RX Family

SDSI Module Using Firmware Integration Technology

Introduction

This application note describes RX Family SD Slave Interface (SDSI) control module and explains its use. The module is a SD slave control module using Firmware Integration Technology (FIT). It is referred to below as the SDSI FIT module. Other similar function control modules using FIT are referred to as FIT modules or as “function name” FIT modules.

Target devices

Microcontrollers used for operation check:

RX65N Group, RX651 Group

When applying the information in this application note to a microcontroller other than the above, modifications should be made as appropriate to match the specification of the microcontroller and careful evaluation performed.

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 Operating Environment".

Related Documents

- Firmware Integration Technology User’s Manual (R01AN1833)
- Board Support Package Module Firmware Integration Technology (R01AN1685)
- RX Family LONGQ Module Firmware Integration Technology (R01AN1889)

Contents

1. Overview

1.1 SDSI FIT Module

This module is incorporated into projects in the form of API functions. For instructions on adding this module to your project, see “2.12, Adding FIT Modules to Projects”.

1.2 Overview of SDSI FIT Module

SDSI built-in RX Family microcontroller is used to implement SD slave control.

Table 1.1 lists the peripheral devices used and their applications, and Figure 1-1 shows the usage example.

The following shows the Functions overview.

- SD slave control device driver using the SDSI, with the RX Family microcontroller as the master device
- Supports high-speed mode and default speed mode
- Supports block transfer mode and byte transfer mode
- Selectable from wide bus mode (4-bit) or default bus mode (1-bit)
- Supports SD mode. SPI mode is not supported.
- Supports CCCR (Card Common Control Register)-based operation
- Supports FBR (Function Basic Register)-based operation
- Supports access to CIS (Card Information Structure) 108 bytes
- Supports access to Function1 area (Function Unique register space)
- Supports direct transfer to MCU’s On-chip RAM using the DMA bus.
- Can call callback function when detecting SDSI interrupt
- Can control a single channel or multiple channels specified by the user
- Reentrancy from a different channel is possible.
- Operation with both big-endian and little-endian data order is supported.

Table 1.1 Peripheral Devices Used and the Usage

Peripheral Device	Usage
SDSI	Single or multiple channels (required)

Figure 1-1 Usage Example

1.3 Overview of APIs

Table 1.2 lists the API functions of SDSI FIT module.

Table 1.2 API Functions

Function Name	Description
R_SDSI_Open()	Driver Open processing
R_SDSI_Close()	Driver Close processing
R_SDSI_Initialize()	Initialization processing
R_SDSI_End()	End processing
R_SDSI_CflagPolling()	C flag polling processing
R_SDSI_WriteCisReg()	CIS Data Register write processing
R_SDSI_ReadCisReg()	CIS Data Register read processing
R_SDSI_WriteFuncReg()	FN1 Data Register n write processing (n=1,2,5)
R_SDSI_ReadFuncReg()	FN1 Data Register m read processing (m=1,3,5)
R_SDSI_WriteIntVectorReg()	FN1 Interrupt Vector Register write processing
R_SDSI_ReadIntVectorReg()	FN1 Interrupt Vector Register read processing
R_SDSI_ReadIntVectorClearReg()	FN1 Interrupt Clear Register read processing
R_SDSI_EnableDirectTrans()	DMA transfer enable processing
R_SDSI_DisableDirectTrans()	DMA transfer disable processing
R_SDSI_SetDirectTransAdr()	DMA transfer start address setting processing
R_SDSI_GetDirectTransAdr()	DMA transfer start address acquisition processing
R_SDSI_RegistIntCallback()	SDSI command interrupt callback function register processing
R_SDSI_RegistCdIntCallback()	SDSI card detection disable (Rise/Fall) interrupt callback function register processing
R_SDSI_RegistDtIntCallback()	SDSI DMA transfer end interrupt callback function register processing
R_SDSI_GetVersion()	Driver version information acquisition processing
R_SDSI_IntHandler0()	Interrupt handler
R_SDSI_SetLogHdlAddress()	LONGQ module handler address setting process
R_SDSI_Log()	Error log acquisition processing

1.4 Processing Example

1.4.1 Hardware

Figure 1-2 is a connection diagram. Using SDSI built-in microcontroller, SD mode of 1-bit/4-bit bus is controlled. One SD host per channel is connectable.

Refer to SDIO module specification to consider the circuit to match the system.

Pull-up resistance should be determined in reference to SD Specifications Part 1 Physical Layer Specification. To achieve high-speed operation, consider adding damping resistors or capacitors to improve the circuit matching of the various signal lines. But pull-up processing is not described here, as there is no rule for SD Specifications Part 1 Physical Layer Specification.

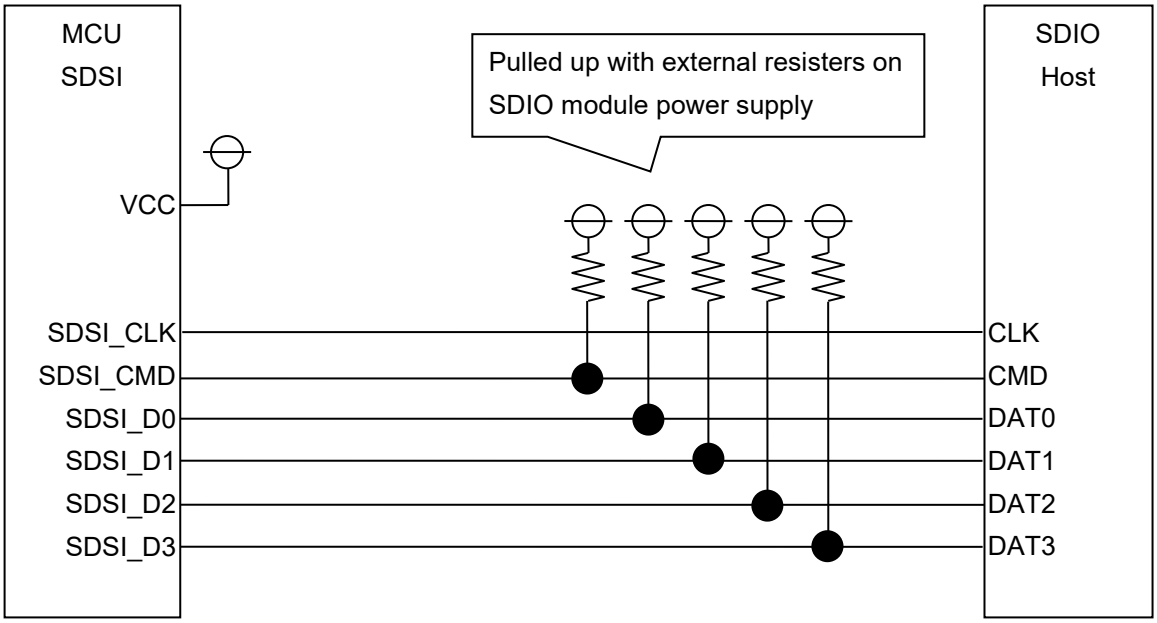


Figure 1-2 Sample Wiring Diagram for a RX Family MCU and SDIO module

(1) List of Pins

Table 1.3 lists the pins used and the functions.

Table 1.3 List of Pins Used and Functions

Pin Name	I/O	Description
SDSI_CLK	Input	SDSI Clock
SDSI_CMD	Input/Output	Command input, response output
SDSI_D3 to SDSI_D0	Input/Output	SDSI Data

1.4.2 Software

(1) Software Structure

Figure 1-3 shows the software structure.

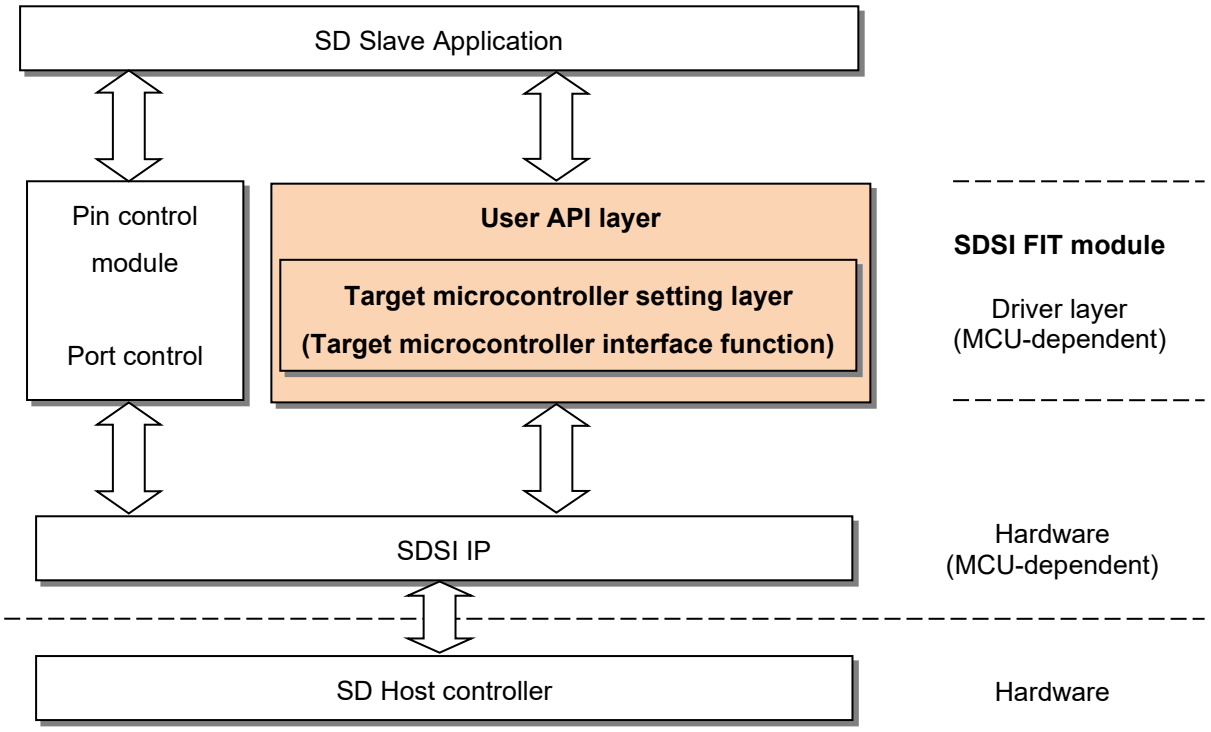


Figure 1-3 Software Structure

(a) User API layer (r_sdsi_rx.c)

This is SDSI FIT module user API, which is not dependent on the specifications of the microcontroller or the SDSI.

(b) Target microcontroller setting layer (r_sdsi_dev.c)

This performs read/write to SDSI register, which is dependent on the specifications of the microcontroller or the SDSI. The processing should be reviewed for each microcontroller.

(c) SD Slave application (r_sdsi_pxpd_rx.c)

SDSI FIT module control sample is included for reference.

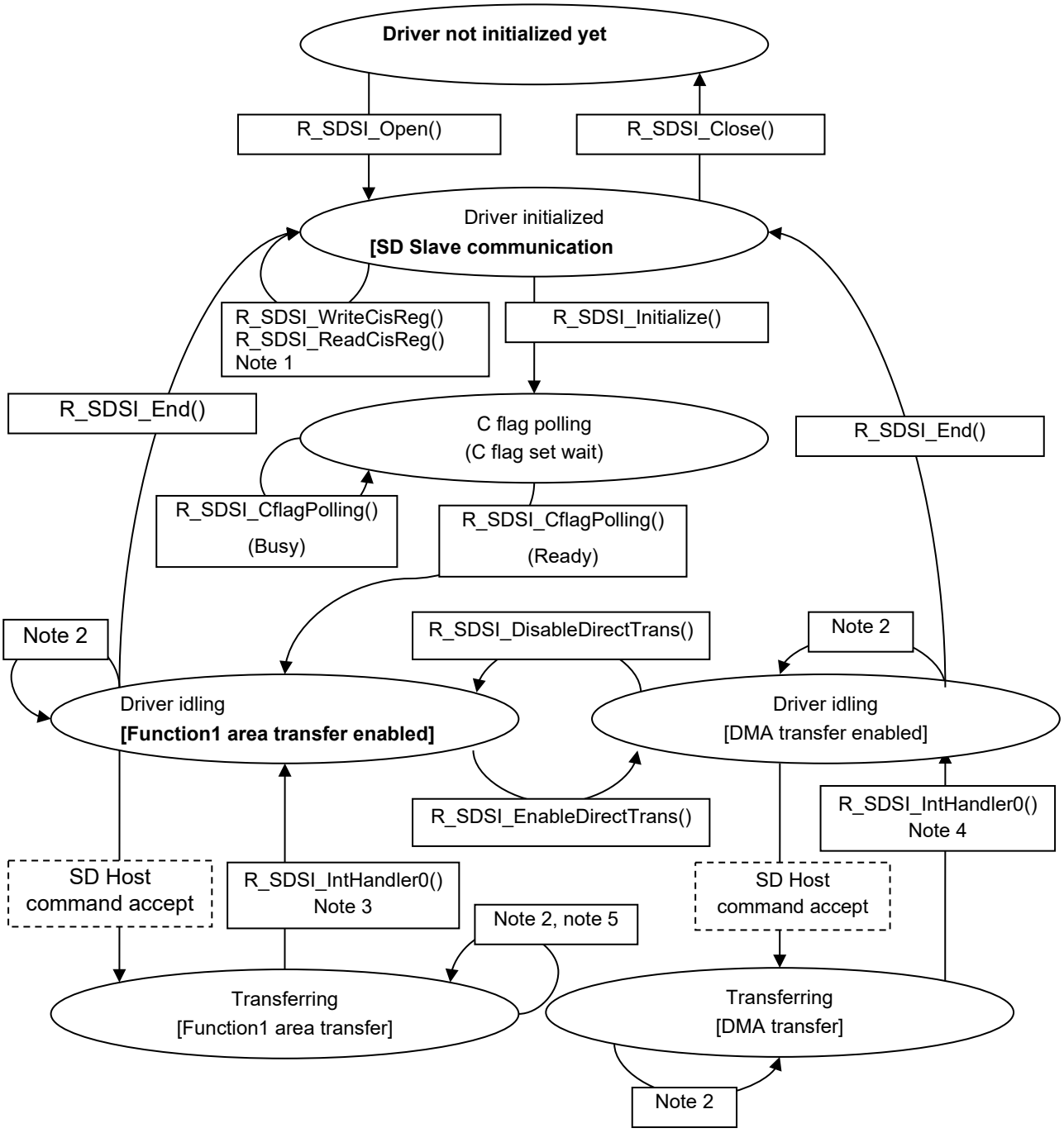
(2) 4-Byte Access to CIS Data Registers (CISDATAR), and Data Arrangement

Based on the SDSI IP specification, the SDSI FIT module accesses the CIS data registers (CISDATAR) in 4-byte units. In such cases, data is written to or read from the start address of the CIS data register (CISDATAR) in order, starting from the start address in RAM. Therefore, the arrangement of 4-byte data will differ depending on the endian setting.

```
typedef union
{
      uint32_t l;
      uint8_t c[4];
} sdsi_union_t;

sdsi_reg_t sdsi_reg;
sdsi_union_t io_buff = { 0 };

io_buff.c[0] = 0x01; /* RAM address 0 (start) */
io_buff.c[1] = 0x02; /* RAM address 1 */
io_buff.c[2] = 0x03; /* RAM address 2 */
io_buff.c[3] = 0x04; /* RAM address 3 */

/*
    Value of iobuff.l differs depending on endian setting.
    Little-endian: io_buff.l = 0x04030201
    Big-endian:    io_buff.l = 0x01020304
*/

sdsi_reg.offset = 0x0;
sdsi_reg.p_buff = &io_buff.l;
if (R_SDSI_WriteCisReg(SDSI_CH0, &sdsi_reg) != SDSI_SUCCESS)
{
      /* Error */
}
```

1.5 State Transition Diagram

Figure 1-4 shows the state transition diagram



Note 1 : R_SDSI_WriteCisReg() can be called only when driver is initialized.
Note 2 : R_SDSI_xxxxReg() other than R_SDSI_WriteCisReg() can be called.
Note 3 : When either interrupt CMD52_W or CMD53_W or CMD53_R in Function1,
or card detection disabled (Rise/Fall) is detected.
Note 4: When DMA transfer end interrupt is detected.
Note 5: For FN1 Data Register 5, simultaneous access from both SD Host/CPU are disabled.

Figure 1-4 State Transition Diagram

2. API Information

The names of the APIs of the control software follow the Renesas API naming standard.

2.1 Hardware Requirements

The microcontroller used must support the following functionality.

- SDSI

2.2 Software Requirements

This driver is dependent on the following packages.

r_bsp Rev.5.00 or higher

2.3 Supported Toolchain

The operation of the control software has been confirmed with the toolchain listed in 6.1.

2.4 Using Interrupt Vector

Executing the R_SDSI_Initialize() function enables the SDSI interrupt[1] corresponding to the channel.

Table 2.1 lists interrupt vector used by SDSI FIT module.

Table 2.1 Using Interrupt Vector

Device	Interrupt Vector
RX65N	GROUPBL2 Interrupt (Vector number: 107) • SDSI Interrupt [channel 0] (Group interrupt source number: 0)

2.5 Header Files

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

Configuration options for individual builds are selected in r_sdsi_rx_config.h.

```
#include "r_sdsi_rx_if.h"
```

2.6 Integer Types

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

[1] "CMD53 read command interrupt", "CMD53 write command interrupt", "CMD52 write command interrupt", “DMA transfer end interrupt” and “card detection disable (rise/fall) interrupt”

2.7 Compile Settings

The configuration option settings for the control software are specified in r_sdsi_rx_config.h.

The option names and setting values are described below.

Configuration options in *r_sdsi_rx_config.h*	
#define SDSI_CFG_USE_FIT Note: The default value is “enabled”.	Selects whether or not the SDSI FIT module is used in a BSP environment. When this option is set to “disabled”, control of FIT modules such as r_bsp is disabled. Also, the equivalent processing must be incorporated separately. When this option is set to “enabled”, control of FIT modules such as r_bsp is enabled.
#define SDSI_CFG_PARAM_CHECKING_ENABLE Note: The default value is “BSP_CFG_PARAM_CHECKING_ENABLE”.	Selects whether or not parameter checking takes place. (0): Checking disabled, (1): Checking enabled The default value is “BSP_CFG_PARAM_CHECKING_ENABLE”. To enable the checking function for the SDSI FIT module only, set this definition to 1. To disable checking, set this definition to 0.
#define SDSI_CFG_CHx_INCLUDED Note: The default value of channel 0 is “enabled”. The channel number is represented by “x”.	Selects whether or not the specified channel is used. When this option is set to “disabled”, code for processing the specified channel is omitted. When this option is set to “enabled”, code for processing the specified channel is included.
#define SDSI_CFG_LONGQ_ENABLE Note: The default value is “disabled”.	Selects whether or not debug error log acquisition processing is used. When this option is set to “disabled”, code for the relevant processing is omitted. When this option is set to “enabled”, code for the relevant processing is included. To use this functionality, the LONGQ FIT module is also required.
#define SDSI_CFG_CHx_INT_LEVEL Note: The default value for channel 0 is “5” The channel number is represented by “x”.	Set SDSI interrupt priority level. SDSI interrupt may be assigned to the group interrupt depending on the MCUs. If such is the case, define the group interrupt priority level.
#define SDSI_CFG_DISABLE_SYSTEM_INTERRUPT Note: The default value is “disabled”.	Selects whether or not to disable all processors interrupt during R_SDSI_Open() processing. If enabled, it detects SDIO command issued in uninitialized state of the driver (communication disabled) and decreases the possibility of returning the response. Disabled : does not disable all processors interrupt Enabled : Disables all processors interrupt When interrupts are disabled, the I flag in the processor stratus word (PSW) register of the CPU is cleared. Therefore, when enabling this definition, set CPU to supervisor mode. In user mode, privileged instruction exception is generated by interrupt disable processing.
#define SDSI_CFG_FBR_ADR_100H Note: The default value is “0x00”.	Set FBR 0x100 Function1 Standard SDIO Function interface code. The value set to b7-b4 is ignored.
#define SDSI_CFG_FBR_ADR_101H Note: The default value is “0x01”.	Set FBR 0x101 Function1 Extended standard SDIO Function interface code.

Configuration options in *r_sdsi_rx_config.h*	
#define SDSI_CFG_FBR_ADR_102H	Setting FBR 0x102 is prohibited. If a setting is made, it is ignored. Set the SPS bit in FBR 0x102 by means of the macro definition SDSI_CFG_FBR_SPS_BIT.
#define SDSI_CFG_FBR_ADR_103H Note: The default value is “0x00”.	Set FBR 0x103 Function1 Standard iSDIO Function Interface Code.
#define SDSI_CFG_FBR_ADR_104H Note: The default value is “0x00”.	Set FBR 0x104 Function1 MID_MANF SDIO Card Manufacturer Code.
#define SDSI_CFG_FBR_ADR_105H Note: The default value is ”0x00”.	Set FBR 0x105 Function1 MID_MANF SDIO Card Manufacturer Code.
#define SDSI_CFG_FBR_ADR_106H Note: The default value is “0x00”.	Set FBR 0x106 Function1 MID_CARD Manufacturer Information.
#define SDSI_CFG_FBR_ADR_107H Note: The default value is “0x00”.	Set FBR 0x107 Function1 MID_CARD Manufacturer Information.
#define SDSI_CFG_FBR_ADR_108H Note: The default value is “0x00”.	Set FBR 0x108 I/O block size for Function1.
#define SDSI_CFG_FBR_SPS_BIT Note: The default value is “0”.	Set FBR 0x102 SPS bit value (0 or 1).
#define SDSI_CFG_CCCR_SMPC_BIT Note: The default value is “0”.	Set CCCR 0x012 SMPC bit value (0 or 1).

2.8 Code Sizes

Table 2.2 shows the code size when the latest version of the module is used.

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

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

Module Revision: r_sdsi_rx rev2.02

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

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

GCC for Renesas RX 4.8.4.201803

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

IAR C/C++ Compiler for Renesas RX version 4.10.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

Table 2.2 Code Sizes

ROM, RAM and Stack Code Sizes (Note1, 2, 3)							
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
RX65N	ROM	**2349 bytes**	**1830 bytes**	**4692 bytes**	**3796 bytes**	**3864 bytes**	**3095 bytes**
	RAM	**8 bytes**		**4 bytes**		**12 bytes**	
	Max. user stack	**56 bytes**		**-**		**108 bytes**	
	Max. interrupt stack	**0 bytes**		**-**		**68 bytes**	

Note 1 Under confirmation conditions listed the following

- r_sdsi_rx.c
- r_sdsi_dev.c
- r_sdsi_register.c

Note 2 The required memory sizes differ according to the C compiler version and the compile conditions.

Note 3 The memory sizes listed apply when the little endian. The above memory sizes also differ according to endian mode.

2.9 Arguments

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

```
typedef struct
{
    uint32_t    reg_no;
    uint32_t    offset;
    uint32_t    * p_buff;
} sdsi_reg_t;

typedef struct
{
    uint32_t    adr;
    uint32_t    mode;
} sdsi_direct_trans_t;

typedef struct
{
    uint32_t    adr;
    uint16_t    blkcnt;
    uint16_t    bytcnt;
    uint8_t     sdcmdcr;
    uint8_t     cmd;
    uint8_t     rsv[2];
} sdsi_cmd_t;
```

2.10 Return Values

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

```
typedef enum e_sdsi_status
{
    SDSI_SUCCESS                =  0,
    SDSI_ERR                    =  -1,
    SDSI_ERR_BUSY               =  -2,
    SDSI_ERR_ADDRESS_BOUNDARY =  -3
} sdsi_status_t;
```

2.11 Callback Function

lists the callback function of the SDSI FIT module.

Table 2.3 Callback Function

Function to register the callback function	Timing to call the callback function
R_SDSI_RegistIntCallback()	Detected SDSI command interrupt[2]
R_SDSI_RegistCdCallback()	Detected card detection disable (rise/fall) interrupt
R_SDSI_RegistDtCallback()	Detected DMA transfer end interrupt

2.12 Adding FIT Modules to Projects

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

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

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

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

[2] SDSI command interrupt are "CMD53 read command interrupt", "CMD53 write command interrupt" and "CMD52 write command interrupt".

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_SDSI_Open()

This function initializes the SDSI FIT module. This function should be run first before using other API functions.

Format

```
sdsi_status_t  R_SDSI_Open(
   uint32_t channel,
   void * p_sdsi_workarea
)
```

Parameters

channel

SDSI channel number

** p_sdsi_workarea*

Working area pointer for 4-byte boundaries (working area size 28 bytes should be secured)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*
SDSI_ERR_ADDRESS_BOUNDARY	*4-byte boundaries address error in *p_sdsi_workarea*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Gets SDSI channel resource specified by the argument channel, and initializes SDSI driver and SDSI channels. Also, this function monopolizes the SDSI channel resource.

Hold the working area specified by *p_sdsi_workarea and do not change the contents until SDSI driver's R_SDSI_Close() is called.

Reentrant

Reentrancy from a different channel is possible.

Example

```
uint32_t    g_sdsi_work[28/sizeof(uint32_t)];

if (R_SDSI_Open(SDSI_CH0, &g_sdsi_work[0]) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

To end the function processing before SDIO command is issued from SD host, run this function immediately after the system power-up.
It is recommended to enable #define SDSI_CFG_DISABLE_SYSTEM_INTERRUPT. When running this function, the state becomes IOR0=1 (Function0 Enabled/ Ready state) during the period from cancelation of SDSI module stop state to SDSI software reset (SDSICR3.SRST). During this period, SD slave detects SDIO command and returns the response. If #define SDSI_CFG_DISABLE_SYSTEM_INTERRUPT is enabled, all processors interrupt requests are disabled during the period from module stop state to software reset, therefore, ready state period can be minimized.
The pin state does not change before/after running this function.
APIs other than R_SDSI_GetVersion() function, R_SDSI_Log() function, and R_SDSI_Set_LogHdlAddress() function cannot be used unless the function is successfully completed.

R_SDSI_Close()

This function is used to release the resources of the SDSI FIT module currently in use.

Format

```
sdsi_status_t  R_SDSI_Close(
   uint32_t channel
)
```

Parameters

channel

SDSI channel number

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Ends all SDSI FIT module processing and releases SDSI channel resource specified by the argument channel. Sets the SDSI channel to the module stop state.

Releases the working area specified by R_SDSI_Open() function.

Reentrant

Reentrancy from a different channel is possible.

Example

```
if (R_SDSI_Close(SDSI_CH0) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Open processing by R_SDSI_Open() function is also required before running the function. The pin state does not change before/after running this function.

R_SDSI_Initialize()

This function performs SDSI IP initial setting. After successful operation, the state changes to C flag polling state.

Format

```
sdsi_status_t  R_SDSI_Initialize(
   uint32_t channel
)
```

Parameters

channel

SDSI channel number

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Performs SDSI IP initial setting. After successful operation, the state changes to C flag polling state.

Reentrant

Reentrancy from a different channel is possible.

Example

```
sdsi_status_t ret = SDSI_SUCCESS;

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

if (R_SDSI_Initialize(SDSI_CH0) != SDSI_SUCCESS)
{
      /* Error */
}

/* ==== C flag polling ==== */
do
{
      ret = R_SDSI_CflagPolling(SDSI_CH0);
      if (SDSI_ERR == ret)
      {
            /* Error */
      }
}
while (SDSI_ERR_BUSY == ret);
```

Special Notes

Before running this function, pin setting is required. Refer to “4 Pin Setting”. Open processing by R_SDSI_Open() function is also required before running the function.
The following settings should be made :

- Enable CPU access to Function1 Register1-4.
- Enable SD host access on Function1 Register5.
- Set FBR, FBR.SPS, and CCCR.SMPC
- Enable SDSI interrupt.
- Set CCCR.IOR1 to “1 (Ready)”.
- Set I/O Function ready 0 bit (SDSICR3.IOR0) to “1”. When the bit is “1” and CMD5 from the SD host is accepted, ”1” is set to C flag on R4 response. C flag status can be confirmed with R_SDSI_CflagPolling() return value. Call R_SDSI_CflagPolling() until CMD5 is issued from SD host.

R_SDSI_End()

This function changes the SDSI FIT module from idle state to initialization state.

Format

```
sdsi_status_t  R_SDSI_End(
   uint32_t channel
)
```

Parameters

channel

SDSI channel number

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Performs SDSI end processing.

Reentrant

Reentrancy from a different channel is possible.

Example

```
if (R_SDSI_End(SDSI_CH0) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() function is required. The pin state does not change before/after running this function.

R_SDSI_CflagPolling()

This function gets C flag status of R4 response.

After initialization processing by the R_SDSI_Initialize() function, call this function and check that SDSI_SUCCESS (C flag is "1 (ready)") as return value.

Format

```
sdsi_status_t  R_SDSI_CflagPolling(
   uint32_t channel
)
```

Parameters

channel

SDSI channel number

Return Values

SDSI_SUCCESS	*C Flag is “1(Ready)”*
SDSI_ERR_BUSY	*C Flag is “0(Busy)”*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Gets C flag status of R4 response.

Reentrant

Reentrancy from a different channel is possible.

Example

```
sdsi_status_t ret = SDSI_SUCCESS;

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

if (R_SDSI_Initialize(SDSI_CH0) != SDSI_SUCCESS)
{
      /* Error */
}

/* ==== C flag polling ==== */
do
{
      ret = R_SDSI_CflagPolling(SDSI_CH0);
      if (SDSI_ERR == ret)
      {
            /* Error */
      }
}
while (SDSI_ERR_BUSY == ret);
```

Special Notes

Before running this function, open processing by R_SDSI_Initialize() is required.

When IOR0 is 1 and CMD5 from the SD host is accepted, ”1” is set to C flag on R4 response.

R_SDSI_WriteCisReg()

This function writes the value to CIS Data Register.

Format

```
sdsi_status_t  R_SDSI_WriteCisReg(
   uint32_t channel,
   sdsi_reg_t * p_sdsi_reg
)
```

Parameters

channel

SDSI channel number

** p_sdsi_reg*

reg_no	*Register No. (Setting is not required)*
offset	*CIS Data Register offset (multiples of 4:0,4,8,12...100,104)*
** p_buff*	*Write buffer pointer (4 bytes)*

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Writes the value to CIS Data Register, and accesses to CIS Data Register at 4 bytes. Can call only when the driver is in the initialization state.

Reentrant

Reentrancy from a different channel is possible.

Example

```
typedef union
{
      uint32_t l;
      uint8_t c[4];
} sdsi_union_t;

sdsi_reg_t sdsi_reg;
sdsi_union_t io_buff = { 0 };

io_buff.c[0] = 0x20;
io_buff.c[1] = 0x04;
io_buff.c[2] = 0x00;
io_buff.c[3] = 0x20;

sdsi_reg.offset = 0;
sdsi_reg.p_buff = &io_buff.l;
if (R_SDSI_WriteCisReg(SDSI_CH0, &sdsi_reg) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

R_SDSI_ReadCisReg()

This function reads the value from CIS Data Register.

Format

```
sdsi_status_t  R_SDSI_ReadCisReg(
   uint32_t channel,
   sdsi_reg_t * p_sdsi_reg
)
```

Parameters

channel

SDSI channel number

** p_sdsi_reg*

reg_no	*Register No. (Setting is not required)*
offset	*CIS Data Register Offset (multiples of 4:0,4,8,12...100,104)*
** p_buff*	*Read buffer pointer (4 bytes)*

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Reads the value from CIS Data Register, and accesses CIS Data Register at 4 bytes.

Reentrant

Reentrancy from a different channel is possible.

Example

```
typedef union
{
      uint32_t l;
      uint8_t c[4];
} sdsi_union_t;

sdsi_reg_t sdsi_reg;
sdsi_union_t io_buff = { 0 };

io_buff.l = 0;

sdsi_reg.offset = 0;
sdsi_reg.p_buff = &io_buff.l;
if (R_SDSI_ReadCisReg(SDSI_CH0, &sdsi_reg) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

R_SDSI_WriteFuncReg()

This function writes the value to FN1 Data Register n (n=1,2,5). It performs processing to write data to the Function1 area.

Format

```
sdsi_status_t  R_SDSI_WriteFuncReg(
   uint32_t channel,
   sdsi_reg_t * p_sdsi_reg
)
```

Parameters

channel

SDSI channel number

** p_sdsi_reg*

reg_no *Register No. (1 or 2 or 5)*

offset *FN1 Data Register n (n=1 or 2 or 5) Offset*

<Allowable setting value>

FN1 Data Register 1 (Function1 Register1) : 0,1,2,3...255

FN1 Data Register 2 (Function1 Register2) : 0,1,2,3...255

FN1 Data Register 5 (Function1 Register5) : 0,1,2,3...1023

** p_buff* *Write buffer pointer (1 bytes)*

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Writes the value to FN1 Data Register n, and accesses FN1 Data Register n at 4 bytes.

Reentrant

Reentrancy from a different channel is possible.

Example

```
sdsi_reg_t sdsi_reg;
uint8_t io_buff = 0;

sdsi_reg.reg_no = SDSI_FUNC1_REG1;
sdsi_reg.offset = 0x0;
sdsi_reg.p_buff = &io_buff;
if (R_SDSI_WriteFuncReg(SDSI_CH0, &sdsi_reg) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

FN1 Data Register 5 cannot be accessed simultaneously by the SD host and the CPU. Therefore, follow the steps below:

1. Enable FN1 Data Register 5 to access from CPU (SDSICR2.REG5EN = 1)

   Access by SD host controller disabled.
2. Access FN1 Data Register 5
3. Enable FN1 Data Register 5 to access from SD host controller (SDSICR2.REG5EN = 0)

   Access by SD host controller enabled.

During the period of 2 mentioned above, when accessing from SD host controller to FN1 Data Register 5, the value wrote is ignored, and the read value is undefined. If writing or reading from both CPU and SD host controller to FN1 Data Register 5 occurs, exclusive access control in FN1 Data Register 5 is required.

R_SDSI_ReadFuncReg()

This function reads the value from FN1 Data Register m (m=1,3,5). It performs processing to read data from the Function1 area.

Format

```
sdsi_status_t  R_SDSI_ReadFuncReg(
   uint32_t channel,
   sdsi_reg_t * p_sdsi_reg
)
```

Parameters

channel

SDSI channel number

** p_sdsi_reg*

reg_no *Register No. (1 or 3 or 5)*

offset *FN1 Data Register m (m=1 or 2 or 5) Offset*

<Allowable setting value>

FN1 Data Register 1 (Function1 Register1) : 0,1,2,3...255

FN1 Data Register 3 (Function1 Register3) : 0,1,2,3...255

FN1 Data Register 5 (Function1 Register5) : 0, 1,2,3...1023

** p_buff* *Read buffer pointer (1 bytes)*

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Reads the value from FN1 Data Register m, and accesses FN1 Data Register m at 4 bytes.

Reentrant

Reentrancy from a different channel is possible.

Example

```
sdsi_reg_t sdsi_reg;
uint8_t io_buff = 0

sdsi_reg.reg_no = SDSI_FUNC1_REG1;
sdsi_reg.offset = 0x0;
sdsi_reg.p_buff = &io_buff;
if (R_SDSI_ReadFuncReg(SDSI_CH0, &sdsi_reg) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

FN1 Data Register 5 cannot be accessed simultaneously by the SD host and the CPU. Therefore, follow the steps below:

1. Enable FN1 Data Register 5 to access from CPU (SDSICR2.REG5EN = 1)

   Access by SD host controller disabled.
2. Access FN1 Data Register 5
3. Enable FN1 Data Register 5 to access from SD host controller (SDSICR2.REG5EN = 0)

   Access by SD host controller enabled.

During the period of 2 mentioned above, when accessing from SD host controller to FN1 Data Register 5, the value wrote is ignored, and the read value is undefined. If writing or reading from both CPU and SD host controller to FN1 Data Register 5 occurs, exclusive access control in FN1 Data Register 5 is required.

R_SDSI_WriteIntVectorReg()

This function writes the value to FN1 interrupt vector register (FN1INTVECR).

Format

```
sdsi_status_t  R_SDSI_WriteIntVectorReg(
   uint32_t channel,
   uint8_t * vector
)
```

Parameters

channel

SDSI channel number

vector

Value of FN1 Interrupt Vector Register (1 byte)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

writes the value to FN1 interrupt vector register. If CCCR IEN1 is set to ”1”, SDIO interrupt using SDSI_D1 occurs.

Reentrant

Reentrancy from a different channel is possible.

Example

```
if (R_SDSI_WriteIntVectorReg(SDSI_CH0, 0xff) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required. If SDIO interrupt occurs, SDSI_D1 changes from H to L.

The timing of issuance of SDIO interrupts differs depending on whether the SD bus width is 1-bit or 4-bit.

- 1-bit bus (CCCR 0x07 bus width = 00b)

  SDIO interrupts are generated with asynchronous timing; they are not synchronized with the SD clock.

- 4-bit bus (CCCR 0x07 bus width = 10b)

  SDIO interrupts are generated in synchronization with the SD clock. If this API function is called when the SD clock is halted, no SDIO interrupt is generated. SDIO interrupts are generated when the SD clock is being supplied.

  To generate SDIO interrupts with asynchronous timing, first set the bus width to 1-bit, then call this API function.

R_SDSI_ReadIntVectorReg()

This function reads the value from FN1 interrupt vector register (FN1INTVECR).

Format

```
sdsi_status_t  R_SDSI_ReadIntVectorReg(
   uint32_t channel,
   uint8_t * p_vector
)
```

Parameters

channel

SDSI channel number

** p_vector*

Read buffer pointer (1 byte)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Reads the value from FN1 interrupt vector register.

Reentrant

Reentrancy from a different channel is possible.

Example

```
uint8_t     vector = 0;

if (R_SDSI_ReadIntVectorReg(SDSI_CH0, &vector) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

R_SDSI_ReadIntClearReg()

This function reads the value from FN1 interrupt clear register.

Format

```
sdsi_status_t  R_SDSI_ReadIntClearReg(
   uint32_t channel,
   uint8_t * p_vector
)
```

Parameters

channel

SDSI channel number

** p_vector*

Read buffer pointer (1 byte)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Reads the value from FN1 interrupt clear register.

Reentrant

Reentrancy from a different channel is possible.

Example

```
uint8_t     vector = 0;

if (R_SDSI_ReadIntClearReg(SDSI_CH0, &vector) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

R_SDSI_EnableDirectTrans()

This function makes DMA transfer enable setting.

Format

```
sdsi_status_t  R_SDSI_EnableDirectTrans(
   uint32_t channel,
   sdsi_direct_trans_t * p_sdsi_direct_trans
)
```

Parameters

channel

SDSI channel number

** p_sdsi_direct_trans*

adr *DMA transfer start address (allowable setting range:On-chip RAM address)*

Use an address aligned with a 4-byte boundary.

mode *DMA transfer mode*

<Address setting:Specify one from the followings>

SDSI_MODE_DIRECT_ADDR_FIXED : Fix the DMA transfer address

SDSI_MODE_DIRECT_ADDR_INC : Increment the DMA transfer address

Specifies the next DMA transfer address when detecting DMA transfer end interrupt.

<Bus setting : Select one from the followings>

SDSI_MODE_DIRECT_BUS_LOCK : Lock the bus used in the DMA transfer

SDSI_MODE_DIRECT_BUS_UNLOCK : Does not lock the bus used in the DMA transfer

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*
SDSI_ERR_ADDRESS_BOUNDARY	*adr 4-byte boundary address error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Makes DMA transfer enable setting. After successful operation, when CMD53 (specified Function1) is issued from SD host controller, SDSI IP performs data transfer for On-chip RAM.

Reentrant

Reentrancy from a different channel is possible.

Example

```
uint32_t    g_io_buff[512/sizeof(uint32_t)];
sdsi_direct_trans_t     sdsi_direct_trans;

sdsi_direct_trans.adr = &g_io_buff[0];
sdsi_direct_trans.mode = (SDSI_MODE_DIRECT_ADDR_INC |
SDSI_MODE_DIRECT_BUS_UNLOCK);
if (R_SDSI_EnableDirectTrans(SDSI_CH0, &sdsi_direct_trans) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

Do not call this function again if this function was operated successfully. If called, error is returned. To run this function again, run DMA transfer disable processing R_SDSI_DisableDirectTrans() in advance.

SDSI performs DMA transfer using DMA bus. For RX65N/RX651 sharing the DMA bus used in SDSI and Ethernet controller (ETHERC), exclusive access control is required. The preconditions necessary for this function to operate properly are listed below. If these preconditions are not satisfied, the value *SDSI_ERR* is returned.

1. The ETHER and EDMAC is in the module stop state.
2. Initial settings have not been applied to the Ethernet FIT module.
   (The ETHERC and EDMAC hardware resource is free.)

And ETHERC cannot release DMA bus dynamically due to its communication method. Therefore, if combining use of SDSI and ETHERC in the user system, do not perform SDSI DMA transfer. For the setting, follow the procedures below (assuming the use of Ethernet FIT module to control ETHERC)

1. Make initialization setting for Ethernet FIT module

   (ETHERC/EDMAC hardware resource lock and module stop cancellation)
2. Make initialization setting for SDSI FIT module
3. Disable R_SDSI_EnableDirectTrans() call after the above procedures

When this function is called in the state mentioned in 3 above, *SDSI_ERR* is returned.

R_SDSI_DisableDirectTrans()

This function makes DMA transfer disable setting.

Format

```
sdsi_status_t  R_SDSI_DisableDirectTrans(
   uint32_t channel
)
```

Parameters

channel

SDSI channel number

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Makes DMA transfer disable setting. After successful operation, when CMD53 (specified Function1) is issued from SD host controller, SDSI IP performs data transfer for Function1 area.

Reentrant

Reentrancy from a different channel is possible.

Example

```
if (R_SDSI_DisableDirectTrans(SDSI_CH0) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

R_SDSI_SetDirectTransAdr()

This function specifies the DMA transfer address.

Format

```
sdsi_status_t  R_SDSI_SetDirectTransAdr(
   uint32_t channel,
   uint32_t adr
)
```

Parameters

channel

SDSI channel number

adr

DMA transfer start address (allowable setting range : On-chip RAM address)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Specifies the DMA transfer address.

Reentrant

Reentrancy from a different channel is possible.

Example

```
uint8_t     g_io_buff[512];

if (R_SDSI_SetDirectTransAdr(SDSI_CH0, &g_io_buff[0]) != SDSI_SUCCESS)
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required. Call this function before the DMA transfer starts.

R_SDSI_GetDirectTransAdr()

This function gets DMA transfer address.

Format

```
sdsi_status_t  R_SDSI_GetDirectTransAdr(
   uint32_t channel,
   uint32_t * p_adr
)
```

Parameters

channel

SDSI channel number

** p_adr*

DMA transfer start address buffer (4 bytes)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Gets DMA transfer address.

Reentrant

Reentrancy from a different channel is possible.

Example

```
uint32_t    adr = 0;

if (R_SDSI_GetDirectTransAdr(SDSI_CH0, &adr))
{
      /* Error */
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required. Call this function before the DMA transfer starts.

R_SDSI_RegistIntCallback()

This function registers SDSI command interrupt[3] callback function.

Format

```
sdsi_status_t  R_SDSI_RegistIntCallback(
   uint32_t channel,
   sdsi_status_t (* callback)(sdsi_cmd_t *)
)
```

Parameters

channel

SDSI channel number

(callback)(sdsi_cmd_t *)*

Callback function to be registered

Callback function is not registered when null pointer is set.

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Registers SDSI command interrupt callback function. Call this function before running R_SDSI_Initialize().

Reentrant

Reentrancy from a different channel is possible.

Example

```
sdsi_cmd_t g_sdsi_cmd;
sdsi_status_t r_sdsi_callback(sdsi_cmd_t * p_cmd);

if (R_SDSI_RegistIntCallback(SDSI_CH0, r_sdsi_callback) != SDSI_SUCCESS)
{
      /* Error */
}

static sdsi_status_t r_sdsi_callback(sdsi_cmd_t * p_cmd)
{
      g_sdsi_cmd.adr     = p_cmd->adr;
      g_sdsi_cmd.blkcnt  = p_cmd->blkcnt;
      g_sdsi_cmd.bytcnt  = p_cmd->bytcnt;
      g_sdsi_cmd.sdcmdcr = p_cmd->sdcmdcr;
      g_sdsi_cmd.cmd     = p_cmd->cmd;

      return SDSI_SUCCESS;
}
```

[3] SDSI command interrupt means “CMD53 read command interrupt”, “CMD53 write command interrupt”, and “CMD52 write command interrupt”.

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

The information stored in the callback function argument (sdsi_cmd_t *) is shown in Table 3.1 This information is overwritten by the command issued from SD host controller.
Read this information before SD host issues the next command.

Table 3.1 SDSI Command Interrupt Callback Function Augment Information

Type	Member	Name	Description		
			CMD52 Write	**CMD53 Write**	**CMD53 Read**
uint32_t	adr	SD command access address	I/O register read/write start address [Valid data: Lower 17 bits b16-b0]		
uint16_t	blkcnt	Block counter	Fixed at 0	0: Infinite 1: 1 block to 511: 511 blocks [Valid data: Lower 9 bits b8-b0]	
uint16_t	bytcnt	Byte counter	Fixed at 0	0: 0 bytes 1: 1 byte to 2048: 2048 bytes [Valid data: Lower 12 bits b11-b0] “Byte mode” Stores the byte count included in the arguments of CMD53. If the byte count is 0, a value of 512 is stored. “Block mode” Stores the block size corresponding to the function number included in the arguments of CMD53.	
uint8_t	sdcmdcr	SD command control information	Stores the value of the SD command control register (SDCMDCR).		
uint8_t	cmd	Command information	SDSI_CMD52_W(0x01)	SDSI_CMD53_W(0x02)	SDSI_CMD53_R(0x04)

Bit	Bit Name	Function
b0	SD command index	0: No command issued, or CMD52 write command issued. 1: CMD53 command issued.
b1	Transfer direction	0: Read from this module by SD host controller 1: Write to this module by SD host controller
b2	Read after SD command write	0: Read data is same as write data*1 1: Read data was read after write.
b3	SD command byte/block mode	0: Byte mode*2 1: Block mode
b4	SD command CMD53 address mode	0: Fixed transfer address 1: Incremental transfer address*2
b7-b5	Reserved bits	The value of these bits is 0.

Note 1. Undefined for CMD53.
Note 2. Undefined for CMD52.

R_SDSI_RegistCdIntCallback()

This function registers SDSI card detection disable (Rise/Fall) interrupt callback function.

Format

```
sdsi_status_t  R_SDSI_RegistCdIntCallback(
   uint32_t channel,
   sdsi_status_t (* callback)(uint32_t)
)
```

Parameters

channel

SDSI channel number

(callback)(uint32_t)*

Callback function to be registered

Callback function is not registered when null pointer is set.

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Registers SDSI card detection disable (Rise/Fall) interrupt callback function. Call this function before running R_SDSI_Initialize().

Reentrant

Reentrancy from a different channel is possible.

Example

```
sdsi_status_t r_sdsi_cd_callback(uint32_t cd);

if (R_SDSI_RegistCdIntCallback(SDSI_CH0, r_sdsi_cd_callback) != SDSI_SUCCESS)
{
      /* Error */
}

static sdsi_status_t r_sdsi_cd_callback(uint32_t cd)
{
      if (SDSI_CD_RISE == cd)
      {
            /* Card detection disable (rise) interrupt. */
            R_BSP_NOP();

      }
      else
      {
            /* Card detection disable (fall) interrupt. */
            R_BSP_NOP();
      }
      return SDSI_SUCCESS;
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

The information stored in the callback function argument (uint32_t) is as follows :

SDSI_CD_RISE : when Card detection disable (Rise) interrupt is detected

SDSI_CD_FALL: when Card detection disable (Fall) interrupt is detected

R_SDSI_RegistDtIntCallback()

This function registers SDSI DMA transfer end interrupt callback function.

Format

```
sdsi_status_t  R_SDSI_RegistDtIntCallback(
   uint32_t channel,
   sdsi_status_t (* callback)(sdsi_cmd_t *)
)
```

Parameters

channel

SDSI channel

(callback)(sdsi_cmd_t *)*

Callback function to be registered

Callback function is not registered when null pointer is set.

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Registers SDSI DMA transfer end interrupt callback function. Call this function before running R_SDSI_Initialize().

Reentrant

Reentrancy from a different channel is possible.

Example

```
sdsi_cmd_t g_sdsi_cmd;
sdsi_status_t r_sdsi_dt_callback(sdsi_cmd_t * p_cmd);

if (R_SDSI_RegistDtIntCallback(SDSI_CH0, r_sdsi_dt_callback) != SDSI_SUCCESS)
{
      /* Error */
}

static sdsi_status_t r_sdsi_dt_callback(sdsi_cmd_t * p_cmd)
{
      g_sdsi_cmd.adr     = p_cmd->adr;
      g_sdsi_cmd.blkcnt  = p_cmd->blkcnt;
      g_sdsi_cmd.bytcnt  = p_cmd->bytcnt;
      g_sdsi_cmd.sdcmdcr = p_cmd->sdcmdcr;
      g_sdsi_cmd.cmd     = p_cmd->cmd;

      return SDSI_SUCCESS;
}
```

Special Notes

Before running this function, open processing by R_SDSI_Open() is required.

The information stored in the callback function argument (sdsi_cmd_t *) is the same as SDSI command interrupt. Refer to Table 3.1 for the detail. This information is overwritten by the command issued from SD host controller. Read this information before SD host issues the next command.

R_SDSI_GetVersion()

This function is used to get the SDSI FIT module version information.

Format

```
uint32_t R_SDSI_GetVersion(
   void
)
```

Parameters

None

Return Values

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

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Returns the driver's version information.

Reentrant

Reentrancy from a different channel is possible.

Example

```
uint32_t    version = 0;

version = R_SDSI_GetVersion();
```

Special Notes

None

R_SDSI_SetLogHdlAddress()

This function specifies the handler address for the LONGQ FIT module.

Format

```
sdsi_status_t  R_SDSI_SetLogHdlAddress(
   uint32_t user_long_que
)
```

Parameters

user_long_que

LONGQ FIT module handler address

Return Values

SDSI_SUCCESS	*Successful operation*

Properties

Prototype declarations are contained in r_sdsi_rx_if.h.

Description

Specifies the handler address of the LONGQ FIT module.

Reentrant

Reentrancy from a different channel is possible.

Example

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

sdsi_status_t     ret = SDSI_SUCCESS;
uint32_t          MtlLogTbl[ERR_LOG_SIZE];
longq_err_t             err;
longq_hdl_t             p_sdsi_user_long_que;
uint32_t          long_que_hndl_address;


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

long_que_hndl_address = (uint32_t)p_sdsi_user_long_que;
ret = R_SDSI_SetLogHdlAddress(long_que_hndl_address);
```

Special Notes

Uses the LONGQ FIT module and performs preparatory processing to get the error log. Run this processing before calling R_SDSI_Open().

Incorporate the LONGQ FIT module separately.

R_SDSI_Log()

This function gets the error log.

Format

```
uint32_t R_SDSI_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 declarations are contained in r_sdsi_rx_if.h.

Description

This function gets the error log.

To end the error log acquisition, call this function.

Reentrant

Reentrancy from a different channel is possible.

Example

```
#define     USER_DRIVER_ID    (0x00000001)
#define     USER_LOG_MAX      (0x0000003f)
#define     USER_LOG_ADR_MAX  (0x00001fff)

uint8_t     io_buff[4] = {0, 0, 0, 0};
sdsi_reg_t  sdsi_reg;

sdsi_reg.reg_no = SDSI_FUNC1_REG2;
sdsi_reg.offset = 0x0;
sdsi_reg.p_buff = (uint32_t *)&io_buff[0];
if (R_SDSI_WriteFuncReg(SDSI_CH0, &sdsi_reg) != SDSI_SUCCESS)
{
      /* Set last error log to buffer. */
      R_SDSI_Log(
            USER_DRIVER_ID,
            USER_LOG_MAX,
            USER_LOG_ADR_MAX
      );
}
```

Special Notes

Incorporate the LONGQ FIT module separately.

4. Pin Setting

To use the SDSI FIT module, input/output signals of the peripheral function have to be allocated to pins with the multi-function pin controller (MPC). This pin allocation is referred to as “pin setting” in this document. Please perform the pin setting before calling the R_SDSI_Open function.

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 output according to the option selected in the Pin Setting window in the Smart configurator. Pins are configured by calling the function defined in the source file. Refer to Table 4.1 for details.

Table 4.1 Function Output by the Smart Configurator

Option Selected	Function to be Output	Remarks
Channel 0	R_SDSI_PinSet()	

4.1 Drive Capacity Control

SDSI_CMD and SDSI_D0-SDSI_D3 are input pins and perform response and data output.

Please reconsider the settings according to the circuit mounting the MCU.

The I/O port setting can be changed using the Drive Capacity Control Register (DSCR) or Drive Capacity Control Register 2 (DSCR2).

When using the pin setting function of "Smart Configurator", set the SDSI_CMD pin and the SDSI_D0-SDSI_D3 pin to DSCR = 1 (high drive output).

Refer to Table 4.2 and check the settings as necessary.

Table 4.2 Drive Capacity Control

DSCR2	DSCR	Drive Capacity	MCU default setting
0	0	Normal drive	-
0	1	High-drive output	RX65N
1	Don’t care	high-speed interface high-drive output	-

5. Demo Program

5.1 Overview of Demo Program

The demo program receives SD commands issued by the SD host and implements control of peripheral functions in response. This enables the SD host to treat the peripheral functions of the SD slave as if they were its own peripheral functions. Note that the sample program provides an example of code for SD slave processing, and that separate code for SD host processing is also necessary.

The demo program implements the following:

- Controls the GPIO FIT module and illuminates LEDs on the RSK board when designated SD commands are received.

5.2 Overview of APIs

Table 5.1 lists the API functions included in the demo program.

Table 5.1 API Functions of Demo Program

Function Name	Description
R_SDSI_PXPD_Open()	SDSI FIT module initialization processing
R_SDSI_PXPD_ReadCmd()	SD command read processing
R_SDSI_PXPD_WriteResp()	SD response write processing
R_SDSI_PXPD_SetSDIOInt()	SDIO interrupt issuance processing
R_SDSI_PXPD_GetSDIOInt()	SDIO interrupt vector read processing

5.3 Operation

5.3.1 Hardware

Figure 5-1 is a block diagram of the hardware.

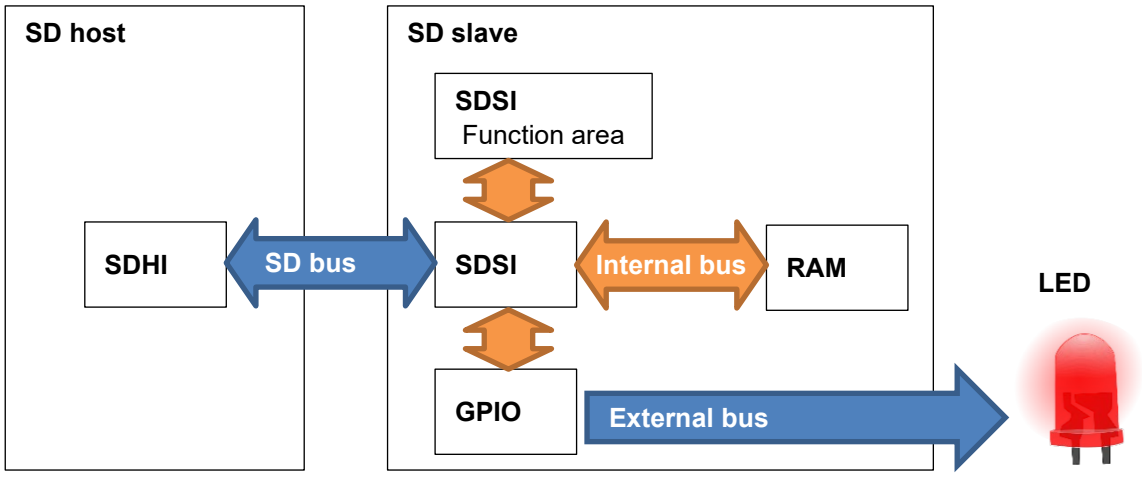


Figure 5-1 Block Diagram

5.3.2 Software

(1) State Transition Diagram

Figure 5-2 is a diagram of state transitions.

When a designated SD command is received following initialization, the software controls the GPIO FIT module and illuminates LEDs on the RSK board. When the processing to illuminate the LEDs completes, an SDIO interrupt is issued to notify the SD host that processing is finished. Upon receiving the notification, the SD host releases the SDIO interrupt of the SD slave. This causes the SD slave to transition from the SDIO interrupt cancellation wait state to the idle state.

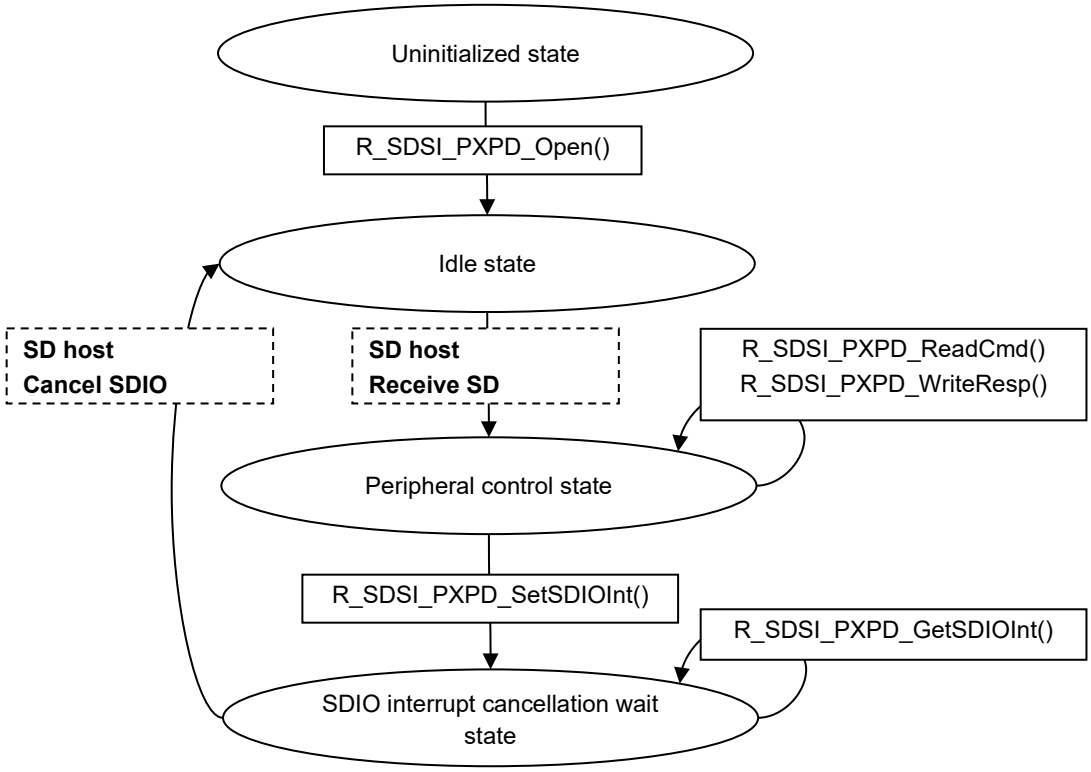


Figure 5-2 State Transition Diagram

(2) Sequence Diagram

Figure 5-3 is a sequence diagram. The SD host accesses the SD slave, which controls the GPIO and illuminates the LEDs. The details are as follows:

① The SD host takes information on the GPIO FIT module, which is controlled by the SD slave, arranges it into a packet, and transmits it to the SD slave.

② The SD slave extracts the information from the packet received from the SD host and runs the appropriate GPIO FIT module API function.

③ The SD slave issues an SDIO interrupt to notify the SD host of the API function execution result.

④ When the SD host detects the SDIO interrupt, it checks the API function execution result and cancels the SDIO interrupt.

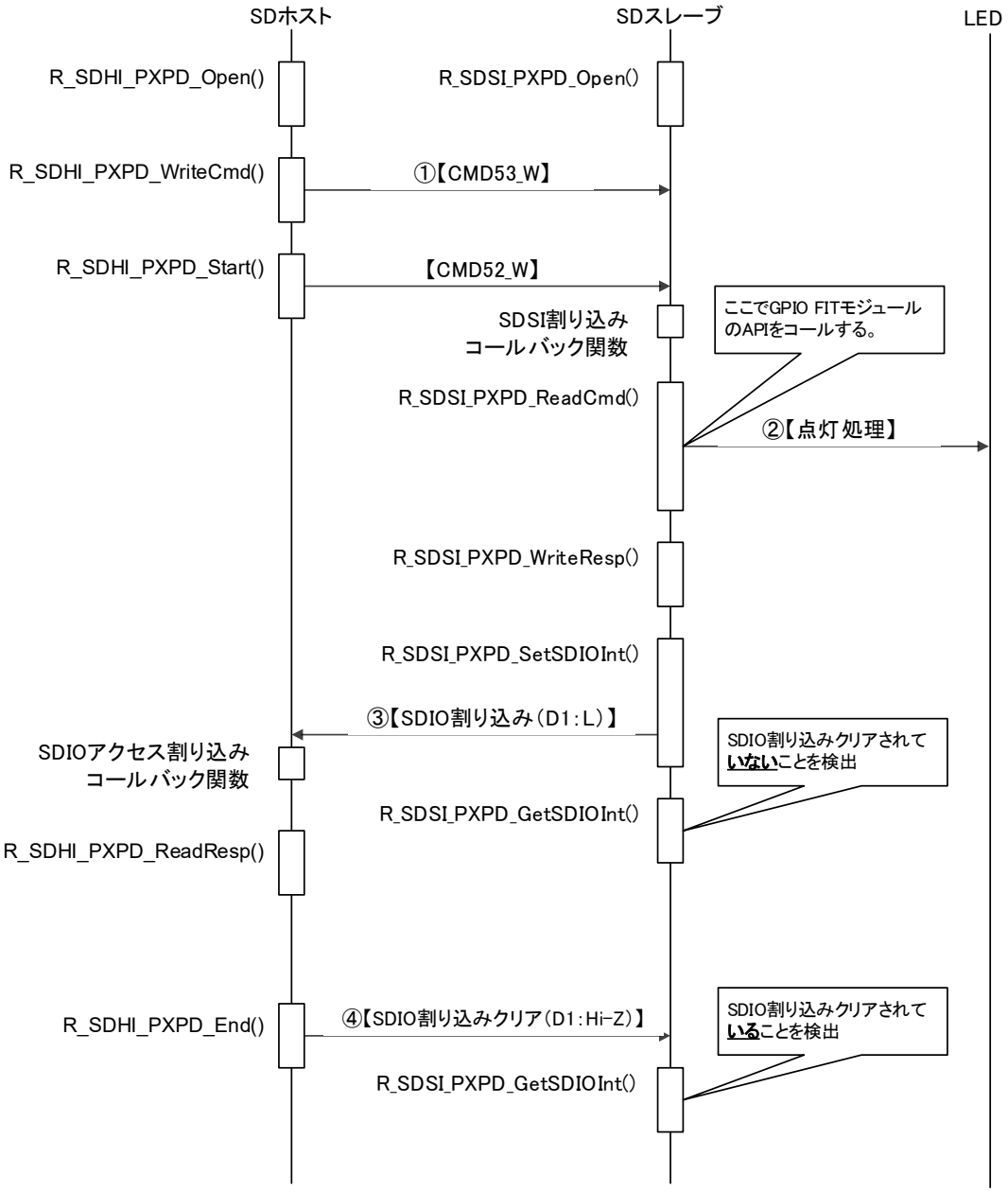


Figure 5-3 Sequence Diagram of LED Illumination under GPIO Control

(3) Discription of r_sdsi_pxpd_rx.c

Table 5.2 shows the allocation of data in the Function1 Register1 (Func1 Reg1) area. The code in r_sdsi_pxpd_rx.c references the data in Func1 Reg1, which is accessed by the SD host, and illuminates the LEDs of the SD slave. The details are described below.

- Runs R_SDSI_PXPD_Open() and waits until the value of Func1 Reg1 offset address “00h (status)” changes to “0xD0”.
- When “0xD0” is detected, calls R_SDSI_PXPD_ReadCmd() to read the information listed in Table 5.2. The sample program reads from “0x0C (function number)” to “0x03”, then runs the GPIO FIT module’s R_GPIO_PinWrite() function. This causes the LEDs on the RSK board to illuminate.
- Calls R_SDSI_PXPD_WriteResp() to update offset address “0x18 (FIT module return value)”.
- To notify the SD host when processing completes, calls R_SDSI_PXPD_SetSDIOInt() to issue an SDIO interrupt.
- The SD host cancels the SDIO interrupt, and processing ends.

Table 5.2 Register1 Allocations

Offset Address	Description	Setting Value
0x00	Status	0x00: Idle 0x01: Transfer information Written by host to Register1. 0x02: Set transfer result Written by slave to Register1. 0xD0: Execute Runs FIT module.
0x04	Function argument information valid data size (multiple of 4)	0x00: Initial value 0x04-0xE0: Settable (multiples of 4 only) Other than above: Setting prohibited
0x08	FIT module number	0x00: Not used 0x01: GPIO 0x02-0xFF: Setting invalid (for extension)
0x0C	Function number	FIT module number: 0x00 (GPIO) 0x00: R_GPIO_PortWrite() 0x01: R_GPIO_PortRead() 0x02: R_GPIO_PortDirectionSet() 0x03: R_GPIO_PinWrite() 0x04: R_GPIO_PinRead() 0x05: R_GPIO_PinDirectionSet() 0x06: R_GPIO_PinControl() 0x07: R_GPIO_GetVersion() 0x08-0xFFFFFFFF: Setting invalid
0x10	Control (SDIO interrupt request bit when processing finished)	0x00: Issue SDIO interrupt request and do not write response format at completion. 0x01: Issue SDIO interrupt request and write response format at completion. 0x02-0xFF: Setting invalid (for extension)
0x14	SDIO interrupt vector number	0x00: Initial value 0x01-0xFF: Settable
0x18-0x1B	FIT module return value	FIT module number: 0x00000001 (GPIO) 0x01000000: SDSI_PXPD_FIT_GPIO_VOID 0x01000001: SDSI_PXPD_FIT_GPIO_SUCCESS 0x01000002: SDSI_PXPD_FIT_GPIO_ERR_INVALID_MODE 0x01000003: SDSI_PXPD_FIT_GPIO_ERR_INVALID_CMD Other 0x00000000: SDSI_PXPD_VOID 0x00000001: SDSI_PXPD_SUCCESS 0xFFFFFFFF: SDSI_PXPD_ERR_FUNCTION
0x1C-0x1F	Reserved area	0x00000000-0xFFFFFFFF: Setting invalid
0x20-0xFF	Function argument information (max.: 224 bytes)	0x00000000-0xFFFFFFFF: Settable Setting values extending beyond the valid data end offset address are ignored.

(4) Discription of r_sdsi_pxpd_rx_config.h

It is a demonstration program that lights the LED.

GPIO_PORT_7_PIN_3 of RSKRX65N-2MB is the default setting. Please customize r_sdsi_pxpd_rx_config.h.

When connecting your MCU and LED, the demonstration program will operate.

1. Prepare the hardware manual of the MCU you are using.
2. Make sure the LED (lit) and the MCU port are connected.
3. Set the port output data register (PODR) of the MCU port to LED0_PODR of the header file.
4. Set the port direction register (PDR) of the MCU port to LED0_PDR of the header file.

5.4 Procedure from Adding FIT Modules to Building

The procedure for adding FIT modules to your project and building it is described below. Note that the procedure below applies to the SD slave, and that a separate environment must be created for the SD host.

- Connect the pins of the SD host and SD slave as indicated below:

-SDHI_CLK<==>SDSI_CLK

-SDHI_CMD<==>SDSI_CMD

-SDHI_D0<==>SDSI_D0

-SDHI_D1<==>SDSI_D1

-SDHI_D2<==>SDSI_D2

-SDHI_D3<==>SDSI_D3

- Create a new project in e^2 studio.
- Refer to 2.12, Adding FIT Modules to Projects, and add the following FIT modules to your project:

-r_bsp
-r_gpio_rx
-r_sdsi_rx

- Refer to 5.5, Downloading the Demo, and obtain the product package.
- Add the following demo program files, contained in the FITDemos folder of the product package, to your project:

-r_sdsi_pxpd_rx.c
-r_sdsi_pxpd_rx.h

- When settings are complete, perform the following step to build the project:

Menu [Project] > [Build Project]

- If the build does not complete successfully, refer to “6.2, Troubleshooting”, or “7, Reference Documents”.

5.5 Downloading the Demo

The demo project is not included in the RX driver package. In order to use the demo project, it is necessary to download each of the FIT modules individually. In the Application Note tab of the Smart Browser, right-click this application note and select Sample Code (Download) to download the demo project.

5.6 API Functions

5.6.1 R_SDSI_PXPD_Open()

This function initializes the SDSI FIT module. Run it before using the other API functions.

Format

```
sdsi_pxpd_status_t   R_SDSI_PXPD_Open(
   sdsi_pxpd_int_callback_info_t * p_int_callback_info
)
```

Parameters

** p_int_callback_info*

Pointer to structure for callback functions

(* callback)(sdsi_cmd_t *)

R_SDSI_RegistIntCallback () callback function

(* callback_dt)(sdsi_cmd_t *)

R_SDSI_RegistDtIntCallback() callback function

Return Values

SDSI_PXPD_SUCCESS	*Successful operation*
SDSI_PXPD_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_pxpd_rx.h.

Description

The sample program follows the steps below to call the function and initialize the SDSI FIT module.

1. Calls R_SDSI_Open() to initialize the SDSI FIT module.
2. Calls R_SDSI_RegistIntCallback() to register the SDSI command interrupt callback function.
3. Calls R_SDSI_RegistDtIntCallback() to register the SDSI DMA transfer end interrupt callback function.
4. Calls R_SDSI_WriteCisReg() and R_SDSI_ReadCisReg(), in that order, to access the CIS registers.
5. Calls R_SDSI_PinSet() to assign ports to pins.
6. Calls R_SDSI_Initialize() to make initial settings to the SDSI IP module. After a Successful operation, transitions to the C flag polling state.
7. Calls R_SDSI_CflagPolling() to get the R4 response C flag state. After initialization processing by R_SDSI_Initialize(), calls this function and confirms that the return value is SDSI_SUCCESS (C flag (ready)).

Reentrant

Reentrancy from a different channel is possible.

Example

```
ret = R_SDSI_PXPD_Open(&call);
if (SDSI_PXPD_SUCCESS != ret)
{
        trap();
}
```

Special Notes

For details of each function, refer to “3, API Functions”.

5.6.2 R_SDSI_PXPD_ReadCmd()

Performs SD command read processing.

Format

```
sdsi_pxpd_status_t   R_SDSI_PXPD_ReadCmd(
   uint32_t * p_result,
   uint8_t * p_arg_data
)
```

Parameters

** p_result*

FIT GPIO return value

** p_arg_data*

Read buffer pointer (1 byte)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_pxpd_rx.h.

Description

The sample program follows the steps below to call the function and perform SD command read processing.

1. Calls R_SDSI_ReadFuncReg() to read the value of FN1 Data Register m (m = 1, 3, or 5).
2. Operation branches based on the offset 0x00 value.

   -When offset 0x00 value is SDSI_PXPD_FIT_GPIO and offset 0x08 value is SDSI_PXPD_FIT_GPIO, R_SDSI_PXPD_ReadCmd() calls r_sdsi_pxpd_fit_gpio().

   -When offset 0x00 value is SDSI_PXPD_STATUS_DO_ENABLE_DIRECT, r_sdsi_pxpd_direct() is called with SDSI direct transfer ON.

   -When offset 0x00 value is SDSI_PXPD_STATUS_DO_DISABLE_DIRECT, r_sdsi_pxpd_direct() is called with SDSI direct transfer OFF.

Reentrant

Reentrancy from a different channel is possible.

Example

```
ret = R_SDSI_PXPD_ReadCmd(&io_buff.l, &g_sdsi_pxpd_buff[0]);
if (SDSI_PXPD_SUCCESS != ret)
{
trap();
}
```

Special Notes

None

5.6.3 R_SDSI_PXPD_WriteResp()

Performs SD response write processing.

Format

```
sdsi_pxpd_status_t R_SDSI_PXPD_WriteResp(
   uint32_t result
)
```

Parameters

result

Write buffer

Return Values

SDSI_SUCCESS	*Successful operation.*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_pxpd_rx.h.

Description

The sample program follows the steps below to call the function and perform SD response write processing.

1. Calls R_SDSI_ReadFuncReg() to read the value at offset 0x10 of FN1 Data Register 1 and confirm the command to be run.
2. If the command read is SDSI_PXPD_CTRL_SDIO_INT_WRITE, calls R_SDSI_WriteFuncReg() to set the result at offset 0x18 of FN1 Data Register 1.

Reentrant

Reentrancy from a different channel is possible.

Example

```
ret = R_SDSI_PXPD_WriteResp(io_buff.l);
if (SDSI_PXPD_SUCCESS != ret)
{
        trap();
}
```

Special Notes

None

5.6.4 R_SDSI_PXPD_SetSDIOInt()

Performs SDIO interrupt issuance processing.

Format

```
sdsi_pxpd_status_t R_SDSI_PXPD_SetSDIOInt(
            void
)
```

Parameters

void

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_pxpd_rx.h.

Description

The sample program follows the steps below to call the function and perform SD command read processing.

1. Calls R_SDSI_WriteFuncReg() to initialize offset 0x00 of FN1 data register to “0”.
2. Calls R_SDSI_ReadFuncReg() to read offset 0x14 of FN1 data register 1 in order to get the SDIO interrupt vector number.
3. Calls R_SDSI_WriteIntVectorReg() to issue an SDIO interrupt.

Reentrant

Reentrancy from a different channel is possible.

Example

```
ret = R_SDSI_PXPD_SetSDIOInt();
if (SDSI_PXPD_SUCCESS != ret)
{
        trap();
}
```

Special Notes

None

5.6.5 R_SDSI_PXPD_GetSDIOInt()

Performs SDIO interrupt vector read processing.

Format

```
sdsi_pxpd_status_t R_SDSI_PXPD_GetSDIOInt(
            uint8_t * p_vector
)
```

Parameters

** p_vector*

SDIO interrupt vector buffer(1 byte)

Return Values

SDSI_SUCCESS	*Successful operation*
SDSI_ERR	*Common error*

Properties

Prototype declarations are contained in r_sdsi_pxpd_rx.h.

Description

The sample program follows the steps below to call the function and perform SDIO interrupt vector read processing.

1. R_SDSI_ReadIntVectorReg()

Reentrant

Reentrancy from a different channel is possible.

Example

```
ret = R_SDSI_PXPD_GetSDIOInt(&io_buff.c[0]);
if (SDSI_PXPD_SUCCESS != ret)
{
    trap();
}
```

Special Notes

None

6. Appendix

6.1 Operating Environment

This section describes confirmed operation environment for the SDSI FIT module.

Table 6.1 Operation Confirmation Environment (Rev.2.00)

Item	Contents
Integrated development environment	Renesas Electronics e² studio V6.0.0
C compiler	Renesas Electronics C/C++ compiler for RX Family V.2.07.00
	Compiler options: The integrated development environment default settings are used, with the following option added. -lang = c99
Endian order	Big-endian/Little-endian
Revision of the module	Rev.2.00
Board used	Renesas Starter Kit for RX65N (product No.: RTK500565NSxxxxxxx) Renesas Starter Kit for RX65N-2MB (product No.: RTK50565N2Sxxxxxxx)

Table 6.2 Operation Confirmation Environment (Rev.2.02)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio V7.3.0 IAR Embedded Workbench for Renesas RX 4.10.01
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V.3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.08.04.201803 Compiler option: The following option is added to the default settings of the integrated development environment. -std=gnu99
	IAR C/C++ Compiler for Renesas RX version 4.10.01 Compiler option: The default settings of the integrated development environment.
Endian order	Big-endian/Little-endian
Version of the module	Ver.2.02
Board used	Renesas Starter Kit+ for RX65N (product No.: RTK500565Nxxxxxxxx)

Table 6.3 Operation Confirmation Environment (Rev.2.03)

Item	Contents
Integrated development environment	Renesas Electronics e2 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.
Endian	Big endian/little endian
Revision of the module	Rev.2.03

Table 6.4 Operation Confirmation Environment (Rev.2.04)

Item	Contents
Integrated development environment	Renesas Electronics e2 studio Version 2023-10 IAR Embedded Workbench for Renesas RX 4.20.3
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202305 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.
Endian	Big endian/little endian
Revision of the module	Rev.2.04

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_sdsi_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.

7. Reference Documents

User's Manual: Hardware
Technical Update/Technical News
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.

Technical Updates

This module reflects the contents of the following technical updates:

TN-RX*-A176A/E

Revision History

		Description	
Rev.	**Date**	**Page**	**Summary**
2.00	Jul 31, 2017	-	First edition issued.
2.02	May 20, 2019	-	Update the following compilers GCC for Renesas RX IAR C/C++ Compiler for Renesas RX
		1	Deleted R01AN1723 and R01AN1826 from Related Documents.
		1	Added Target Compilers.
		10	Added revision of dependent r_bsp module in 2.2 Software Requirements.
		13	2.8 Code Size, amended.
		39	Changed nop to BSP's built in function in Example in function 3.18 R_SDSI_RegistCdIntCallback.
		58	Added Table 6.2 Operation Confirmation Environment (Rev.2.02).
2.03	Dec 27, 2022	61	Added Table 6.3 Operation Confirmation Environment (Rev.2.03).
		Program	Updated slash format of included header file paths for Linux compatibility.
2.04	Dec 13, 2023	16, 48	Deleted the description of FIT configurator from "2.12 Adding the FIT Module to Your Project", "4. Pin Settings".
		17	Added 2.13 “for”, “while” and “do while” statements.
		62	Added Table 6.4 Operation Confirmation Environment (Rev.2.04).
		Program	Added WAIT_LOOP comments.

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/.