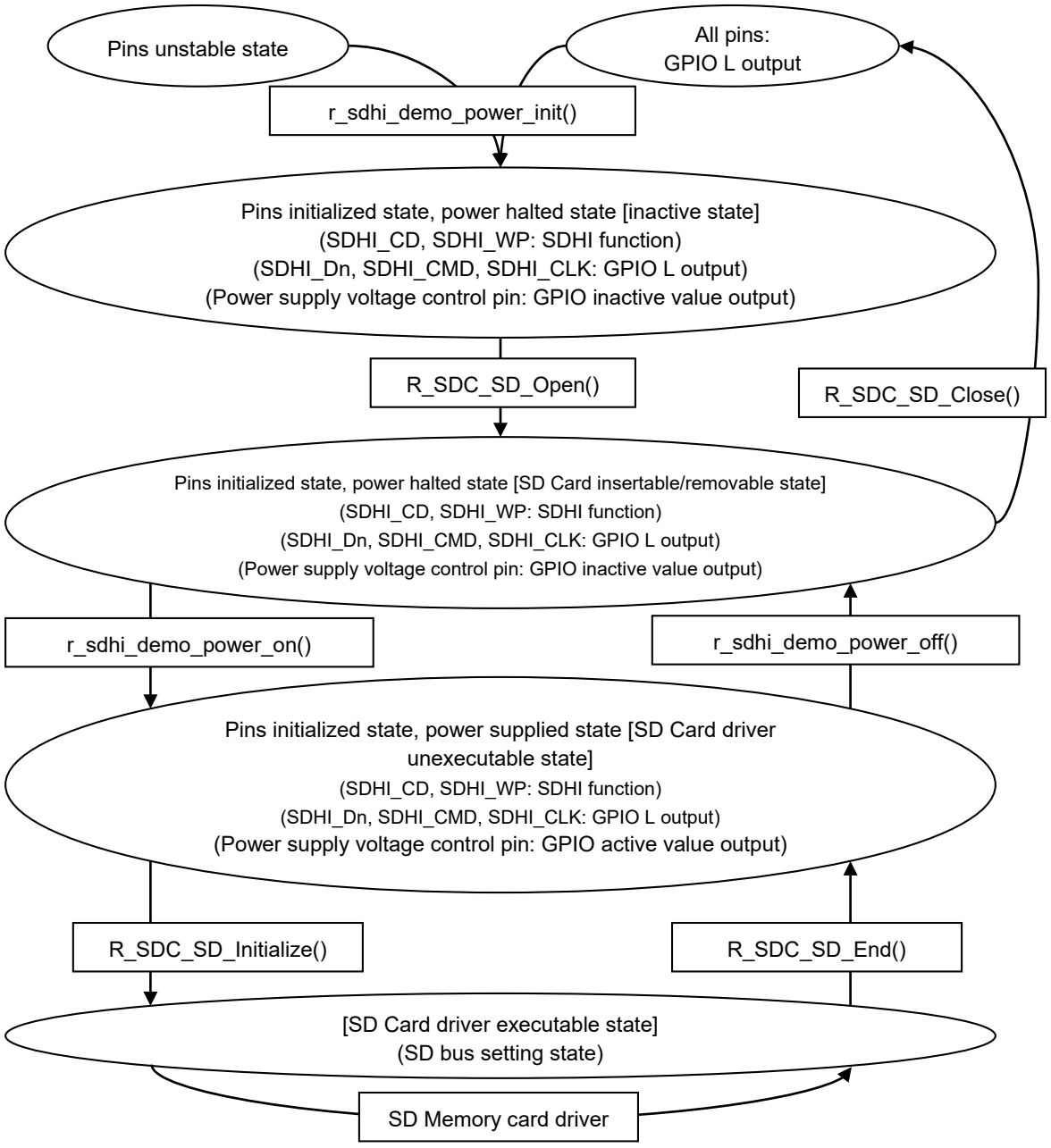


Figure 5.1 State Transition Diagram

5.3 Configuration Overview

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

The table below lists the option names and set values when the RX64M RSK, RX65N-2M RSK or RX72N RSK is used.

Configuration options in *r_sdhi_rx_demo_pin_config.h*	
#define SDHI_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 SDHI_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 SDHI_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 SDHI_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_SDHI_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_SDHI_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_sdhi_demo_power_init()	Initializes the power supply voltage control pin settings
r_sdhi_demo_power_on()	Starts supply of power supply voltage
r_sdhi_demo_power_off()	Stops supply of power supply voltage
r_sdhi_demo_softwaredelay()	Performs delay

(1) r_sdhi_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

```
sdhi_status_t r_sdhi_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

SDHI_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_sdhi_demo_power_on()

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

Format

```
sdhi_status_t r_sdhi_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

SDHI_SUCCESS	*Successful operation*
SDHI_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 SDHI_CFG_POWER_ON_WAIT in r_sdhi_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_sdhi_demo_softwaredelay() function in order to wait until the operating voltage is reached. Set the wait time using SDHI_CFG_POWER_ON_WAIT in section 5.3, Compile Time Settings.

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

(3) r_sdhi_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
```
sdhi_status_t r_sdhi_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

SDHI_SUCCESS	*Successful operation*
SDHI_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 SDHI_CFG_POWER_OFF_WAIT in r_sdhi_rx_demo_pin_config.h has elapsed, returns the result.

Special Notes
After stopping the supply of power supply voltage, executes the r_sdhi_demo_softwaredelay() function in order to wait until the removable voltage is reached. Set the wait time using SDHI_CFG_POWER_OFF_WAIT in section 5.3, Compile Time Settings.

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

(4) r_sdhi_demo_softwaredelay()

This function is used when waiting for a particular time.

Format
```
bool r_sdhi_demo_softwaredelay(
  uint32_t delay,
  sdhi_delay_units_t units
)
```

Parameters
delay

Timeout time (Units: set with the units)

units

Microseconds: SDHI_DELAY_MICROSECS
Milliseconds: SDHI_DELAY_MILLISECS
Seconds: SDHI_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 SDHI_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 SDHI_CFG_POWER_OFF_WAIT.

5.5 Replacing Wait Time Processing with Operating System Processing

The r_sdhi_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).

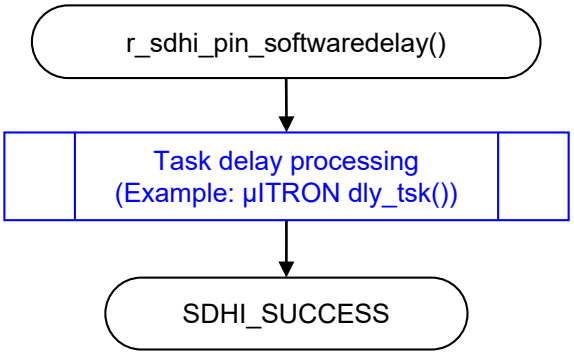


Figure 5.2 Wait Example Using Operating System Task Delay Processing

5.6 sdhi_demo_rskrx64m, sdhi_demo_rskrx65n_2m, sdhi_demo_rskrx72n, sdhi_demo_rskrx64m_gcc, sdhi_demo_rskrx65n_2m_gcc, sdhi_demo_rskrx72n_gcc

Once the code is compiled and downloaded to the target board and is running, LED0 will turn ON after initialization. After the SDHI module is successfully opened, LED1 will turn ON. After data has been successfully written to the SD card, LED2 will turn ON. After data has been successfully read from the SD card, LED3 will turn ON. After the SDHI module is successfully closed, all LEDs will turn OFF.

Setup and Execution

1. Ensure driver support for channel 0 is enabled in r_sdhi_rx_config.h:
```
#define SDHI_CFG_CH0_INCLUDED (1)
```

2. Selection of data transfer module:
   When the DMAC transfer mode is used, set **Data transfer type** in Smart Configurator/Components/r_sdc_sdmem_rx to **DMAC transfers**.

   When the DTC transfer mode is used, set **Data transfer type** in Smart Configurator/Components/r_sdc_sdmem_rx to **DTC transfers**.

   By default, the transmit mode is **Software Transfers**.

3. Connect the RSK board to the PC (using Renesas E1 Emulator). The DC 5V 3A power adapter needs be plugged into the power jack (PWR) on the RSK board for external power supply. Build this sample application, download it to the board.

4. In Renesas e2 studio IDE, click the Renesas Views tab -> Move mouse over the Debug item, and select the Renesas Debug Virtual Console to show it.

5. By checking the log and LEDs, confirm that the application writes and reads data of 3 sectors (512-byte/sector) to the SD card.

Boards Supported

RSKRX64M, RSKRX65N-2M, RSKRX72N

5.7 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. To add a demo project to a workspace, select *File* >> *Import* >> *General* >> *Existing Projects into Workspace*, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

5.8 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 Browser* >> *Application Notes* tab.

6. Appendices

6.1 Operation Confirmation Environment

This section describes operation confirmation environment for the FIT module.

Table 6.1 Operation Confirmation Environment (Ver. 2.02)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.1.0
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endianness	Little endian
Version of the module	Ver.2.02
Board used	Renesas Starter Kit for RX65N (product No.: RTK500565NSxxxxxxx)

Table 6.2 Operation Confirmation Environment (Ver. 2.03)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0
C compiler	Renesas Electronics C/C++ Compiler for RX Family V3.01.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
Endianness	Little endian
Version of the module	Ver.2.03

Table 6.3 Operation Confirmation Environment (Ver. 2.04)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
	GCC for Renesas RX 4.08.04.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big endian/Little endian
Version of the module	Ver.2.04
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxx)

Table 6.4 Operation Confirmation Environment (Ver. 2.05)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big endian/Little endian
Version of the module	Ver.2.05
Board used	Renesas Starter Kit+ for RX72M (product No.: RTK5572Mxxxxxxxxxx)

Table 6.5 Operation Confirmation Environment (Ver. 2.06)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.4.0 IAR Embedded Workbench for Renesas RX 4.12.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
	GCC for Renesas RX 4.08.04.201902 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.12.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big endian/Little endian
Version of the module	Ver.2.06
Board used	Renesas Starter Kit+ for RX72N (product No.: RTK5572Nxxxxxxxxxx)

Table 6.6 Operation Confirmation Environment (Ver. 2.07)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio 2021-01 (21.1.0) IAR Embedded Workbench for Renesas RX 4.14.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.03.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = C99
	GCC for Renesas RX 8.03.00.202002 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.14.01 Compiler option: The default settings of the integrated development environment.
Endianness	Big endian/Little endian
Version of the module	Ver.2.07
Board used	Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx)

Table 6.7 Operation Confirmation Environment (Ver. 2.10)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 2022-10 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.04.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202202 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99 Linker option: The following user defined option should be added to the default settings of the integrated development environment, if “Optimize size (-Os)” is used: -Wl,--no-gc-sections This is to work around a GCC linker issue whereby the linker erroneously discard interrupt functions declared in FIT peripheral module
	IAR C/C++ Compiler for Renesas RX version 4.20.3 Compiler option: The default settings of the integrated development environment.
Endianness	Big endian/little endian
Version of the module	Rev.2.10
Board used	Renesas Starter Kit+ for RX64M (product No.: R0K50564Mxxxxxx) Renesas Starter Kit+ for RX65N-2MB (product No.: RTK50565N2Cxxxxxxx) Renesas Starter Kit+ for RX72N (product No.: RTK5572NNDCxxxxxx)

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:

- When using CS+:
  Application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)”
- When using e^2 studio:
  Application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)”

When using a FIT module, the board support package FIT module (BSP module) must also be added to the project. For this, 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_sdhi_rx module.

A: The FIT module you added may not support the target device chosen in the user project. Check if the FIT module supports the target device for the project used.

6.3 Using the SD Card Socket of the RSK for SD Card Evaluation

The procedure for performing read and write access to an SD card using a Renesas Starter Kit (RSK) is described below.

6.3.1 Hardware Settings

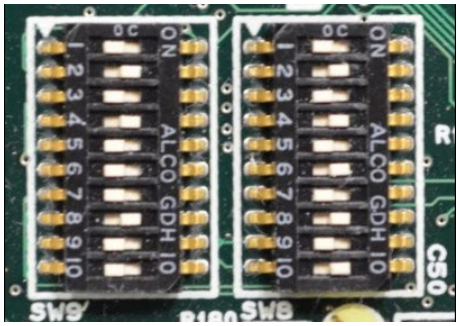
Settings must be made on the RSK for each target microcontroller.

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.

(5) RSK for RX72M

No settings are necessary.

(6) RSK for RX72N

No settings are necessary.

(7) RSK for RX671

No settings are necessary.

7. Reference Documents

User's Manual: Hardware
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User's Manual: Development Tools
RX Family CC-RX Compiler User's Manual (R20UT3248)
The latest version can be downloaded from the Renesas Electronics website.

8. Technical Updates

This module reflects the content of the following technical updates.

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

Revision History

Rev.	Date	Description	
		Page	Summary
2.00	Jul. 31, 2017	-	First edition issued
2.01	Dec. 31, 2017	44-45	For Figure 4.1 and Table 4.1 of 4.1 SD Card Insertion and Power-On Timing, updated the contents.
		46	For Figure 4.2 and Table 4.2 of 4.2 SD Card Removal and Power-Off Timing, updated the contents.
		48	For Table 5.1 of 5.1 Operation Confirmation Environment, updated the following contents. - Version of Integrated development environment - Version of the module
2.02	Nov. 30, 2018	-	Version upgrade for xml file update. Updated 7 Technical Updates.
2.03	Feb. 01, 2019	49	Added Table 5.2 Operation Confirmation Environment (Ver. 2.03)
		-	Changes associated with functions: Added support setting function of configuration option Using GUI on Smart Configurator. [Description] Added a setting file to support configuration option setting function by GUI.
2.04	May. 20, 2019	-	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Related Documents: Deleted R01AN1723 and R01AN1826. Added RX Family SD Mode SD Memory Card Driver Firmware Integration Technology(R01AN4233).
		1	Added Target Compilers.
		8	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		11	2.8 Code Size, amended.
		50	Added Table 5.3 Operation Confirmation Environment (Ver. 2.04).
2.05	Jul. 30, 2019	-	Changes associated with RX23W and RX72M.
		1	Changes Related Documents.
		8	Added RX23W and RX72M interrupt vector informations in 2.4 Interrupt Vectors.
		11	2.8 Code Size, amended.
		14	Added Section 2.13 “for”, “while” and “do while” statements.
		15-45	Delete “Reentrant” item on the API description page.
		52	Added Table 5.4 Operation Confirmation Environment (Ver. 2.05).
		54	Added RX72M Hardware Settings.
2.06	Nov. 22, 2019	-	Changes associated with RX66N and RX72N.
		1	Added Target Compilers.
		8	Added RX66N and RX72N interrupt vector informations in 2.4 Interrupt Vectors.
		11	2.8 Code Size, amended.
		44-45	Changed “Special Notes” in 3.26 R_SDHI_SetLogHdlAddress and 3.27 R_SDHI_Log.

Rev.	Date	Description	
		Page	Summary
2.06	Nov. 22, 2019	52	Added Table 5.5 Operation Confirmation Environment (Ver. 2.06).
		55	Added RX72N Hardware Settings.
2.07	Jun. 30, 2021	-	Changes associated with RX671.
		1	Added Target Compilers.
		8	Added RX671 interrupt vector informations in 2.4 Interrupt Vectors.
		11	2.8 Code Size, amended.
		13	Changed Section 2.12 Adding the FIT Module to Your Project.
		52	Added Table 5.6 Operation Confirmation Environment (Ver. 2.07).
		54	Added RX671 Hardware Settings.
2.10	Dec. 27, 2022	50	Added “5. Demo Projects”.
		55	Added RSKRX64M, RSKRX65N-2M, RSKRX72N to “5. Demo Projects”.
		59	6.1 Confirmed Operation Environment: Added Table for Rev. 2.10.
		Program	Added new demo projects. Updated slash format of included header file paths for Linux compatibility.

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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. 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.
6. 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.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. 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.5.0-1 October 2020)

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/.