RZ/T1 Group

RIIC Sample Program

R01AN2596EJ0140
Rev.1.40
Jun. 07, 2018

Introduction

This application note explains a sample program that uses the RZ/T1 I^{2}C bus interface function (RIIC) to execute read/write operations on the EEPROM (R1EX24016ASAS0A) mounted on the evaluation board.

The RIIC sample program has the following features:

- Supports master transmission and reception
- Supports Fast Mode as the communication mode (maximum transfer rate: 400 kbps)

Limitations

This sample program has the following limitations:

(1) This program cannot be combined with DMA.
(2) This program does not support the RIIC timeout function.
(3) This program does not support the RIIC NACK arbitration-lost detection function.
(4) This program does not support 10-bit address transmission.
(5) This program does not support the acceptance of a restart condition as a slave device.
Do not specify the address of this module for the address immediately after a restart condition.

Target Devices

RZ/T1

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.

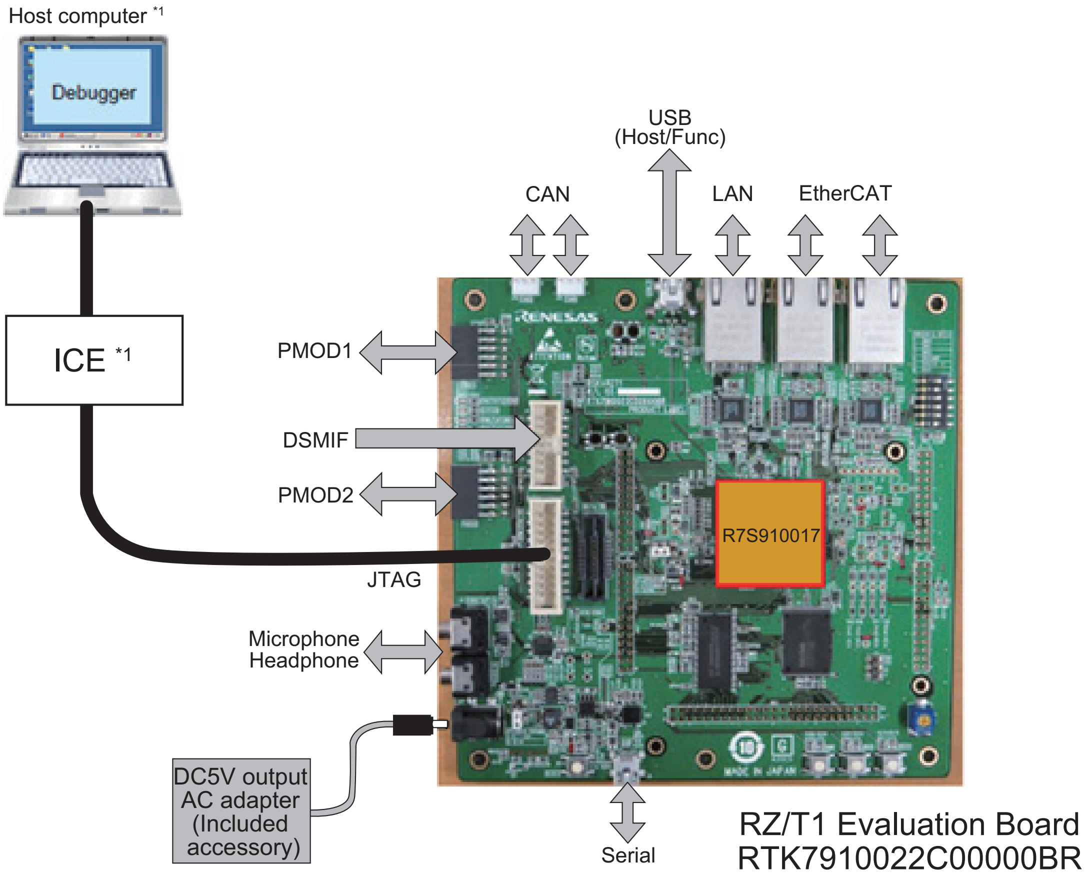
Table of Contents

1. Specifications

Table 1.1 Peripheral Functions and Applications lists the peripheral functions to be used and their applications, and Figure 1.1 shows the Operating Environment where the sample code is executed.

Table 1.1 Peripheral Functions and Applications

Peripheral Function	Application
RIIC Ch(0)	I²C communication
Power saving function	RIIC module start/stop control (MSTPCRB3)
Interrupt controller (ICUA)	RIIC interrupt control (Unit0/Unit1) Transmit end interrupt (vector 121/124) Receive end interrupt (vector 122/125) Transmission-data-empty interrupt (vector 123/126) Error detection interrupt (vector 260/261) Compare match interrupt (Unit0 ch0) Compare match interrupt (vector 21)
I/O ports (PF7, P56, P77, PA0)	LED control
Timer (CMT Unit0 ch0)	1-millisecond-interval measurement timer



Note 1. Indicates the device that the user needs to prepare.

Figure 1.1 Operating Environment

2. Operating Environment

The sample code covered in this application note is for the environment below.

Table 2.1 Operating Environment

Item	Description
Microcomputer	RZ/T1 Group
Operating frequency	CPUCLK = 450 MHz
Operating voltage	3.3 V
Integrated Development Environment	Manufactured by IAR Systems Embedded Workbench® for Arm Version 8.20.2 Manufactured by Arm DS-5™ 5.26.2 Manufactured by RENESAS e2studio 6.1.0
Operating mode	SPI boot mode 16-bit bus boot mode
Board	RZ/T1 Evaluation Board (RTK7910022C00000BR)
Device (functions to be used on the board)	• NOR flash memory (connected to CS0 and CS1 spaces) Manufacturer: Macronix International Co., Ltd. Model: MX29GL512FLT2I-10Q • SDRAM (connected to CS2 and CS3 spaces) Manufacturer: Integrated Silicon Solution Inc. Model: IS42S16320D-7TL • Serial flash memory Manufacturer: Macronix International Co., Ltd. Model: MX25L51245G • EEPROM Manufacturer: Renesas Electronics Co., Ltd. Model: R1EX24016ASAS0A • LED LED0 to LED3 (PF7, P56, P77, PA0)

3. Related Application Note

The application note related to this application note is listed below for reference.

- Application Note: RZ/T1 Group Initial Settings (R01AN2554EJ)

Note: For any registers not covered by this application note, use the values specified in the Application Note: RZ/T1 Group Initial Settings.

4. Peripheral Functions

The basics of the operating modes, I^2C bus interface (RIICa), power saving function, interrupt controller (ICUA), general I/O ports, and compare match timer (CMT) are described in the RZ/T1 Group User's Manual: Hardware.

5. Hardware

5.1 Hardware Configuration

Figure 5.1 shows the Hardware Configuration.

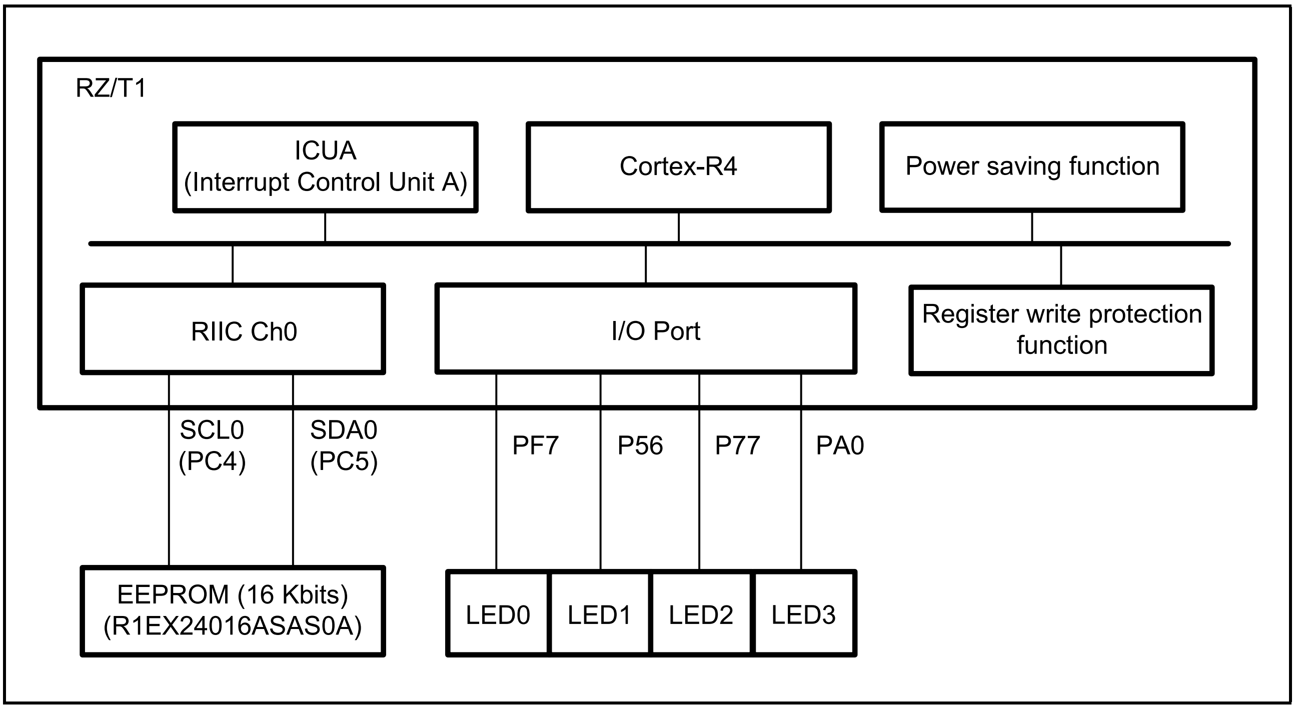


Figure 5.1 Hardware Configuration

5.2 Pins

Table 5.1 shows the Pins and Functions.

Table 5.1 Pins and Functions

Pin Name	Input/Output	Description
SCL0 (PC4)	Input/Output	I^{2}C clock line
SDA0 (PC5)	Input/Output	I^{2}C data line
PF7	Output	LED0 control
P56	Output	LED1 control
P77	Output	LED2 control
PA0	Output	LED3 control

6. Software

6.1 Operation Outline

Table 6.1 Operation Outline presents a functional overview of the RIIC sample program. Figure 6.1 shows the System Block Diagram for this program.

Table 6.1 Operation Outline

Function	Outline
Double-duty pin setup	• Sets PC4 and PC5 to SCL0 and SDA0, respectively
RIIC communication channel	• Sets to channel 0 to which EEPROM is connected
Interrupt source (interrupt priority level)	• RIIC module Transmit end interrupt (1)/receive end interrupt (1)/transmit buffer empty (1)/ error detection interrupt (1) • CMT module (for one-millisecond-interval detection) Compare match interrupt (15)
Transfer rate setup	• 400[kbps]
Operating mode	• Master transmission and reception
Operation outline	1. Back up the entire content of the EEPROM (RAM). 2. Write 0xFF to the entire EEPROM. 3. Write 0xA5 to the entire EEPROM. 4. Check the data written to the EEPROM. 5. Restore the original contents of the EEPROM. (The EEPROM is accessed (read/written) at one millisecond intervals.)
Operation result display	• LED0 lights An RIIC communication error was detected. • LED1 lights Test pattern has matched. • LED2 lights Data is being written to the EEPROM. • LED3 lights Data is being read from the EEPROM.

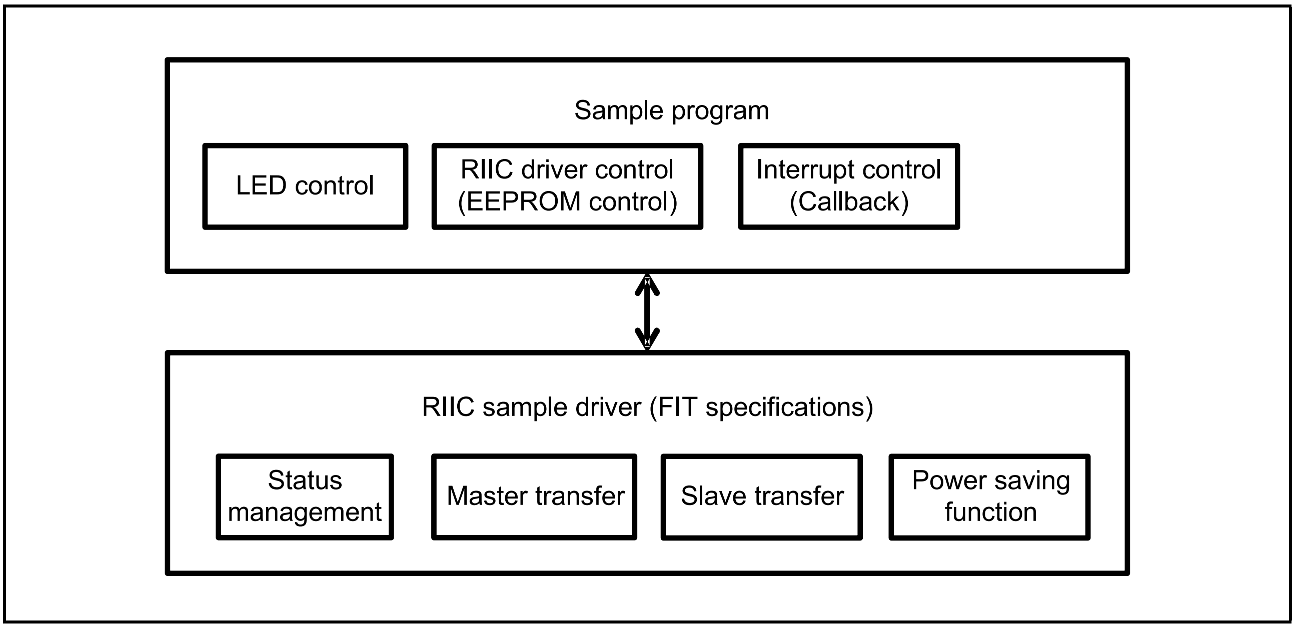

Sample program
LED control
RIIC driver control (EEPROM control)
Interrupt control (Callback)
RIIC sample driver (FIT specifications)
Status management
Master transfer
Slave transfer
Power saving function

Figure 6.1 System Block Diagram

6.1.1 Project Setup

How to set up projects used in the EWARM development environment is described in the Application Note: RZ/T1 Group Initial Settings.

6.1.2 Preparation

There is no need to prepare for executing this sample program.

6.1.3 Operation Outline of the RIIC Sample Driver

(1) State transition diagram for the RIIC sample driver

Figure 6.2 shows the State Transition Diagram for the RIIC Sample Driver.

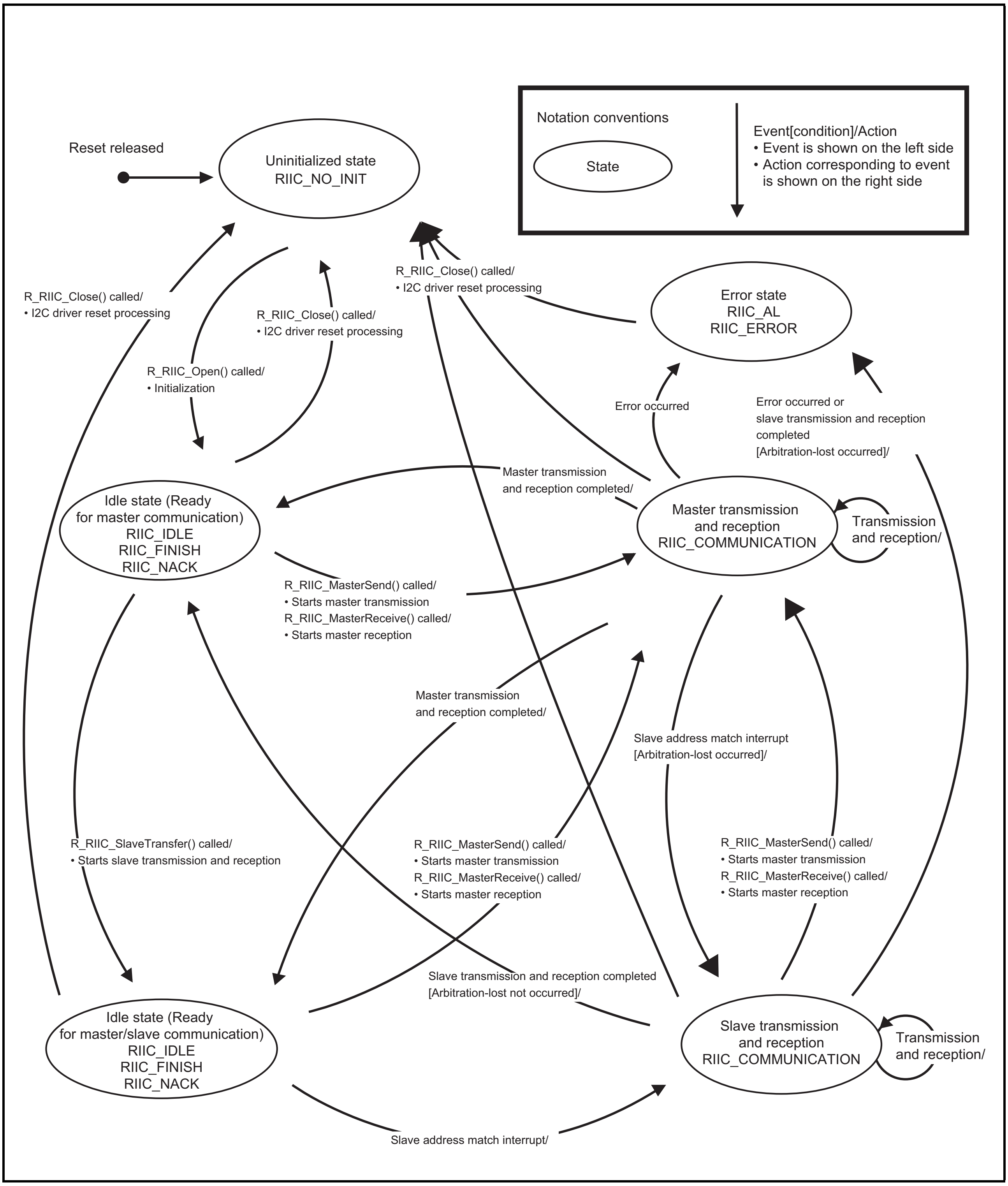

Notation conventions
State
Event[condition]/Action
• Event is shown on the left side
• Action corresponding to event is shown on the right side
Reset released
Uninitialized state
RIIC_NO_INIT
R_RIIC_Close() called/
• I2C driver reset processing
R_RIIC_Close() called/
• I2C driver reset processing
R_RIIC_Close() called/
• I2C driver reset processing
R_RIIC_Open() called/
• Initialization
Error state
RIIC_AL
RIIC_ERROR
Error occurred
Error occurred or slave transmission and reception completed [Arbitration-lost occurred]/
Master transmission and reception completed/
Idle state (Ready for master communication)
RIIC_IDLE
RIIC_FINISH
RIIC_NACK
Master transmission and reception
RIIC_COMMUNICATION
Transmission and reception/
R_RIIC_MasterSend() called/
• Starts master transmission
R_RIIC_MasterReceive() called/
• Starts master reception
Master transmission and reception completed/
Slave address match interrupt [Arbitration-lost occurred]/
R_RIIC_SlaveTransfer() called/
• Starts slave transmission and reception
R_RIIC_MasterSend() called/
• Starts master transmission
R_RIIC_MasterReceive() called/
• Starts master reception
R_RIIC_MasterSend() called/
• Starts master transmission
R_RIIC_MasterReceive() called/
• Starts master reception
Slave transmission and reception completed [Arbitration-lost not occurred]/
Idle state (Ready for master/slave communication)
RIIC_IDLE
RIIC_FINISH
RIIC_NACK
Slave transmission and reception
RIIC_COMMUNICATION
Transmission and reception/
Slave address match interrupt/

Figure 6.2 State Transition Diagram for the RIIC Sample Driver

(2) Each flag indicating the state transition of the RIIC sample driver

The I^2C communication information structure members include the device state flags (dev_sts). The device state flags store the communication states of the device. These flags can also control multiple slave devices on the same channel. Table 6.2 lists the device state flags that indicate the state transitions of the device.

Table 6.2 Device State Flags Indicating the State Transitions of the Device

State	Device State Flag (dev_sts)
Uninitialized state	RIIC_NO_INIT
Idle states	RIIC_IDLE RIIC_FINISH RIIC_NACK
Communicating state (master transmission, master reception, slave transmission, and slave reception)	RIIC_COMMUNICATION
Arbitration-lost detection state	RIIC_AL
Error state	RIIC_ERROR

(3) Arbitration-lost detection function for the RIIC sample driver

This module detects arbitration-lost states for the reasons below. This module does not support arbitration-lost detection during slave transmission, while the RIIC does.

(i) When a start condition is issued during the bus busy state
If this module issues a start condition when another master device has already issued a start condition and occupied the bus (bus busy state), this module detects an arbitration-lost state.

(ii) When this module issues a start condition after another master issued a start condition while the bus is not busy
When this module issues a start condition, it attempts to drive the SDA line low. However, if another master device issued a start condition earlier, the signal level on the SDA line does not match the signal level output by the module. Then, this module detects an arbitration-lost state.

(iii) When multiple start conditions are issued at the same time:
If multiple master devices issue start conditions at the same time, the module may determine that the start condition has been issued successfully on each device. Then, each master device starts communication, but if any of the conditions shown below is met, this module detects an arbitration-lost state.

(a) If each master device sends different data
This module compares the signal level on the SDA line with the signal level output by itself during data communications. If these signal levels do not match while data (including the slave address) is being transmitted, this module immediately detects an arbitration-lost state.

(b) If the number of data transmissions differs between each master device even though the data sent by each master device is the same
In cases other than "a" above (i.e., if each master device sends the same data and slave address), the module does not detect an arbitration-lost state. However, if the number of data transmissions differs between each master device, this module detects an arbitration-lost state.

6.2 Memory Mapping

Memory mapping for the address spaces in the RZ/T1 Group MCU and the memory in the RZ/T1 Evaluation Board is described in the Application Note: RZ/T1 Group Initial Settings.

6.2.1 Section Allocation for the Sample program

The sections used by the sample program, the section allocation for the sample program in the initial state (load view), and the section allocation for the sample program after the scatter loading function is used (execution view) are described in the Application Note: RZ/T1 Group Initial Settings.

6.2.2 MPU Setup

MPU setup is described in the Application Note: RZ/T1 Group Initial Settings.

6.2.3 Exception Processing Vector Table

Exception processing vector tables are described in the Application Note: RZ/T1 Group Initial Settings.

6.3 Interrupts

Table 6.3 shows the interrupts for the Sample Code.

Table 6.3 Interrupts for the Sample Code

Interrupt (Source ID)	Priority	Process Outline
unit0 communication error interrupt (EEI)	RIIC_CFG_CH0_INT_PRIORITY	Communication error/event occurrence processing (vector number: 260) • Arbitration-lost detection • NACK detection • Timeout detection • Start condition detection (including a restart condition) • Stop condition detection
unit0 receive data full interrupt (RXI)	RIIC_CFG_CH0_INT_PRIORITY	Receive-data-full processing (vector number: 122)
unit0 transmit data empty interrupt (TXI)	RIIC_CFG_CH0_INT_PRIORITY	Transmission-data-empty processing (vector number: 123)
unit0 transmit end interrupt (TEI)	RIIC_CFG_CH0_INT_PRIORITY	Transmit-end processing (vector number: 121)
Compare match interrupt (CMI0)	ICU_PRIORITY_15	1-millisecond-interval measurement processing (vector number: 21)

6.4 Fixed-Width Integer Types

Table 6.4 shows the Fixed-Width Integer Types for the Sample Code.

Table 6.4 Fixed-Width Integer Types for the Sample Code

Symbol	Description
int8_t	8-bit signed integer (defined in the standard library)
int16_t	16-bit signed integer (defined in the standard library)
int32_t	32-bit signed integer (defined in the standard library)
int64_t	64-bit signed integer (defined in the standard library)
uint8_t	8-bit unsigned integer (defined in the standard library)
uint16_t	16-bit unsigned integer (defined in the standard library)
uint32_t	32-bit unsigned integer (defined in the standard library)
uint64_t	64-bit unsigned integer (defined in the standard library)

6.5 Constants/Error Codes

Table 6.5 shows the Constants for the Sample Code, and Table 6.6 shows the Error Codes for the Sample Code.

Table 6.7 shows the constants that can be configured during compilation.

Table 6.5 Constants for the Sample Code

Constant Name	Setting Value	Description
RIIC_NO_INIT*1	0	Uninitialized state
RIIC_IDLE*1	1	Idle state
RIIC_FINISH*1	2	Idle state
RIIC_NACK*1	3	Idle state
RIIC_COMMUNICATION*1	4	Master or slave is sending or receiving data
RIIC_AL*1	5	Arbitration-lost detection state
RIIC_ERROR*1	6	Error state
RIIC_GEN_START_CON*2	(uint8_t)(0x01)	Start condition generated
RIIC_GEN_STOP_CON*2	(uint8_t)(0x02)	Stop condition generated
RIIC_GEN_RESTART_CON*2	(uint8_t)(0x04)	Restart condition generated
RIIC_GEN_SDA_HI_Z*2	(uint8_t)(0x08)	SDA pin set to high impedance
RIIC_GEN_SCL_ONESHOT*2	(uint8_t)(0x10)	One-shot output of the SCL clock
RIIC_GEN_RESET*2	(uint8_t)(0x20)	RIIC module reset
FIT_NO_PTR	(void *)0	FIT-defined NULL pointer

Note 1. Used as the values of riic_ch_dev_status_t type flags
Note 2. Used as the output patterns for R_RIIC_Control()

Table 6.6 Error Codes for the Sample Code

Constant Name	Setting Value	Description
RIIC_SUCCESS	0U	The function called successfully
RIIC_ERR_LOCK_FUNC	1U	RIIC being used by another module
RIIC_ERR_INVALID_CHAN	2U	Nonexistent channel specified
RIIC_ERR_INVALID_ARG	3U	Invalid argument specified
RIIC_ERR_NO_INIT	4U	Uninitialized state
RIIC_ERR_BUS_BUSY	5U	Bus busy
RIIC_ERR_AL	6U	Function called in an arbitration-lost detection state
RIIC_ERR_OTHER	7U	Other errors

The configuration options in this module are specified in r_riic_rx_config.h.

The following table shows the option names and describes the setting values.

Table 6.7 Options Configurable during Compilation (1 / 2)

Option Name	Description
RIIC_CFG_PARAM_CHECKING_ENABLE Note: Default value = 1	Selects whether to include parameter checking in the code. If this option is set to 0, parameter checking is omitted from the code, so that the code size can be reduced. When this is set to 0, parameter checking is omitted from the code. When this is set to 1, parameter checking is included in the code.
RIIC_CFG_PCLK_Hz Note: Default value = 75000000	Sets the frequency of the PCLK clock signals supplied to the RIIC0 module. The respective values to be set in the bit rate register and the internal reference clock selection bits are calculated according to the settings in RIIC_CFG_CH0_kBPS and RIIC_CFG_PCLK_Hz.
RIIC_CFG_CH0_INCLUDED Note: Default value = 1	Selects whether to use the channel. When not using the channel, set this option to 0. When this is set to 0, processing related to the channel is omitted from the code. When this is set to 1, processing related to the channel is included in the code.
RIIC_CFG_CH0_kBPS Note: Default value = 400	Specifies the RIIC0 communication rate. The respective values to be set in the bit rate register and the internal reference clock selection bits are calculated according to the settings in RIIC_CFG_CH0_kBPS and RIIC_CFG_PCLK_Hz. Specify a value less than or equal to 400.
RIIC_SCL_100K_UP_TIME Note: Default value = 1000E-9	Specifies the SCL rise time [s] when the RIIC0 communication rate is 1-100 kbps. (Specify a value in double-precision floating-point format.)
RIIC_SCL_100K_DOWN_TIME Note: Default value = 300E-9	Specifies the SCL fall time [s] when the RIIC0 communication rate is 1-100 kbps. (Specify a value in double-precision floating-point format.)
RIIC_SCL_400K_UP_TIME Note: Default value = 175E-9	Specifies the SCL rise time [s] when the RIIC0 communication rate is 101-400 kbps. (Specify a value in double-precision floating-point format.)
RIIC_SCL_400K_DOWN_TIME Note: Default value = 175E-9	Specifies the SCL fall time [s] when the RIIC0 communication rate is 101-400 kbps. (Specify a value in double-precision floating-point format.)
RIIC_CFG_CH0_DIGITAL_FILTER Note: Default value = 2	Selects the number of noise filter stages. When this is set to 0, the noise filter is disabled. When this is set to a value from 1 to 4, the values to enable the selected number of filter stages are selected for the noise filter stage selection bits and digital noise filter circuit enable bits.
RIIC_CFG_CH0_SCL0 Note: Default value = 1	Selects the output pin to be used for RIIC0 SCL. Processing for setting the selected pin as the SCL pin is included in the code. When this is set to 0, processing for setting the SCL pin is omitted from the code. When this is set to 1, PC4 is set as the SCL0 pin.
RIIC_CFG_CH0_SDA0 Note: Default value = 1	Selects the output pin to be used for RIIC0 SDA. Processing for setting the selected pin as the SDA pin is included in the code. When this is set to 0, processing for setting the SDA0 pin is omitted from the code. When this is set to 1, PC5 is set as the SDA0 pin.
RIIC_CFG_CH0_MASTER_MODE Note: Default value = 1	Selects whether to enable or disable the master arbitration-lost detection function for RIIC0. Set this to 1 (enabled) when using multiple masters. When this is set to 0, the master arbitration-lost detection function is disabled. When this is set to 1, the master arbitration-lost detection function is enabled.
RIIC_CFG_CH0_SLV_ADDR0_FORMAT*1 RIIC_CFG_CH0_SLV_ADDR1_FORMAT*2 RIIC_CFG_CH0_SLV_ADDR2_FORMAT*2 Note 1. Default value = 1 Note 2. Default value = 0	Selects the slave address format from 7 bits and 10 bits. When this is set to 0, the slave address is not set. When this is set to 1, the 7-bit slave address format is set. When this is set to 2, the 10-bit slave address format is set.

Table 6.7 Options Configurable during Compilation (2 / 2)

Option Name	Description
RIIC_CFG_CH0_SLV_ADDR0*1 RIIC_CFG_CH0_SLV_ADDR1*2 RIIC_CFG_CH0_SLV_ADDR2*2 Note 1. Default value = 0x0025 Note 2. Default value = 0x0000	Specifies the slave address. Effective bits of the setting value vary according to the value set in RIIC_CFG_CH0_SLV_ADDRi_FORMAT. The following numbers are the values of RIIC_CFG_CH0_SLV_ADDRi_FORMAT: 0: The setting value is ignored. 1: The lower 7 bits of the setting value take effect. 2: The lower 10 bits of the setting value take effect.
RIIC_CFG_CH0_SLV_GCA_ENABLE Note: Default value = 0	Selects whether to enable or disable the general call address. When this is set to 0: General call address is disabled. When this is set to 1: General call address is enabled.
RIIC_CFG_CH0_INT_PRIORITY Note: Default value = 1	Selects the priority levels of the communication error/event occurrence interrupt (EEI), receive data full interrupt (RXI), transmit data empty interrupt (TXI), and transmit end interrupt (TEI). Specify a value from 1 to 15.
RIIC_CFG_BUS_CHECK_COUNTER Note: Default value = 1000	Specifies the timeout counter (number of times of bus checking) when the RIIC API function performs bus checking. Specify a value less than or equal to 0xFFFFFFFF. Bus checking is performed in the following timings: • Before generating a start condition • After detecting a stop condition • After generating each condition and SCL one-shot pulses by using the RIIC control function (R_RIIC_Control function) With the bus checking, when the bus is busy, the timeout counter is decremented until the bus becomes free. When the counter reaches 0, the API determines that a timeout has occurred and returns an error ("Busy") as the return value. Note: Because the timeout counter is to prevent the bus from being locked due to a bus lock or some other means, specify a value greater than the time during which the other device holds the SCL pin low. Timeout period (ns) ≈ (1 / ICLK(Hz)) × counter value × 10

6.6 Structures, Unions, and Enumerated Types

Figure 6.3 shows the Structures, Unions, and Enumerated Types for the Sample Code.

```
/* ---- Return Value of IIC Driver API. ---- */
typedef enum
{
        RIIC_SUCCESS = 0U,              /* Successful operation                         */
        RIIC_ERR_LOCK_FUNC,             /* Lock has already been acquired by another task. */
        RIIC_ERR_INVALID_CHAN,          /* None existent channel number                 */
        RIIC_ERR_INVALID_ARG,           /* Parameter error                              */
        RIIC_ERR_NO_INIT,               /* Uninitialized state                          */
        RIIC_ERR_BUS_BUSY,              /* Channel is on communication.                 */
        RIIC_ERR_AL,                     /* Arbitration lost error                      */
        RIIC_ERR_OTHER                   /* Other error                                 */
} riic_return_t;

/* ---- IIC Channel status type. ---- */
typedef uint8_t        riic_ch_dev_status_t;

/* ---- Callback function type. ---- */
typedef void (*riic_callback)(void);    /* Callback function type                       */

/* ---- IIC Information structure type. ---- */
typedef volatile struct
{
    uint8_t                 rsv2;           /* reserved                                 */
    uint8_t                 rsv1;           /* reserved                                 */
    riic_ch_dev_status_t    dev_sts;        /* Device status flag                       */
    uint8_t                 ch_no;          /* Channel No.                              */
    riic_callback           callbackfunc;   /* Callback function                        */
    uint32_t                cnt2nd;         /* 2nd Data Counter                         */
    uint32_t                cnt1st;         /* 1st Data Counter                         */
    uint8_t *               p_data2nd;      /* Pointer for 2nd Data buffer              */
    uint8_t *               p_data1st;      /* Pointer for 1st Data buffer              */
    uint8_t *               p_slv_adr;      /* Pointer for Slave address buffer         */
} riic_info_t;

/* ---- IIC Status type. ---- */
typedef union
{
        uint32_t            LONG;
        struct
    {
            uint32_t        rsv1:20;        /* reserve                                  */
            uint32_t        AL:1;           /* Arbitration lost detection flag          */
            uint32_t        rsv2:4;         /*                                          */
            uint32_t        SCLO:1;         /* SCL pin output control status            */
            uint32_t        SDAO:1;         /* SDA pin output control status            */
            uint32_t        SCLI:1;         /* SCL pin level                            */
            uint32_t        SDAI:1;         /* SDA pin level                            */
            uint32_t        NACK:1;         /* NACK detection flag                      */
            uint32_t        rsv3:1;         /* reserve                                  */
            uint32_t        BSY:1;          /* Bus status flag                          */
        }BIT;
}riic_mcu_status_t;
```

Figure 6.3 Structures, Unions, and Enumerated Types for the Sample Code

6.7 Global Variables

Table 6.8 lists the global variables for the sample code.

Table 6.8 Global Variables

Type	Variable Name	Description	Function

riic_info_t *	gp_riic_info_m[]	I2C communication information structure (master)*1	R_RIIC_MasterSend()
			R_RIIC_MasterReceive()

riic_info_t *	gp_riic_info_s[]	I2C communication information structure (slave)*1	R_RIIC_SlaveTransfer()
riic_ch_dev_status_t	g_riic_ChStatus[]	Driver state*1	Global variable

riic_api_event_t	g_riic_api_Event[]	Event*1	R_RIIC_Open()
			R_RIIC_MasterSend()
			R_RIIC_MasterReceive()
			R_RIIC_SlaveTransfer()

riic_api_info_t	g_riic_api_Info[]	For internal management*1	–
volatile riic_callback	riic_callbackfunc_m	For internal management	–
volatile riic_callback	riic_callbackfunc_s	For internal management	–

static const riic_mtx_t	gc_riic_mtx_tbl[][]	State transition table	r_riic_rzt1.c
			R_RIIC_Open()
			R_RIIC_MasterSend()
			R_RIIC_MasterReceive()
			R_RIIC_SlaveTransfer()

static uint8_t	s_riic_backup[2048]	EEPROM backup area	main.c main()
static volatile uint32_t	wait_flag	RIIC sample driver interrupt wait flag	main.c main()
static volatile uint32_t	wait_cmt_flag	Compare match interrupt wait flag	main.c main()
static riic_info_t	riic_info	I2C communication information structure entity	main.c main()
static uint8_t	init_ram[16]	EEPROM initialization data	main.c main()

Note 1. Declared using as many arrays as there are channels

6.8 Functions

Table 6.9 shows the Functions for the sample code.

Table 6.9 Functions

Function Name	Page Number
R_RIIC_Open	20
R_RIIC_MasterSend	21
R_RIIC_MasterReceive	28
R_RIIC_SlaveTransfer	33
R_RIIC_GetStatus	38
R_RIIC_Control	39
R_RIIC_Close	41
R_RIIC_GetVersion	42
main	42

6.9 Specifications of Functions

This section presents the specifications of the functions for the sample code.

6.9.1 R_RIIC_Open

R_RIIC_Open	
Synopsis	This function is required first when using this module.
Header	r_riic_rx_if.h
Declaration	riic_return_t R_RIIC_Open(riic_info_t * p_riic_info)
Description	This function makes initial settings of the RIIC to start communications. It sets the RIIC channel specified by the argument. If the state of the channel is "uninitialized (RIIC_NO_INIT)", the following processes are performed: • Setting the state flag • Setting I/O ports • Allocating I^{2}C output ports • Releasing the stopped state of the RIIC module • Initializing the variables used by the API • Initializing the RIIC registers used for RIIC communications • Disabling the RIIC interrupts *p_riic_info This is the pointer to the I^{2}C communication information structure. Only the members of this structure that are used by this function are shown below. For details on this structure, see Figure 6.3. For the arguments where "(to be updated)" appears in the comment below, the values of these arguments are updated during the API execution. riic_ch_dev_status_t dev_sts; /* Pointer to the device state flag (to be updated) */ uint8_t ch_no; /* Channel number */
Arguments	riic_info_t * p_riic_info : Pointer to the RIIC communication information structure
Return values	RIIC_SUCCESS : Processing completed successfully RIIC_ERR_LOCK_FUNC : The API is locked by another task RIIC_ERR_INVALID_CHAN : Nonexistent channel RIIC_ERR_INVALID_ARG : Invalid argument RIIC_ERR_OTHER : An invalid event occurred in the current state
Remarks	None

Example

```
volatile riic_return_t ret;
riic_info_t iic_info_m;

iic_info_m.dev_sts = 0;
iic_info_m.ch_no = 0;

ret = R_RIIC_Open(&iic_info_m);
```

6.9.2 R_RIIC_MasterSend

R_RIIC_MasterSend

Synopsis This function is used when this module starts transmission as a master device.

Header r_riic_rx_if.h

Declaration riic_return_t R_RIIC_MasterSend(riic_info_t * p_riic_info)

Description This function starts the RIIC master transmission. The transmission is performed with the RIIC channel and transmission pattern specified by the arguments. If the state of the channel is "idle" (RIIC_IDLE, RIIC_FINISH, or RIIC_NACK), the following processes are performed:

- Setting the state flag
- Initializing the variables used by the API
- Enabling the RIIC interrupts
- Generating a start condition

*p_riic_info
This is the pointer to the I^{2}C communication information structure. The transmission pattern can be selected from four patterns by the argument settings.
See Table 6.10 for the specification method and allowable argument settings for each transmission pattern. See Figure 6.4 to Figure 6.7 for the signal waveforms of each transmission pattern.
Only the members of this structure that are used by this function are shown below. For details on this structure, see Figure 6.3.
When setting the slave address, store it without shifting 1 bit to left.
For the arguments where "(to be updated)" appears in the comment below, the values of these arguments are updated during the API execution.

```
riic_ch_dev_status_t dev_sts; /* Device state flag (to be updated) */
uint8_t ch_no; /* Channel number */
riic_callback callbackfunc; /* Callback function */
uint32_t cnt2nd; /*  Second data counter (number of bytes)(to be updated for only patterns 1
and 2) */
uint32_t cnt1st; /*  First data counter (number of bytes)(to be updated for only pattern 1) */
uint8_t * p_data2nd; /* Pointer to the second data storage buffer */
uint8_t * p_data1st; /* Pointer to the first data storage buffer */
uint8_t * p_slv_adr; /* Pointer to the slave address storage buffer */
```

Arguments riic_info_t * p_riic_info : Pointer to the RIIC communication information structure

Return values

RIIC_SUCCESS	: Processing completed successfully
RIIC_ERR_INVALID_CHAN	: Nonexistent channel
RIIC_ERR_INVALID_ARG	: Invalid argument
RIIC_ERR_NO_INIT	: Uninitialized state
RIIC_ERR_BUS_BUSY	: Bus busy
RIIC_ERR_AL	: Arbitration-lost error occurred
RIIC_ERR_OTHER	: An invalid event occurred in the current state

Remarks None

Example

```
/* for MasterSend(Pattern 1) */
#include "r_riic_rx_if.h"

void CallbackMaster(void);
void main(void);

void main(void)
{
    volatile riic_return_t ret;
    riic_info_t iic_info_m;
    uint8_t addr_eeprom[1]={0x50};
    uint8_t access_addr1[1]={0x00};
    uint8_t mst_send_data[5]={0x81,0x82,0x83,0x84,0x85};

    /* Sets IIC Information for sending pattern 1. */
    iic_info_m.dev_sts = 0;
    iic_info_m.ch_no = 0;
    iic_info_m.callbackfunc = &CallbackMaster;
    iic_info_m.cnt2nd = 3;
    iic_info_m.cnt1st = 1;
    iic_info_m.p_data2nd = mst_send_data;
    iic_info_m.p_data1st = access_addr1;
    iic_info_m.p_slv_adr = addr_eeprom;

    /* RIIC open */
    ret = R_RIIC_Open(&iic_info_m);

    /* RIIC send start */
    ret = R_RIIC_MasterSend(&iic_info_m);
    while(1);
}

void CallbackMaster(void)
{
    /* callback process */
}
```

Special Notes:

The following table lists the allowable argument settings for each transmission pattern.

Table 6.10 Allowable Argument Settings for Each Transmission Pattern

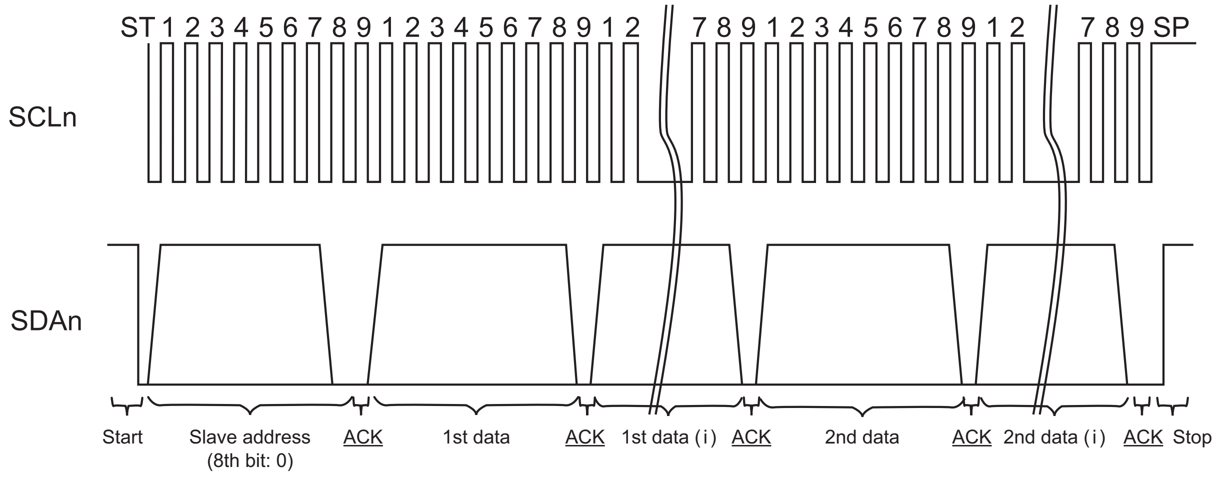
	User Settable Range			
Structure Member	**Master Transmission Pattern 1**	**Master Transmission Pattern 2**	**Master Transmission Pattern 3**	**Master Transmission Pattern 4**
*p_slv_adr	Pointer to the slave address storage buffer	Pointer to the slave address storage buffer	Pointer to the slave address storage buffer	FIT_NO_PTR*1
*p_data1st	Pointer to the first data storage buffer for transmission	FIT_NO_PTR*1	FIT_NO_PTR*1	FIT_NO_PTR*1
*p_data2nd	Pointer to the second data storage buffer for transmission	Pointer to the second data storage buffer for transmission	FIT_NO_PTR*1	FIT_NO_PTR*1
cnt1st	0000 0001h to FFFF FFFFh*2	0	0	0
cnt2nd	0000 0001h to FFFF FFFFh*2	0000 0001h to FFFF FFFFh*2	0	0
callbackfunc	Specify the function name to be used.	Specify the function name to be used.	Specify the function name to be used.	Specify the function name to be used.
ch_no	00h to FFh	00h to FFh	00h to FFh	00h to FFh
dev_sts	Device state flag	Device state flag	Device state flag	Device state flag
rsv1,rsv2	Reserved (value set here has no effect)	Reserved (value set here has no effect)	Reserved (value set here has no effect)	Reserved (value set here has no effect)

Note 1. When using pattern 2, 3, or 4, set "FIT_NO_PTR" for the applicable structure members as shown in the table above.
Note 2. 0 cannot be set.

(1) Pattern 1

As a master device, this function transmits data in two buffers (for the first data and second data) to the slave device.

A start condition (ST) is generated first and then the slave device address is transmitted. The eighth bit specifies the transfer direction, and so this bit is set to 0 (write) when transmitting data. Then, the first data is transmitted. The first data is used when there is data to be transmitted before performing the data transmission. For example, if the slave device is an EEPROM, an internal address in the EEPROM can be transmitted. Next, the second data is transmitted. The second data is the data to be written to the slave device. When a data transmission has started and all data transmissions have completed, a stop condition (SP) is generated, and the bus is released.



Legend:
n: Channel number
ST: Start condition generation
SP: Stop condition generation
ACK:Acknowledge“0”

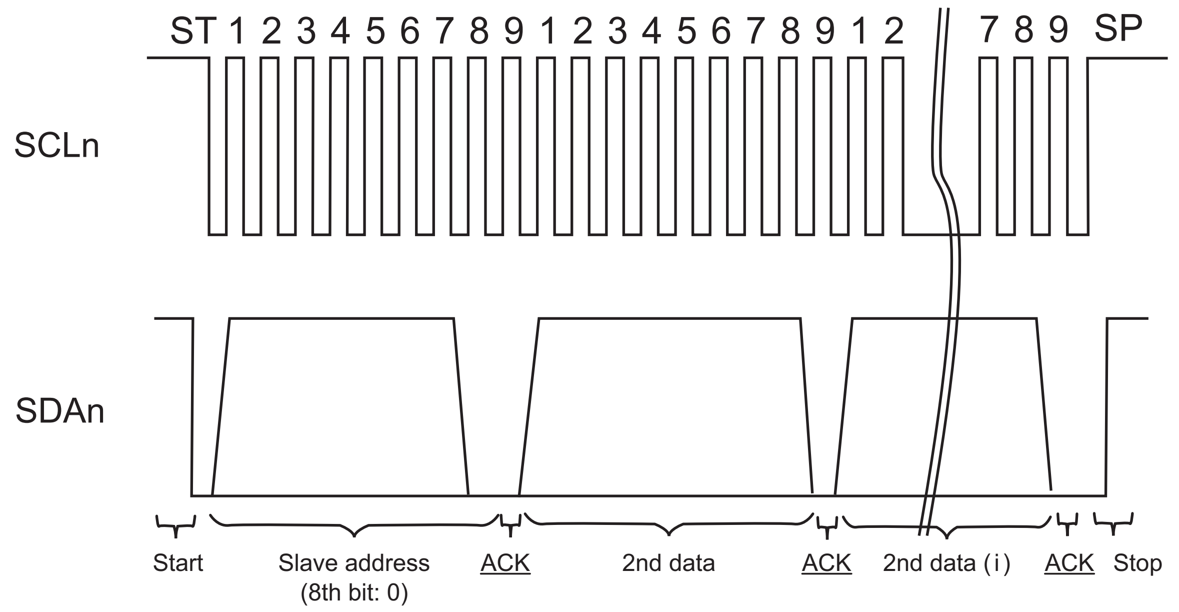
Note: A signal with an underline indicates data transmission from the slave to the master.

Figure 6.4 Signals for Master Transmission Pattern 1

(2) Pattern 2

As a master device, this function transmits data in a buffer (for the second data) to the slave device.

Operations from start condition (ST) generation through to slave device address transmission are the same as for pattern 1. Then the second data is transmitted without transmitting the first data. When all data transmissions have completed, a stop condition (SP) is generated and the bus is released.



Legend:
n: Channel number
ST: Start condition generation
SP: Stop condition generation
ACK:Acknowledge“0”

Note 1. A signal with an underline indicates data transmission from the slave to the master.

Figure 6.5 Signals for Master Transmission Pattern 2

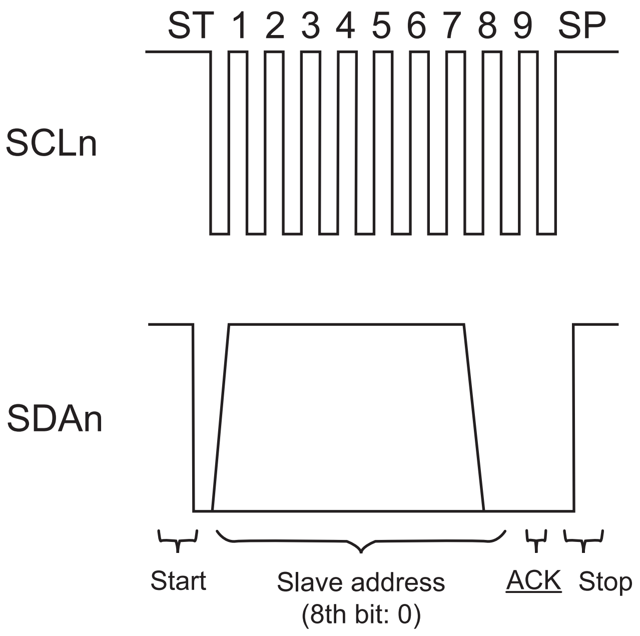
(3) Pattern 3

As a master device, this function transmits only the slave address to the slave device.

Operations from start condition (ST) generation through to slave address transmission are the same as for pattern 1.

After transmitting the slave address, if neither the first data nor the second data is set, data transmission is not performed, then a stop condition (SP) is generated, and the bus is released.

This pattern is useful for detecting connected devices or when performing acknowledge polling to verify the EEPROM rewriting state.



Legend:
n: Channel number
ST: Start condition generation
SP: Stop condition generation
ACK:Acknowledge“0”

Note 1. A signal with an underline indicates data transmission from the slave to the master.

Figure 6.6 Signals for Master Transmission Pattern 3

(4) Pattern 4

As a master device, this function transmits only a start condition and stop condition to the slave device.

After a start condition (ST) is generated, if the slave address, first data, and second data are not set, slave address transmission and data transmission are not performed, then a stop condition (SP) is generated and the bus is released. This pattern is useful for just releasing the bus.

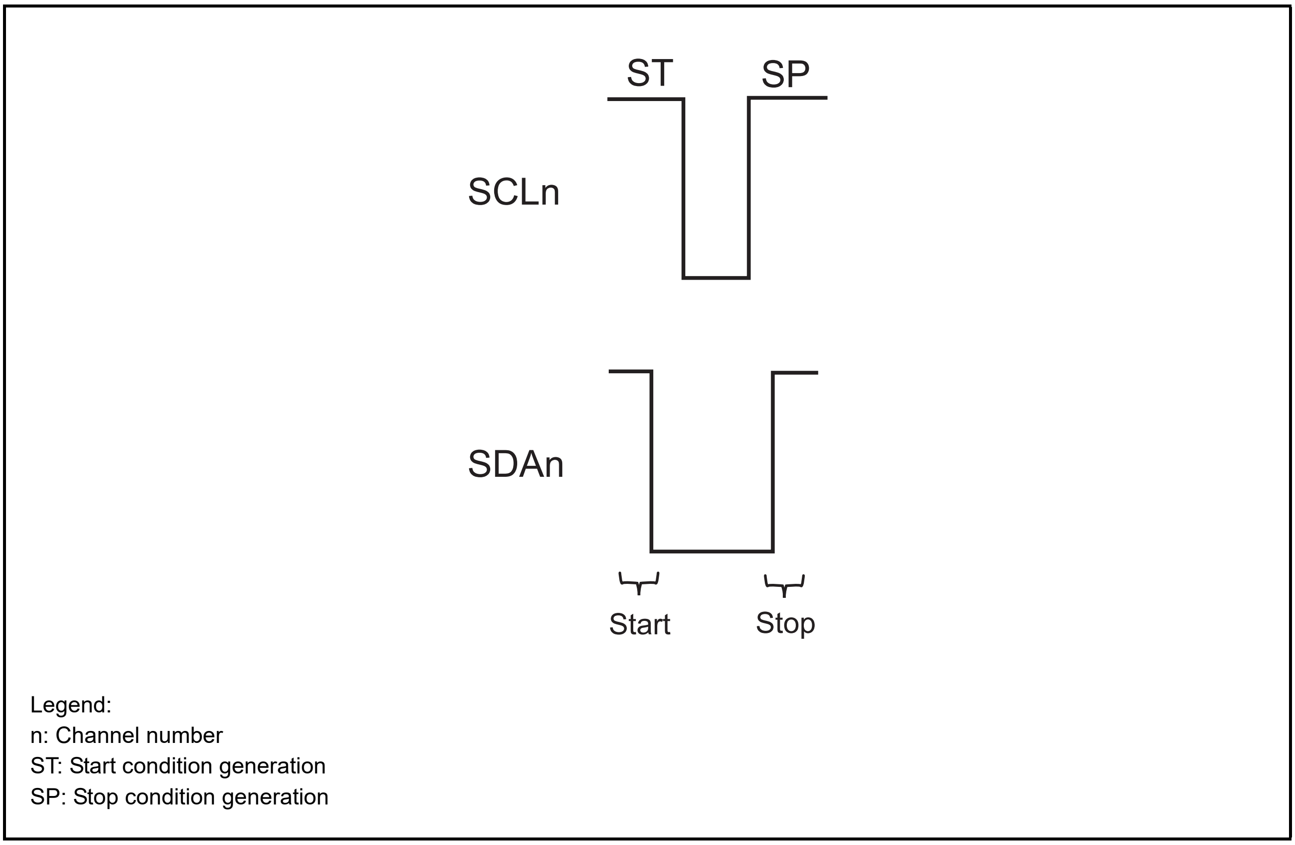


Figure 6.7 Signals for Master Transmission Pattern 4

6.9.3 R_RIIC_MasterReceive

R_RIIC_MasterReceive

Synopsis This function is used when the module starts reception as a master device.

Header r_riic_rx_if.h

Declaration riic_return_t R_RIIC_MasterRecive(riic_info_t * p_riic_info)

Description This function starts the RIIC master reception. The reception is performed with the RIIC channel and reception pattern specified by the arguments. If the state of the channel is "idle" (RIIC_IDLE, RIIC_FINISH, or RIIC_NACK), the following processes are performed:

- Setting the state flag
- Initializing the variables used by the API
- Enabling the RIIC interrupts
- Generating a start condition

*p_riic_info
This is the pointer to the RIIC communication information structure. The reception pattern can be selected from master reception and master composite by the argument settings. See Table 6.11 for the specification method and allowable argument settings for each reception pattern. See Figure 6.8 and Figure 6.9 for the signal waveforms of each reception pattern.
Only the members of this structure that are used by this function are shown below. For details on this structure, see Figure 6.3.
When setting the slave address, store it without shifting 1 bit to left.
For the arguments where "(to be updated)" appears in the comment below, the values of these arguments are updated during the API execution.

```
riic_ch_dev_status_t dev_sts; /* Device state flag (to be updated) */
uint8_t ch_no; /* Channel number */
riic_callback callbackfunc; /* Callback function */
uint32_t cnt2nd; /* Second data counter (number of bytes)(to be updated) */
uint32_t cnt1st; /* First data counter (number of bytes)(to be updated only for master composite)
*/
uint8_t * p_data2nd; /* Pointer to the second data storage buffer */
uint8_t * p_data1st; /* Pointer to the first data storage buffer */
uint8_t * p_slv_adr; /* Pointer to the slave address storage buffer */
```

Arguments riic_info_t * p_riic_info : Pointer to the RIIC communication information structure

Return values	RIIC_SUCCESS	: Processing completed successfully
	RIIC_ERR_INVALID_CHAN	: Nonexistent channel
	RIIC_ERR_INVALID_ARG	: Invalid argument
	RIIC_ERR_NO_INIT	: Uninitialized state
	RIIC_ERR_BUS_BUSY	: Bus busy
	RIIC_ERR_AL	: Arbitration-lost error occurred
	RIIC_ERR_OTHER	: An invalid event occurred in the current state

Remarks None

Example

```
/* for MasterReceive(combination mode) */
#include "r_riic_rx_if.h"

void CallbackMaster(void);
void main(void);

void main(void)
{
      volatile riic_return_t ret;
      riic_info_t iic_info_m;
      uint8_t addr_eeprom[1]={0x50};
      uint8_t access_addr1[1]={0x00};
      uint8_t mst_store_area[5]={0xFF,0xFF,0xFF,0xFF,0xFF};

      /* Sets IIC Information. */
      iic_info_m.dev_sts = 0;
      iic_info_m.ch_no = 0;
      iic_info_m.callbackfunc = &CallbackMaster;
      iic_info_m.cnt2nd = 3;
      iic_info_m.cnt1st = 1;
      iic_info_m.p_data2nd = mst_store_area;
      iic_info_m.p_data1st = access_addr1;
      iic_info_m.p_slv_adr = addr_eeprom;

      /* RIIC open */
      ret = R_RIIC_Open(&iic_info_m);

      /* RIIC receive start */
      ret = R_RIIC_MasterReceive(&iic_info_m);

      while(1);
}

void CallbackMaster(void)
{
   /* callback process */
}
```

Special Notes:

The following table lists the allowable argument settings for each reception pattern.

Table 6.11 Allowable Argument Settings for Each Reception Pattern

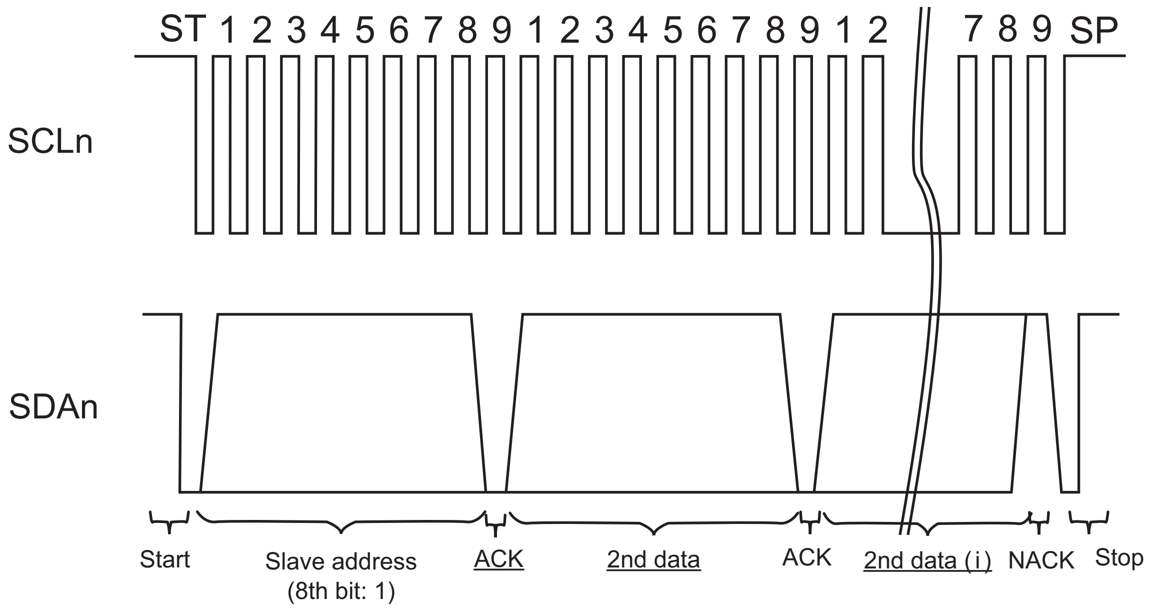
Structure Member	User Settable Range	
	Master Reception	**Master Composite**
*p_slv_adr	Pointer to the slave address storage buffer	Pointer to the slave address storage buffer
*p_data1st	Not used (value set here has no effect)	Pointer to the first data storage buffer for transmission
*p_data2nd	Pointer to the second data storage buffer for reception	Pointer to the second data storage buffer for reception
dev_sts	Device state flag	Device state flag
cnt1st*1	0	0000 0001h to FFFF FFFFh*2
cnt2nd	0000 0001h to FFFF FFFFh*2	0000 0001h to FFFF FFFFh*2
callbackfunc	Specify the function name to be used.	Specify the function name to be used.
ch_no	00h to FFh	00h to FFh
rsv1,rsv2	Reserved (value set here has no effect)	Reserved (value set here has no effect)

Note 1. The reception pattern is determined by whether the first data is 0 or not.
Note 2. 0 cannot be set.

(1) Master Reception

As a master device, this function receives data from the slave device.

A start condition (ST) is generated first and then the slave device address is transmitted. The eighth bit specifies the transfer direction, and so this bit is set to 1 (read) when receiving data. Then, data reception starts. An ACK is sent each time one byte of data is received, but when the last data is received, a NACK is sent to notify the slave device that all data receptions have completed. When all data receptions have completed, a stop condition (SP) is generated and the bus is released.



Legend:
n: Channel number
ST: Start condition generation
SP: Stop condition generation
NACK:Acknowledge“1”
ACK:Acknowledge“0”

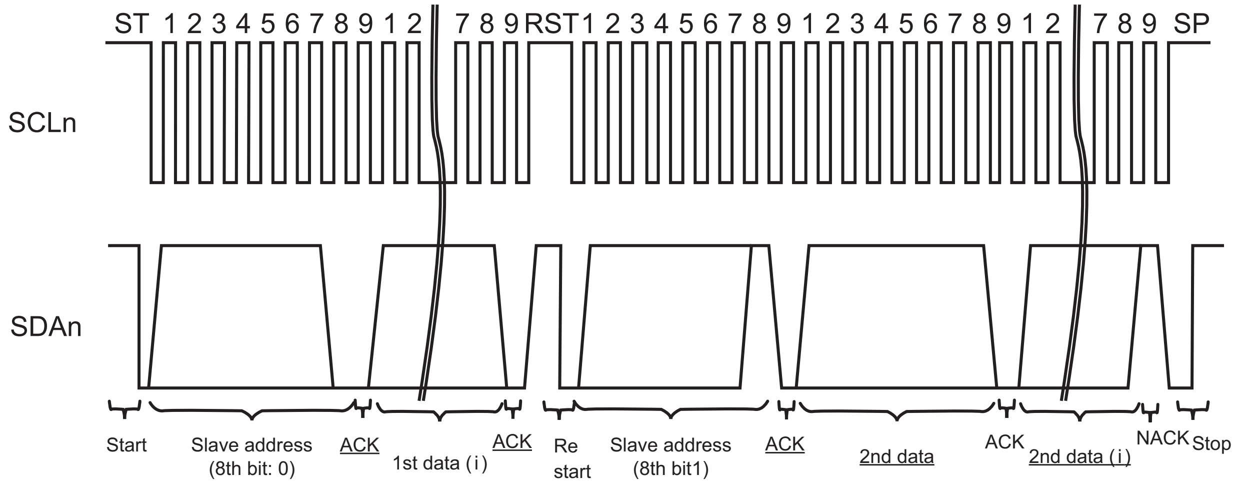
Note 1. A signal with an underline indicates data transmission from the slave to the master.

Figure 6.8 Signals for Master Reception

(2) Master Composite

As a master device, this function transmits data to the slave device. After the transmission completes, this function generates a restart condition and receives data from the slave.

A start condition (ST) is generated first and then the slave device address is transmitted. The eighth bit specifies the transfer direction, and so this bit is set to 0 (write) when transmitting data. Next, the first data is transmitted. When the data transmission completes, a restart condition (RST) is generated and the slave address is transmitted. Then, the eighth bit is set to 1 (read) and a data reception starts. An ACK is sent each time one byte of data is received, but when the last data is received, a NACK is sent to notify the slave device that all data receptions have completed. When all data receptions have completed, a stop condition (SP) is generated and the bus is released.



Legend:
n: Channel number
ST: Start condition generation
SP: Stop condition generation
RST: Restart condition generation
NACK:Acknowledge“1”
ACK:Acknowledge“0”

Note 1. A signal with an underline indicates data transmission from the slave to the master.

Figure 6.9 Signals for Master Composite

6.9.4 R_RIIC_SlaveTransfer

R_RIIC_SlaveTransfer

Synopsis: This function is used when the module functions as a slave device to prepare for transmission and reception.

Header: r_riic_rx_if.h

Declaration: riic_return_t R_RIIC_SlaveTransfer (riic_info_t *p_riic_info)

Description: This function prepares for the RIIC slave transmission or slave reception. If this function is called while the master is communicating, an error occurs. This function sets the RIIC channel specified by the argument. If the state of the channel is "idle" (RIIC_IDLE, RIIC_FINISH, or RIIC_NACK), the following processes are performed:

- Setting the state flag
- Initializing the variables used by the API
- Initializing the RIIC registers used for RIIC communications
- Enabling the RIIC interrupts
- Setting the slave address and enabling the slave address match interrupt

*p_riic_info
This is the pointer to the RIIC communication information structure. The operation can be selected from preparation for slave reception, slave transmission, or both of them by the argument settings. See Table 6.12 for the allowable argument settings for each slave operation pattern. See Figure 6.10 for the signal waveforms of the reception pattern and Figure 6.11 for the signal waveforms of the transmission pattern.
Only the members of this structure that are used by this function are shown below. For details on this structure, see Figure 6.3.
For the arguments where "(to be updated)" appears in the comment below, the values of these arguments are updated during the API execution.

```
riic_ch_dev_status_t dev_sts; /* Device state flag (to be updated) */
uint8_t ch_no; /* Channel number */
riic_callback callbackfunc; /* Callback function */
uint32_t cnt2nd; /* Second data counter (number of bytes)(to be updated for only slave
reception) */
uint32_t cnt1st; /* First data counter (number of bytes) (to be updated for only slave
transmission) */
uint8_t * p_data2nd; /* Pointer to the second data storage buffer */
uint8_t * p_data1st; /* Pointer to the first data storage buffer */
```

Arguments	riic_info_t * p_riic_info	: Pointer to the RIIC communication information structure
Return values	RIIC_SUCCESS	: Processing completed successfully
	RIIC_ERR_INVALID_CHAN	: Nonexistent channel
	RIIC_ERR_INVALID_ARG	: Invalid argument
	RIIC_ERR_NO_INIT	: Uninitialized state
	RIIC_ERR_BUS_BUSY	: Bus busy
	RIIC_ERR_AL	: Arbitration-lost error occurred
	RIIC_ERR_OTHER	: An invalid event occurred in the current state

Remarks: None

Example

```
/* for MasterReceive(combination mode) */
#include "r_riic_rx_if.h"

void CallbackMaster(void);
void CallbackSlave(void);
void main(void);

void main(void)
{
    volatile riic_return_t ret;
    riic_info_t iic_info_m;
    riic_info_t iic_info_s;
    uint8_t addr_eeprom[1]={0x50};
    uint8_t access_addr1[1]={0x00};
    uint8_t mst_send_data[5]={0x81,0x82,0x83,0x84,0x85};
    uint8_t slv_send_data[5]={0x71,0x72,0x73,0x74,0x75};
    uint8_t mst_store_area[5]={0xFF,0xFF,0xFF,0xFF,0xFF};
    uint8_t slv_store_area[5]={0xFF,0xFF,0xFF,0xFF,0xFF};

    /* Sets IIC Information for Master Send. */
    iic_info_m.dev_sts = 0;
    iic_info_m.ch_no = 0;
    iic_info_m.callbackfunc = &CallbackMaster;
    iic_info_m.cnt2nd = 3;
    iic_info_m.cnt1st = 1;
    iic_info_m.p_data2nd = mst_store_area;
    iic_info_m.p_data1st = access_addr1;
    iic_info_m.p_slv_adr = addr_eeprom;

    /* Sets IIC Information for Slave Transfer. */
    iic_info_s.dev_sts = 0;
    iic_info_s.ch_no = 0;
    iic_info_s.callbackfunc = &CallbackSlave;
    iic_info_s.cnt2nd = 3;
    iic_info_s.cnt1st = 3;
    iic_info_s.p_data2nd = slv_store_area;
    iic_info_s.p_data1st = slv_send_data;
    iic_info_s.p_slv_adr = (uint8_t*)FIT_NO_PTR;

    /* RIIC open */
    ret = R_RIIC_Open(&iic_info_m);

    /* RIIC slave transfer enable */
    ret = R_RIIC_SlaveTransfer(&iic_info_s);

    /* RIIC master send start */
    ret = R_RIIC_MasterSend(&iic_info_m);
    while(1);
}

void CallbackMaster(void)
{
    /* callback process (master)*/
}

void CallbackSlave(void)
{
    /* callback process (slave)
}
```

Special Notes:

The following table lists the allowable argument settings for each slave operation pattern.

Table 6.12 Allowable Argument Settings for Each Slave Operation Pattern

	User Settable Range	
Structure Member	Slave Reception	Slave Transmission
*p_slv_adr	Not used (value set here has no effect)	Not used (value set here has no effect)
*p_data1st	(For slave transmission)	Pointer to the first data storage buffer for transmission*1
*p_data2nd	Pointer to the second data storage buffer for reception*2	(For slave reception)
dev_sts	Device state flag	Device state flag
cnt1st	(For slave transmission)	0000 0001h to FFFF FFFFh
cnt2nd	0000 0001h to FFFF FFFFh	(For slave reception)
callbackfunc	Specify the function name to be used.	Specify the function name to be used.
ch_no	00h to FFh	00h to FFh
rsv1,rsv2	Reserved (value set here has no effect)	Reserved (value set here has no effect)

Note 1. Set this when performing slave transmission.
When slave transmission is not used, set FIT_NO_PTR.

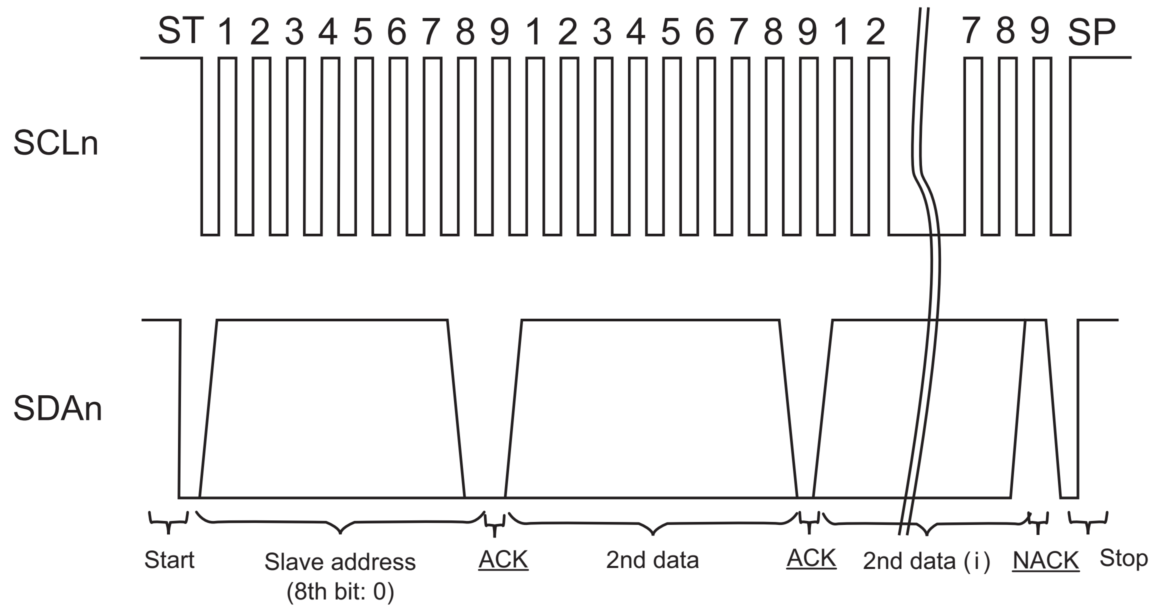
Note 2. Set this when performing slave reception.
When slave reception is not used, set FIT_NO_PTR.

(1) Slave Reception

As a slave device, this function receives data from the master device.

When the slave address specified by the master device matches the slave address specified in r_riic_config_if.h, slave transmission and reception starts. This module performs processing by automatically determining whether the operation is slave reception or slave transmission according to the eighth bit (transfer direction specification bit) of the slave address.

After a start condition (ST) generated by the master device is detected, if the received slave address matches its own address and the eighth bit of the slave address is 0 (write), then the module starts reception operation as a slave device . When the last data (the number of received data items specified in the RIIC communication information structure) is received, a NACK is returned to the master device to notify that all the necessary data has been received.



Legend:
n: Channel number
ST: Start condition generation
SP: Stop condition generation
NACK:Acknowledge“1”
ACK:Acknowledge“0”

Note 1. A signal with an underline indicates data transmission from the slave to the master.

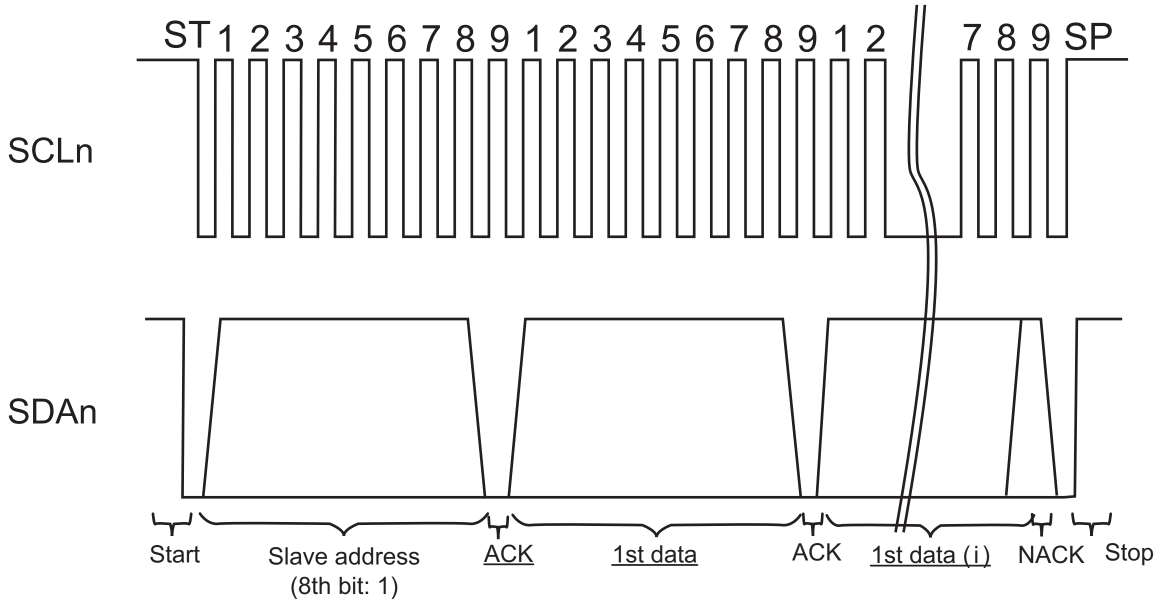
Figure 6.10 Signals for Slave Reception

(2) Slave Transmission

As a slave device, this function transmits data to the master device.

When the slave address specified by the master device matches the slave address specified in r_riic_config_if.h, slave transmission and reception starts. This module performs processing by automatically determining whether the operation is slave reception or slave transmission according to the eighth bit (transfer direction specification bit) of the slave address.

After a start condition (ST) from the master device is detected, if the received slave address matches its own address and the eighth bit of the slave address is 1 (read), then the module starts transmission operation as a slave device. If the transmission request exceeds the number of sent data items specified in the I2C communication information structure, 0xFF is sent as data. The slave continues transmitting data until a stop condition (SP) is detected.



Legend:
n: Channel number
ST: Start condition generation
SP: Stop condition generation
NACK:Acknowledge“1”
ACK:Acknowledge“0”

Note 1. A signal with an underline indicates data transmission from the slave to the master.

Figure 6.11 Signals for Slave Transmission

6.9.5 R_RIIC_GetStatus

R_RIIC_GetStatus

Synopsis	This function is used when verifying the state of this module.	
Header	r_riic_rx_if.h	
Declaration	riic_sts_flg_t R_RIIC_GetStatus(riic_info_t * p_riic_info, riic_mcu_status_t * p_riic_status)	
Description	This function returns the state of this module. This function obtains the state of the RIIC channel specified in the argument by reading the registers, pin levels, variables, and others, and then returns the state as a return value (32-bit structure). When this function is called, the RIIC arbitration-lost detection flag and NACK flag are cleared to 0. If the device state is "RIIC_ AL", the value is updated to "RIIC_FINISH". *p_riic_info This is the pointer to the RIIC communication information structure. Only the members of this structure that are used by this function are shown below. For details on this structure, see Figure 6.3. For the arguments where "(to be updated)" appears in the comment below, the values of these arguments are updated during the API execution. riic_ch_dev_status_t dev_sts; /* Device state flag (to be updated when the state is “RIIC_AL”) uint8_t ch_no; /* Channel number */ *p_riic_status Pointer to the variable to store the RIIC state	
Arguments	riic_info_t * p_riic_info	: Pointer to the RIIC communication information structure
	riic_mcu_status_t * p_riic_status	: Pointer to the variable to store the RIIC state
Return values	RIIC_SUCCESS	: Processing completed successfully
	RIIC_ERR_INVALID_CHAN	: Nonexistent channel
	RIIC_ERR_INVALID_ARG	: Invalid argument
Remarks	None	

Example

```
volatile riic_return_t ret;
riic_info_t iic_info_m;
riic_mcu_status_t riic_status;

iic_info_m.ch_no = 0;

ret = R_RIIC_GetStatus(&iic_info_m, &riic_status);
```

6.9.6 R_RIIC_Control

R_RIIC_Control

Synopsis This function is mainly used when a communication error occurs.
It outputs conditions, high impedance signals to the SDA pin, and one-shot of the SCL clock. It also resets the RIIC module.

Header r_riic_rx_if.h

Declaration riic_return_t R_RIIC_Control(r_riic_info_t *p_riic_info, uint8_t ctrl_ptn)

Description This function outputs control signals for the RIIC module. It outputs conditions specified by the arguments, high impedance signals to the SDA pin, and one-shot of the SCL clock. It also resets the RIIC module.

*p_riic_info
This is the pointer to the I^{2}C communication information structure.
Only the members of this structure that are used by this function are shown below. For details on this structure, see Figure 6.3.
For the arguments where "(to be updated)" appears in the comment below, the values of these arguments are updated during the API execution.

```
riic_ch_dev_status_t dev_sts; /* Device state flag (to be updated when "RIIC_GEN_RESET" is
                                  specified as the output pattern ) */
uint8_t ch_no; /* Channel number */
```

ctrl_ptn
Specifies the output pattern.
See Table 6.5 for the values that can be specified as output patterns.
The following output patterns can be specified simultaneously. When specifying multiple patterns simultaneously, separate them with a vertical bar ("|").

- The following output patterns can be specified simultaneously by combining two or three of them: RIIC_GEN_START_CON, RIIC_GEN_STOP_CON, and RIIC_GEN_RESTART_CON
- The following two can specified simultaneously: RIIC_GEN_SDA_HI_Z and RIIC_GEN_SCL_ONESHOT

```
RIIC_GEN_START_CON     0x01 /* Start condition generation */
RIIC_GEN_STOP_CON      0x02 /* Stop condition generation */
RIIC_GEN_RESTART_CON 0x04 /* Restart condition generation */
RIIC_GEN_SDA_HI_Z       0x08 /* SDA pin set to high impedance */
RIIC_GEN_SCL_ONESHOT 0x10 /* SCL clock one-shot output */
RIIC_GEN_RESET           0x20 /* RIIC module reset */
```

Arguments	riic_info_t * p_riic_info	: Pointer to the RIIC communication information structure
	riic_mcu_status_t * p_riic_status	: Pointer to the variable to store the RIIC state
Return values	RIIC_SUCCESS	: Processing completed successfully
	RIIC_ERR_INVALID_CHAN	: Nonexistent channel
	RIIC_ERR_INVALID_ARG	: Invalid argument
	RIIC_ERR_NO_INIT	: Uninitialized state
	RIIC_ERR_BUS_BUSY	: Bus busy
	RIIC_ERR_AL	: Arbitration-lost error occurred
	RIIC_ERR_OTHER	: An invalid event occurred in the current state

Remarks None

Example

```
/* Outputs an extra SCL clock cycle after changes the SDA pin in a high-impedance state*/
volatile riic_return_t ret;
riic_info_t iic_info_m;

iic_info_m.ch_no = 0;

ret = R_RIIC_Control(&iic_info_m, RIIC_GEN_SDA_HI_Z | RIIC_GEN_SCL_ONESHOT);
```

6.9.7 R_RIIC_Close

R_RIIC_Close

Synopsis	This function is used when completing the RIIC communication and releasing the RIIC module used.
Header	r_riic_rx_if.h
Declaration	riic_return_t R_RIIC_Close(riic_info_t *p_riic_info)
Description	This function configures the settings to complete the RIIC communication. It disables the RIIC channel specified by the argument. This function performs the following processes: • Changing the RIIC module to a stopped state • Releasing the RIIC output ports (switching SCL0 and SDA0 to port mode (input)) • Disabling the RIIC interrupts To restart the communication, call the R_RIIC_Open() function (initialization function). If the communication is forcibly terminated, that communication is not guaranteed. *p_riic_info This is the pointer to the RIIC communication information structure. Only the members of this structure that are used by this function are shown below. For details on this structure, see Figure 6.3. For the arguments where "(to be updated)" appears in the comment below, the values of these arguments are updated during the API execution. riic_ch_dev_status_t dev_sts; /* Device state flag (to be updated) */ uint8_t ch_no; /* Channel number */
Arguments	riic_info_t * p_riic_info : Pointer to the RIIC communication information structure
Return values	RIIC_SUCCESS : Processing completed successfully RIIC_ERR_INVALID_CHAN : Nonexistent channel RIIC_ERR_INVALID_ARG : Invalid argument
Remarks	None

Example

```
volatile riic_return_t ret;
riic_info_t iic_info_m;

iic_info_m.ch_no = 0;

ret = R_RIIC_Close(&iic_info_m);
```

6.9.8 R_RIIC_GetVersion

R_RIIC_GetVersion	
Synopsis	This function returns the API version.
Header	r_riic_rx_if.h
Declaration	uint32_t R_RIIC_GetVersion(void)
Description	This function returns the version number of this API.
Arguments	None : –
Return values	Version number
Remarks	None

Example

```
uint32_t version;

version = R_RIIC_GetVersion();
```

6.9.9 main

main	
Synopsis	This function is the main function of the sample program.
Header	–
Declaration	int main(void)
Description	This function performs the main processing for the sample program. For details on the processing, see Section 6.10.1, Main Processing.
Arguments	None
Return values	0
Remarks	None

6.10 Flowcharts

6.10.1 Main Processing

Figure 6.12 shows the flowchart for the Main Processing of the sample code.

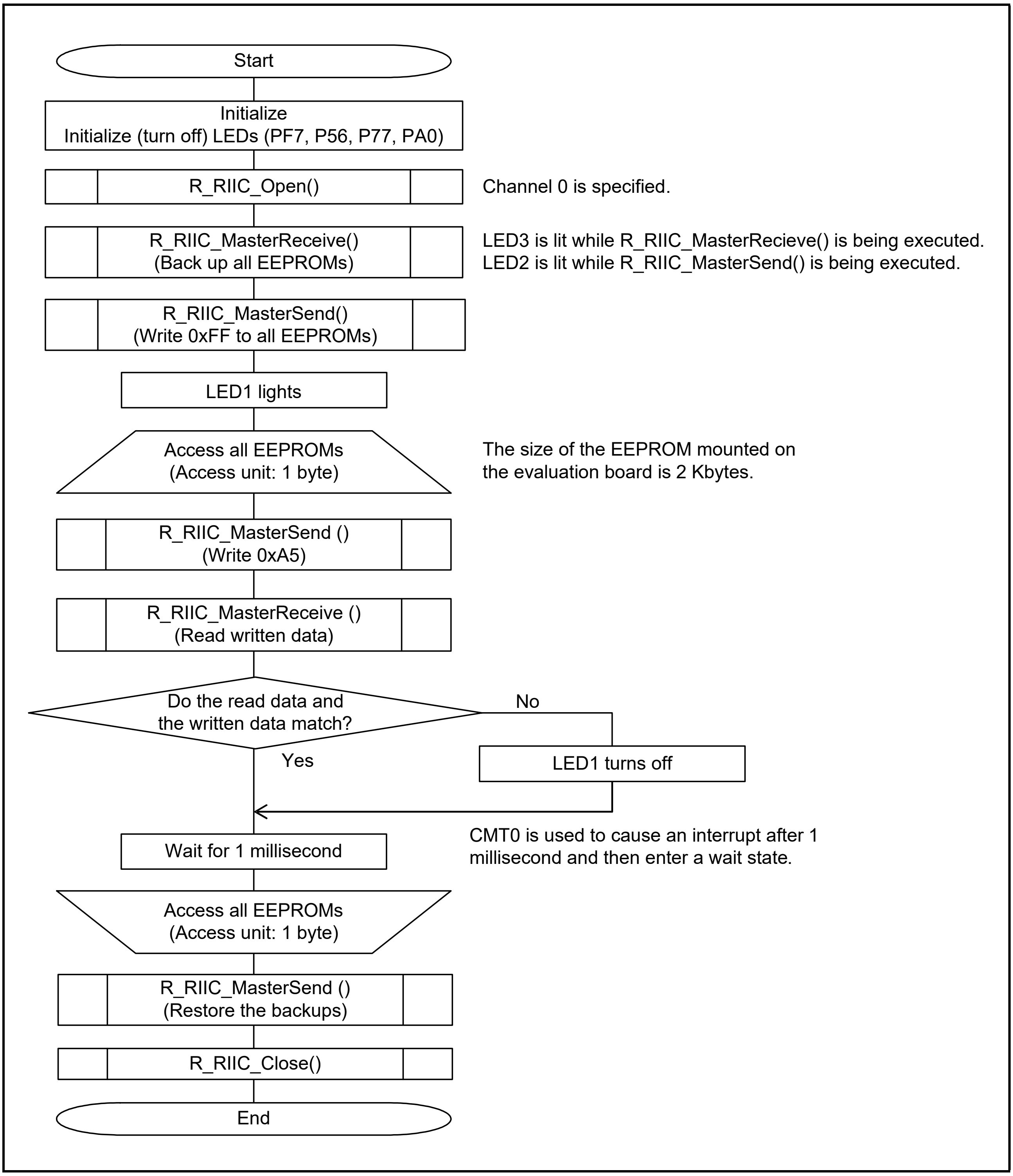

Start
Initialize
Initialize (turn off) LEDs (PF7, P56, P77, PA0)
R_RIIC_Open()
Channel 0 is specified.
R_RIIC_MasterReceive()
(Back up all EEPROMs)
LED3 is lit while R_RIIC_MasterRecieve() is being executed.
LED2 is lit while R_RIIC_MasterSend() is being executed.
R_RIIC_MasterSend()
(Write 0xFF to all EEPROMs)
LED1 lights
Access all EEPROMs
(Access unit: 1 byte)
The size of the EEPROM mounted on the evaluation board is 2 Kbytes.
R_RIIC_MasterSend ()
(Write 0xA5)
R_RIIC_MasterReceive ()
(Read written data)
Do the read data and the written data match?
No
LED1 turns off
Yes
Wait for 1 millisecond
CMT0 is used to cause an interrupt after 1 millisecond and then enter a wait state.
Access all EEPROMs
(Access unit: 1 byte)
R_RIIC_MasterSend ()
(Restore the backups)
R_RIIC_Close()
End

Figure 6.12 Main Processing

6.10.2 Callback Processing

Figure 6.13 shows the flowchart for the Callback Processing of the sample code.

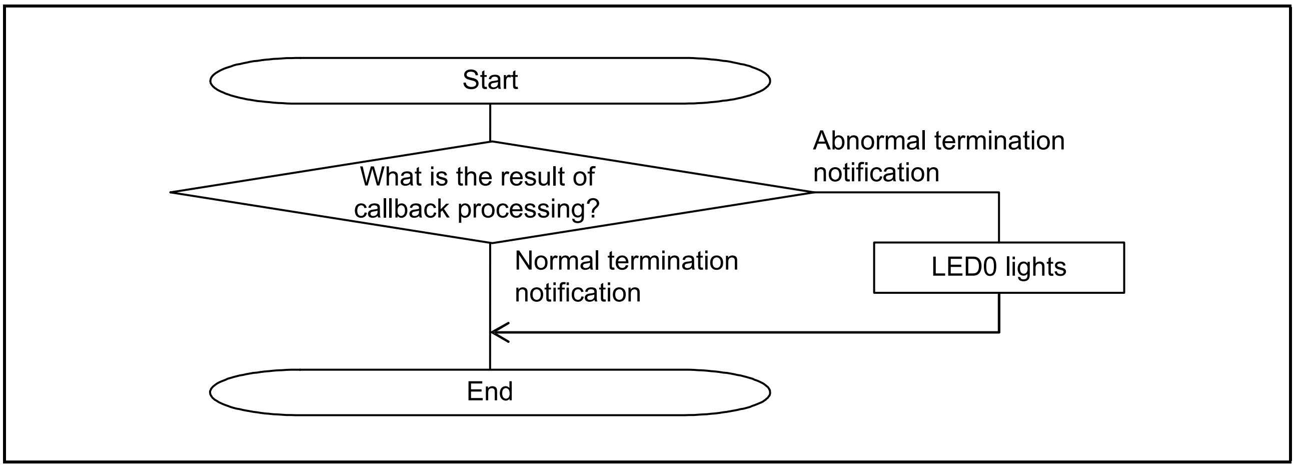

Start
What is the result of callback processing?
Abnormal termination notification
LED0 lights
Normal termination notification
End

Figure 6.13 Callback Processing

6.10.3 Compare Match Timer Interrupt Processing

Figure 6.14 shows the flowchart for the Compare Match Timer Interrupt Processing of the sample code.

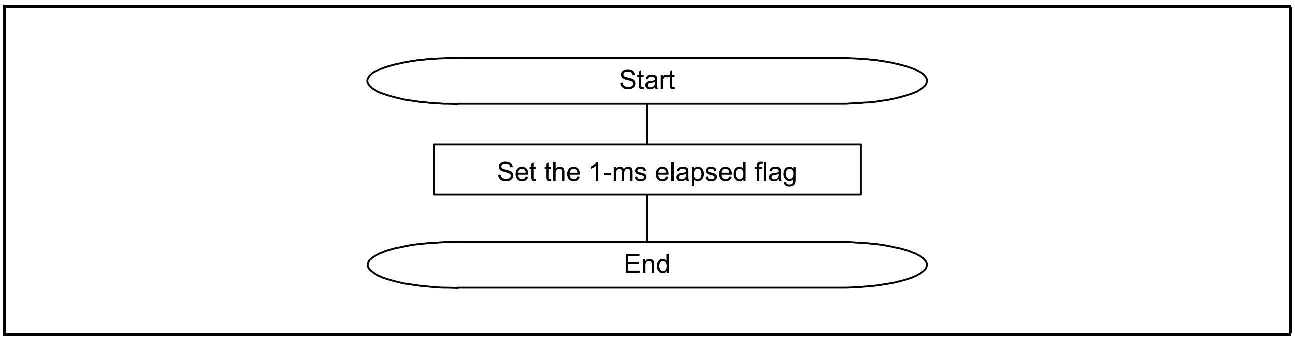


Figure 6.14 Compare Match Timer Interrupt Processing

7. Sample Code

Download the sample code from the Renesas Electronics website.

8. Related Documents

- User's Manuals: Hardware
  RZ/T1 Group User's Manual: Hardware
  (Download the latest edition from the Renesas Electronics website.)

  RZ/T1 Evaluation Board RTK7910022C00000BR User's Manual
  (Download the latest edition from the Renesas Electronics website.)

- Technical Update and Technical News
  (Download the latest information from the Renesas Electronics website.)

- User's Manuals: Development Environment
  For the IAR Embedded Workbench® for Arm, download the user's manual from the IAR website.
  (Download the latest edition from the IAR website.)

Website and Support

Renesas Electronics website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision History	Application Note: RIIC Sample Program

Rev.	Date	Description	
		Page	Summary
0.10	Apr. 02, 2015	—	First Edition issued
1.00	Apr. 10, 2015	—	Only the revision number was changed to be posted on a website.
1.10	Jul. 16, 2015	2. Operating Environment	
		4	Table 2.1 Operating Environment: Description added to Integrated Development Environment
		6. Software	
		12	6.2.4 Required Memory Size: Description and reference added
		12	Table 6.3: Table title and size description were partially amended
		12	Table 6.3 Memory Requirements: Description on the Note and Size, changed
		13	Table 6.4 added
		13	Table 6.5 added
1.20	Dec. 04, 2015	2. Operating Environment	
		4	Table 2.1 Operating Environment: Integrated Development Environment, information partially amended
1.30	Jul. 18, 2017	1. Specifications	
		3	Table 1.1 Peripheral Functions and Applications: I/O ports, modified
		2. Operating Environment	
		4	Table 2.1 Operating Environment: Integrated Development Environment, modified
		5. Hardware	
		7	Figure 5.1 Hardware Configuration: I/O ports and LEDs, modified
		7	Table 5.1 Pins and Functions: Pin names and description of I/O ports, modified
		6. Software	
		8	Table 6.1 Operation Outline: LEDs for the operation result display, modified
		—	6.2.4 Required Memory Size, deleted
		43	Figure 6.12 Main Processing: I/O ports and LEDs, modified
		44	Figure 6.13 Callback Processing: LED modified
1.40	Jun. 07, 2018	2. Operating Environment	
		4	Table 2.1 Operating Environment: The description on the integrated development environment, modified
		8. Related Documents	
		46	The name of IAR Embedded Workbench, modified

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. 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 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. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment 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 moment 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 moment 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 moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has 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. Moreover, 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.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.0