SH7262/SH7264 Group

Reading/Writing EEPROM Using I^{2}C Bus Interface 3 Interrupts

R01AN0068EJ0101
Rev. 1.01
Feb.10, 2012

Summary

This application note describes examples of reading/writing EEPROM using the SH7262/SH7264 Microcomputers (MCUs) I^{2}C Bus Interface 3 (IIC3) transmission and reception in single master mode and interrupts.

Target Device

SH7262/SH7264 MCU (In this document, SH7262/SH7264 are described as SH7264.)

Contents

1. Introduction

1.1 Specifications

- Specifies the SH7264 MCU as the master device, and EEPROM as the slave device to write data to EEPROM
- Specifies the SH7264 MCU as the master device, and EEPROM as the slave device to read data from EEPROM
- The transfer rate is set to 391 kHz
- Uses interrupts for transmit end and receive data full

Note: Set the transfer rate to satisfy the EEPROM specifications.

1.2 Modules Used

- I^{2}C Bus Interface (IIC3)
- Interrupt Controller (INTC)

1.3 Applicable Conditions

MCU	SH7262/SH7264
Operating Frequency	Internal clock: 144 MHz Bus clock: 72 MHz Peripheral clock: 36 MHz
Integrated Development Environment	Renesas Electronics Corporation High-performance Embedded Workshop Ver.4.07.00
C Compiler	Renesas Electronics SuperH RISC engine Family C/C++ compiler package Ver.9.03 Release 00
Compiler Options	Default setting in the High-performance Embedded Workshop (-cpu=sh2afpu -fpu=single -debug -gbr=auto -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1)

1.4 Related Application Notes

For more information, refer to the following application notes:

- SH7262/SH7264 Group Example of Initialization
- SH7262/SH7264 Group I^{2}C Bus Interface 3 Reception in Single-Master Mode (Read from EEPROM)
- SH7262/SH7264 Group I^{2}C Bus Interface 3 Transmission in Single-Master Mode (Write in EEPROM)

1.5 About Active-low Pins (Signals)

The symbol "#" suffixed to the pin (or signal) names indicates that the pins (or signals) are active-low.

1.6 Hardware Conditions

This application may be run on the RSK+ for the SH7264 if an IIC EEPROM is added. This may be done by adding a header at J12 and a small PCB with the EEPROM. Refer to RSK+SH7264 User Manual, REG10J0171 for details. Please see section 2.3 of this App note for the specific EEPROM part number.

2. Applications

The SH7264 (master device) writes data to an EEPROM (slave device) using the IIC3 and then receives data from the EEPROM. This application uses interrupts for transmit end and receive data full.

2.1 IIC3 Operation

IIC3 is compliant to the I^{2}C bus (Inter IC Bus) interface specifications invented by Phillips and supports subsets, However, the configuration of registers to control the I^{2}C bus partly differs from that of Philips.

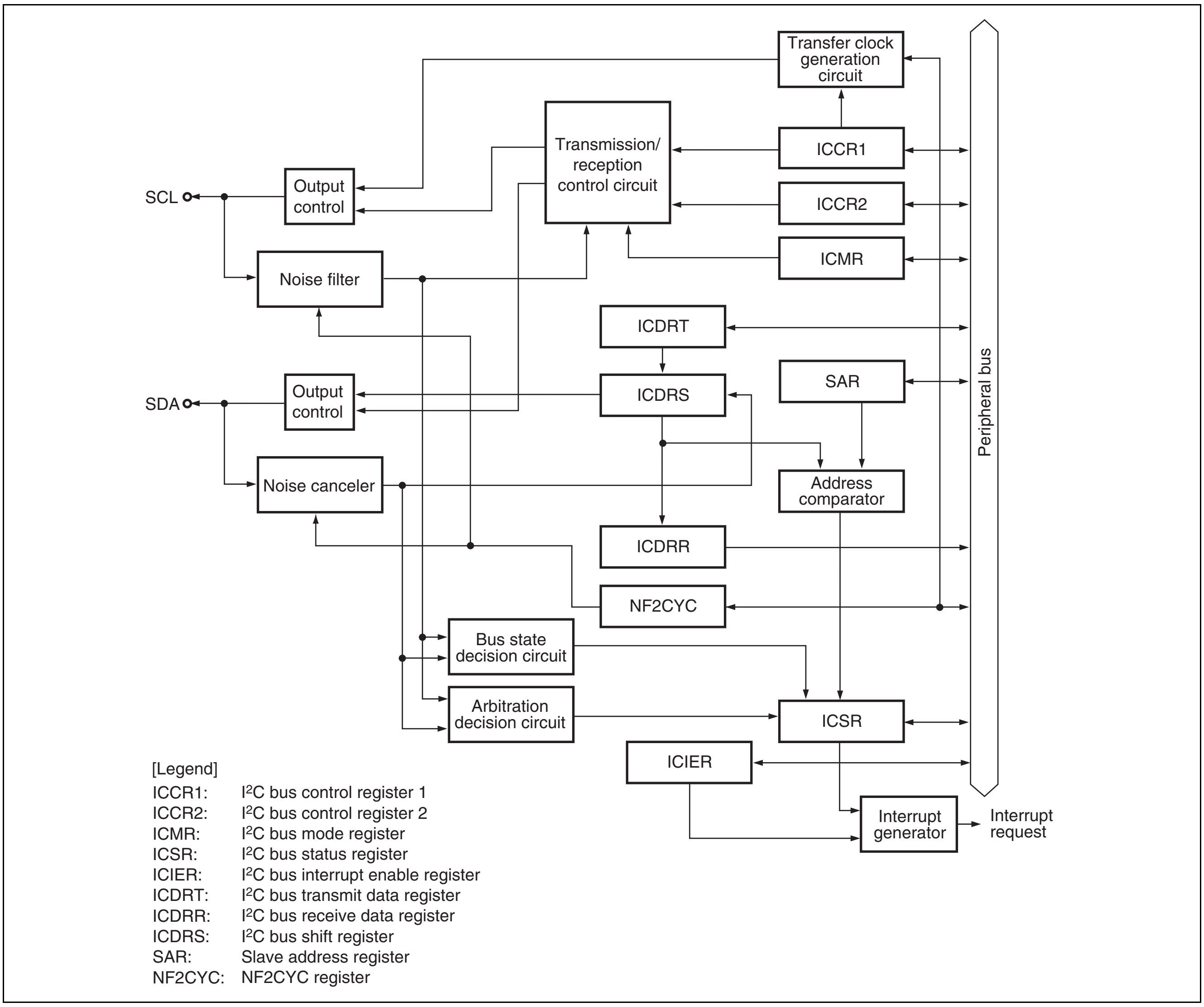
The SH7264 IIC3 has the following features:

- Format options selectable, I^{2}C bus format or clocked synchronous serial format
- Transmits or receives data continuously
  As the shift register, transmit data register and receive data register are separate registers, IIC3 can transmit and receive data continuously.

Table 1 lists the features of two format options. Figure 1 shows the IIC3 block diagram. For details on IIC3, refer to I^{2}C Bus Interface 3 chapter in the SH7262 Group, SH7264 Group Hardware User's Manual.

Table 1 Format Options

Format Name	Description
I^{2}C Bus Format	• Automatically generates the START and STOP conditions in master mode • An output level of an ACK can be selected when receiving data • Automatically loads an ACK bit when transmitting data • Includes the bit synchronization/wait function IIC3 monitors the SCL status per bit in master mode to synchronize automatically. When it is not ready for transfer, it specifies the SCL to low level to wait • Six interrupt sources (1) Transmit data empty (including when slave address match) (2) Transmit end (3) Receive data full (including when slave address match) (4) Arbitration lost (5) NACK detection (6) Stop condition detection • Using the transmit data empty interrupt and the receive data full interrupt to activate the Direct Memory Access Controller (DMAC) and transfer data • Bus can be driven directly SCL and SDA pins are driven by an NMOS open-drain output when selecting the bus drive function
Clocked Synchronous Serial Format	• Four interrupt sources (1) Transmit data empty (2) Transmit end (3) Receive data full (4) Overrun error • Using the transmit data empty interrupt and the receive data full interrupt to activate the Direct Memory Access Controller (DMAC) and transfer data



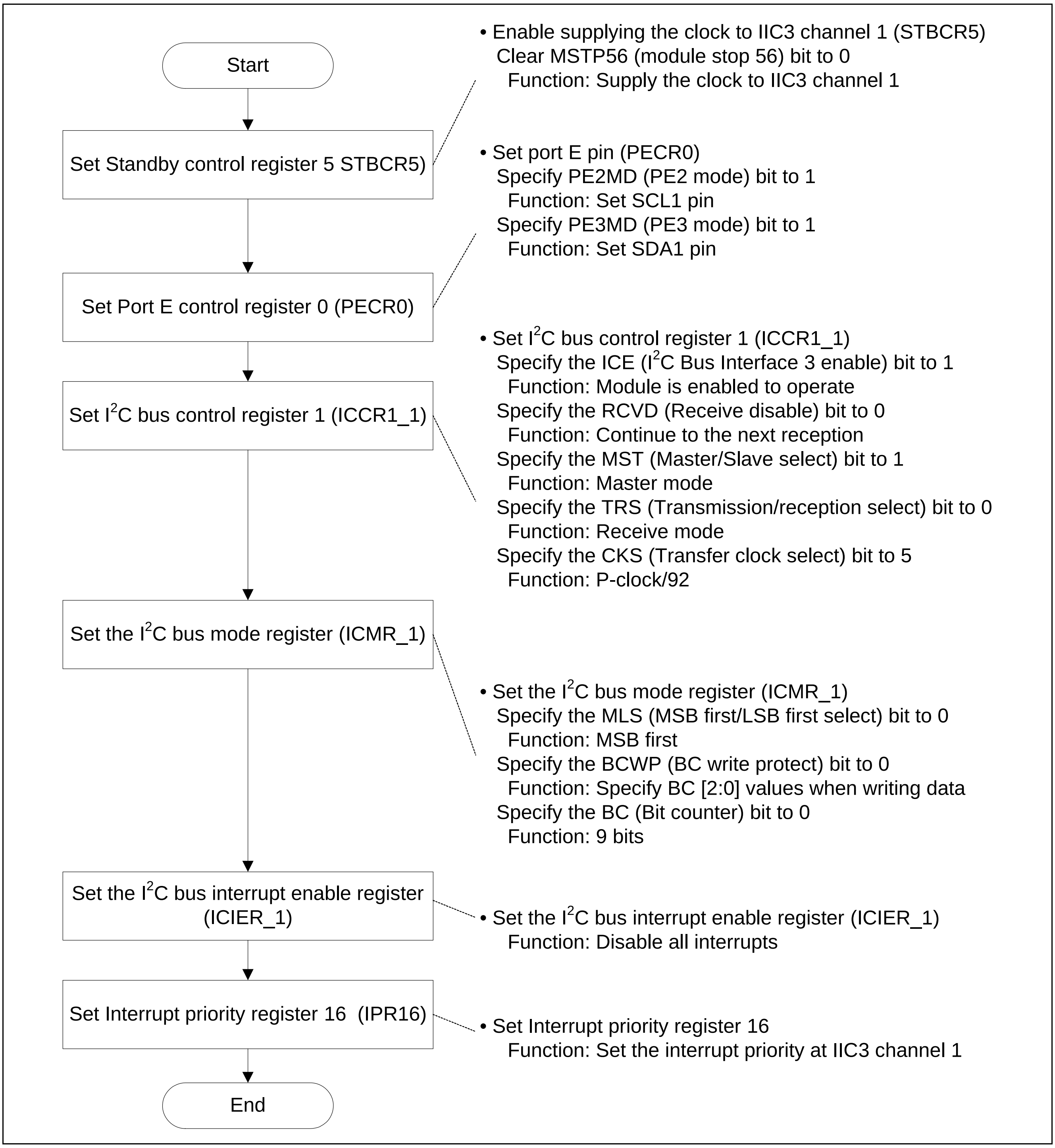
[Legend]

ICCR1:	I²C bus control register 1
ICCR2:	I²C bus control register 2
ICMR:	I²C bus mode register
ICSR:	I²C bus status register
ICIER:	I²C bus interrupt enable register
ICDRT:	I²C bus transmit data register
ICDRR:	I²C bus receive data register
ICDRS:	I²C bus shift register
SAR:	Slave address register
NF2CYC:	NF2CYC register

Figure 1 IIC3 Block Diagram

2.2 IIC3 Setting Procedure

This section describes how to set up IIC3. Make sure to specify the transfer rate to satisfy EEPROM electrical characteristics. Pφ/92 is specified in the sample program. Figure 2 shows the flow chart for configuring IIC3. For more information about the register setting, refer to the SH7262 Group, SH7264 Group Hardware User's Manual.


Start
Set Standby control register 5 STBCR5)
Set Port E control register 0 (PECR0)
Set I2C bus control register 1 (ICCR1_1)
Set the I2C bus mode register (ICMR_1)
Set the I2C bus interrupt enable register (ICIER_1)
Set Interrupt priority register 16 (IPR16)
End

- Enable supplying the clock to IIC3 channel 1 (STBCR5)
  Clear MSTP56 (module stop 56) bit to 0
  Function: Supply the clock to IIC3 channel 1

- Set port E pin (PECR0)
  Specify PE2MD (PE2 mode) bit to 1
  Function: Set SCL1 pin
  Specify PE3MD (PE3 mode) bit to 1
  Function: Set SDA1 pin

- Set I²C bus control register 1 (ICCR1_1)
  Specify the ICE (I²C Bus Interface 3 enable) bit to 1
  Function: Module is enabled to operate
  Specify the RCVD (Receive disable) bit to 0
  Function: Continue to the next reception
  Specify the MST (Master/Slave select) bit to 1
  Function: Master mode
  Specify the TRS (Transmission/reception select) bit to 0
  Function: Receive mode
  Specify the CKS (Transfer clock select) bit to 5
  Function: P-clock/92

- Set the I²C bus mode register (ICMR_1)
  Specify the MLS (MSB first/LSB first select) bit to 0
  Function: MSB first
  Specify the BCWP (BC write protect) bit to 0
  Function: Specify BC [2:0] values when writing data
  Specify the BC (Bit counter) bit to 0
  Function: 9 bits

- Set the I²C bus interrupt enable register (ICIER_1)
  Function: Disable all interrupts

- Set Interrupt priority register 16
  Function: Set the interrupt priority at IIC3 channel 1

Figure 2 IIC3 Configuration Flow Chart

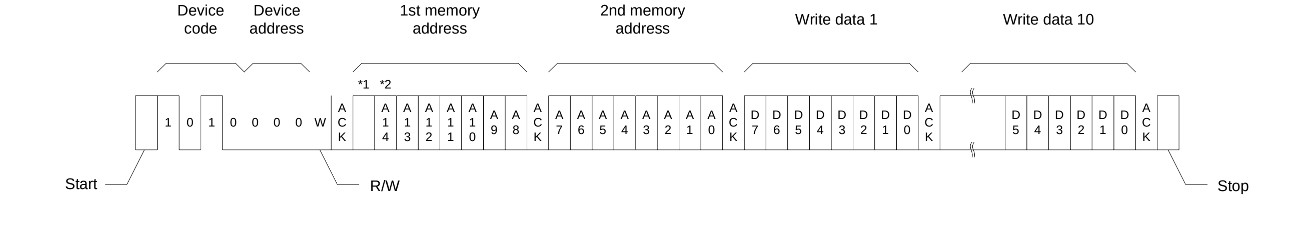
2.3 Sample Program Operation

The sample program specifies IIC3 in master transmit mode to write 10-byte data in pages (page write). From byte 20 and above, it uses the transmit end interrupt (TEI) to write data to the register. Then, it specifies IIC3 in master receive mode to read 10-byte data sequentially (sequential read) and uses the receive data full interrupt (RXI) to read data from the register.

For device codes, refer to the EEPROM data sheet provided by the manufacturer. The sample program uses the device code "B'1010". The sample program uses the device address "B'000". For more information, refer to the EEPROM data sheet provided by the manufacturer.

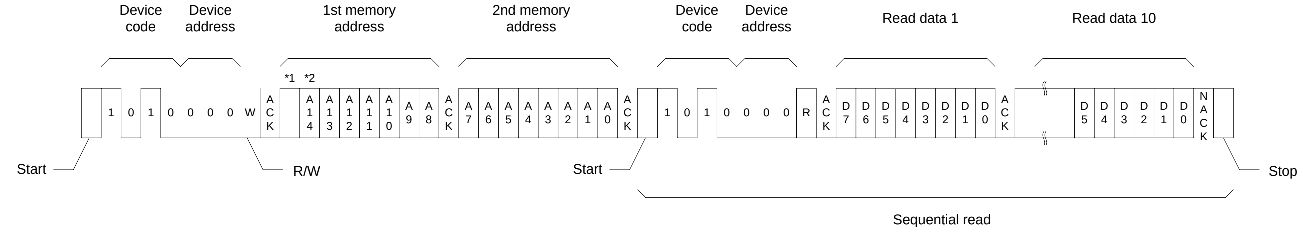
The memory address indicates the write start address or read start address, and the address is incremented at every time writing or reading to/from EEPROM. Figure 3 shows the page write operation. Figure 4 shows the sequential read operation. Figure 5 shows the operating environment of the sample program.

The sample program is tested with the EEPROM (part number: R1EX24128ASA00A, Renesas Electronics).


Device code
Device address
1st memory address
2nd memory address
Write data 1
Write data 10
*1 *2
1 0 1 0 0 0 W ACK A14 A13 A12 A11 A10 A9 A8 ACK A7 A6 A5 A4 A3 A2 A1 A0 ACK D7 D6 D5 D4 D3 D2 D1 D0 ACK D5 D4 D3 D2 D1 D0 ACK
Start
R/W
Stop

Notes:
1. Don't care bits for 128k and 256k.
2. Don't care bit for 128k.

Figure 3 Page Write Operation



Notes:
1. Don't care bits for 128k and 256k
2. Don't care bit for 128k

Figure 4 Sequential Read Operation

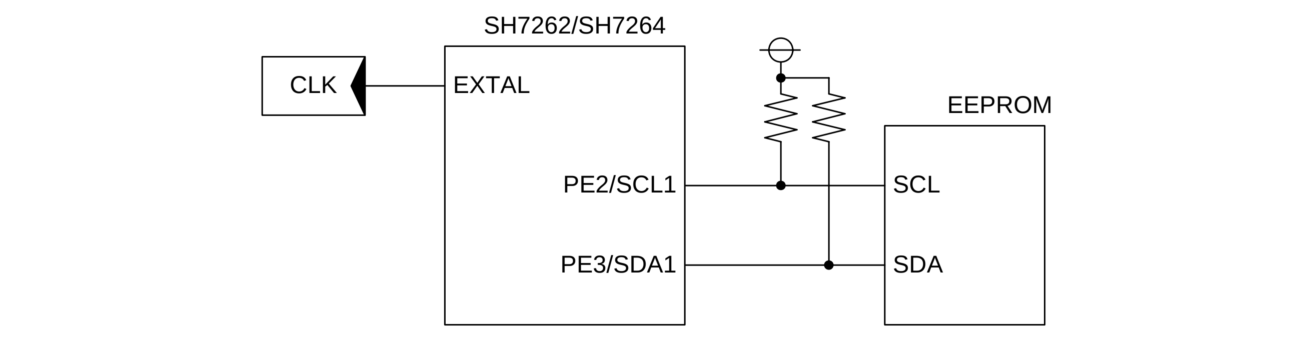


Figure 5 Sample Program Operating Environment

2.4 Sample Program Procedure

Table 2 lists the register settings in the sample program. Table 3 lists the macro definitions used in the sample program. Figure 7 shows the structures used in the sample program Figure 7 to Figure 17 show flow charts of the sample program.

Table 2 Register Settings (Default)

Register Name	Address	Setting	Description
Standby control register 5 (STBCR5)	H'FFFE 0410	H'00	MSTP56 = "0": IIC3 channel 1 is operating

I^{2}C bus control register 1 (ICCR1_1)	H'FFFE E400	H'B5	ICE = "1": SCL/SDA pins are driven by bus
			RCVD = "0": Continues the next reception
			MST = "1", TRS = "1": Master transmit mode
			CKS = "B'0101": Transfer rate is Pφ/92

I^{2}C bus mode register (ICMR_1)	H'FFFE E402	H'30	MLS = "0": MSB first
			BCWP = "0": Sets BC value when writing
			BC = "B'000": 9 bits

I^{2}C bus interrupt enable register (ICIER_1)	H'FFFE E403	H'00	TIE = "0": Disables the transmit data empty interrupt request (TXI) TEIE = "0": Disables the transmit end interrupt request (TEI) RIE = "0": Disables the receive data full interrupt request (RXI) NAKIE = "0": Disables the NACK receive interrupt request (NAKI) STIE = "0": Disables the stop condition detection interrupt request (STPI)
Interrupt priority register 16	H'FFFE 0C14	H'0050	Sets the interrupt priority at IIC3 channel 1 to 5

Table 3 Macro Definitions

Macro Definitions	Setting	Function
EEPROM_MEM_ADDR	H'0000	EEPROM start address
DEVICE_CODE	H'A0	Device code
DEVICE_ADDR	H'00	Device address
IIC_DATA_WR	H'00	Write code
IIC_DATA_RD	H'01	Read code
IIC3_DATA	10	Data transfer size
E_OK	0	Normal end
E_ERR	−1	Error end
IIC3_IDOL	0	Indicates that IIC3 is in idle state
IIC3_NACK	1	Indicates that IIC3 ends with receiving NACK
IIC3_SEND	2	Indicates that IIC3 is transmitting data
IIC3_RECV	3	Indicates that IIC3 is receiving data

Structure variables to define IIC3 control information (iic3_info)

```
typedef enum{
         IIC3_IDOL=0,                        /* IIC3 is in idle state */
                                             /* (transmission/reception is completed) */
         IIC3_NACK,                          /* IIC3 ends with receiving NACK */
         IIC3_SEND,                          /* IIC is transmitting data (using transmit interrupt) */
         IIC3_RECV,                          /* IIC is receiving data (using receive interrupt) */
}IIC3_MODE;

typedef struct{
         volatile IIC3_MODE mode;   /* IIC3 state */
         unsigned char *buffer;              /* Pointer to control the destination when transmitting or */
                                             /* receiving data continuously */
         int count;                          /* Remaining data size when transmitting or */
                                             /* receiving data */
}IIC3_INFO;

static IIC3_INFO iic3_info;
```

iic3_info.mode state transition diagram

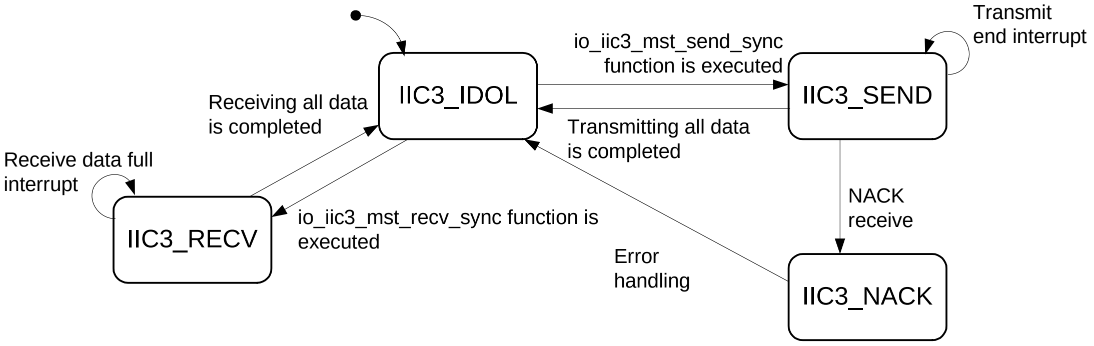
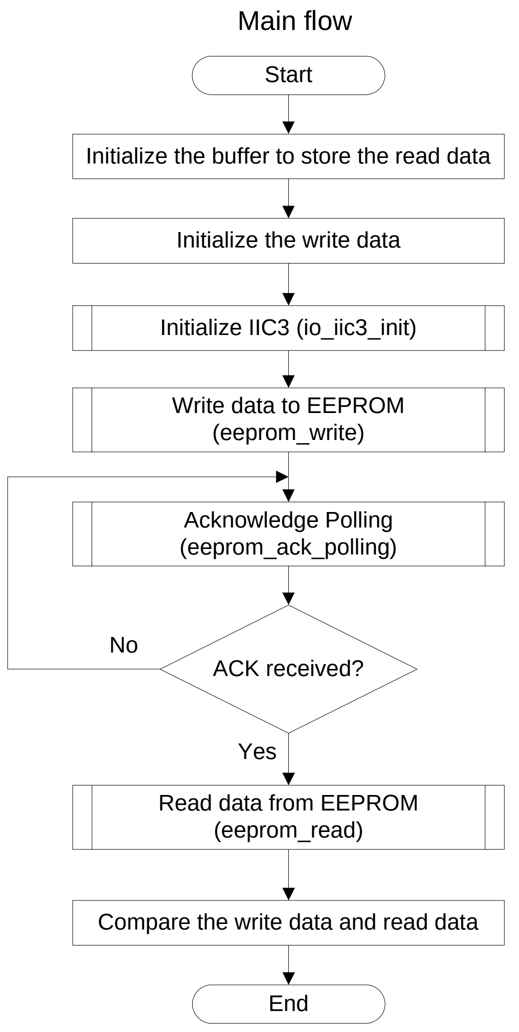


Figure 6 Structures


Main flow
Start
Initialize the buffer to store the read data
Initialize the write data
Initialize IIC3 (io_iic3_init)
Write data to EEPROM (eeprom_write)
Acknowledge Polling (eeprom_ack_polling)
ACK received?
No
Yes
Read data from EEPROM (eeprom_read)
Compare the write data and read data
End

- Refer to Figure 2, IIC3 Configuration Flow Chart
- Set IIC3 in master transmit mode to 10-byte data. Then, use the transmit end interrupt (TEI) to write data to the I^2C bus transmit data register (ICDRT).
  When transmitting all data is completed, it returns to main function.
- Wait until the EEPROM data is replaced
- Set IIC3 in master receive mode to read 10-byte data from EEPROM. Then, use the receive data full interrupt (RXI) to read data from the I^2C bus receive data register (ICDRR).
  When issuing the stop condition and reading all data is completed, it returns to main function.

Figure 7 Sample Program Flow Chart (1/11)

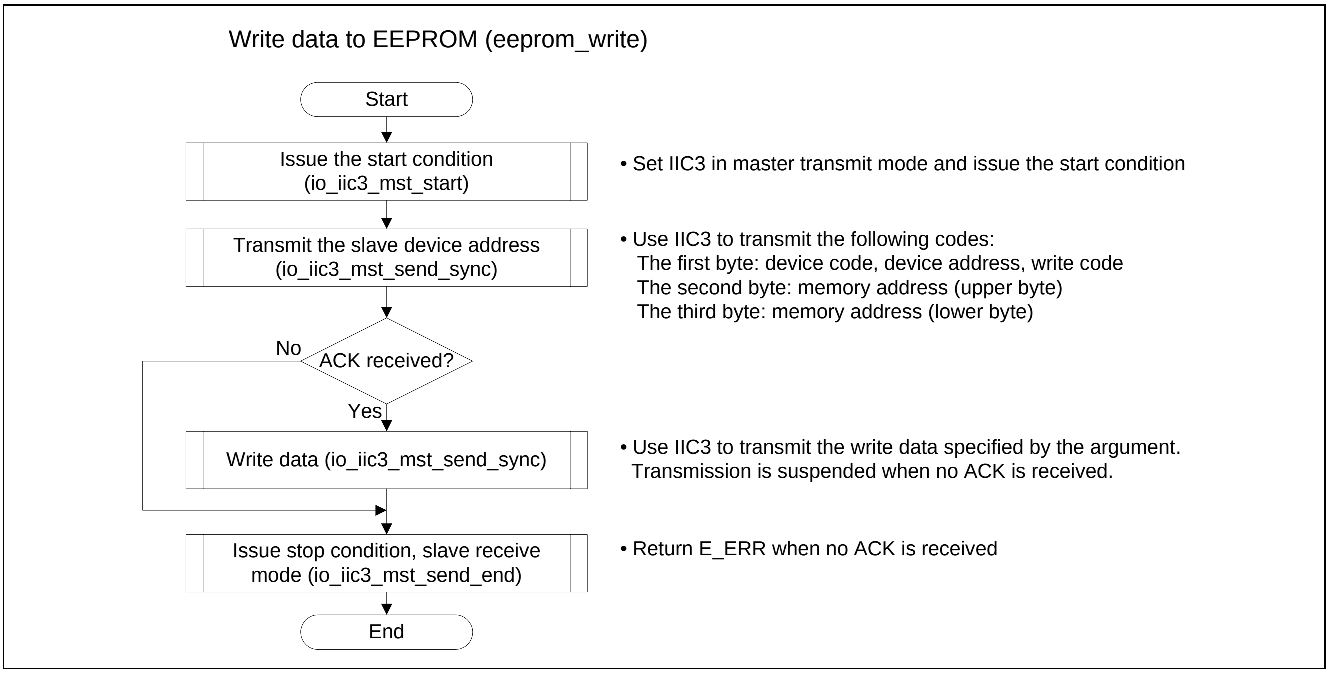

Write data to EEPROM (eeprom_write)
Start
Issue the start condition (io_iic3_mst_start)
Transmit the slave device address (io_iic3_mst_send_sync)
ACK received?
No
Yes
Write data (io_iic3_mst_send_sync)
Issue stop condition, slave receive mode (io_iic3_mst_send_end)
End
• Set IIC3 in master transmit mode and issue the start condition
• Use IIC3 to transmit the following codes:
The first byte: device code, device address, write code
The second byte: memory address (upper byte)
The third byte: memory address (lower byte)
• Use IIC3 to transmit the write data specified by the argument.
Transmission is suspended when no ACK is received.
• Return E_ERR when no ACK is received

Figure 8 Sample Program Flow Chart (2/11)

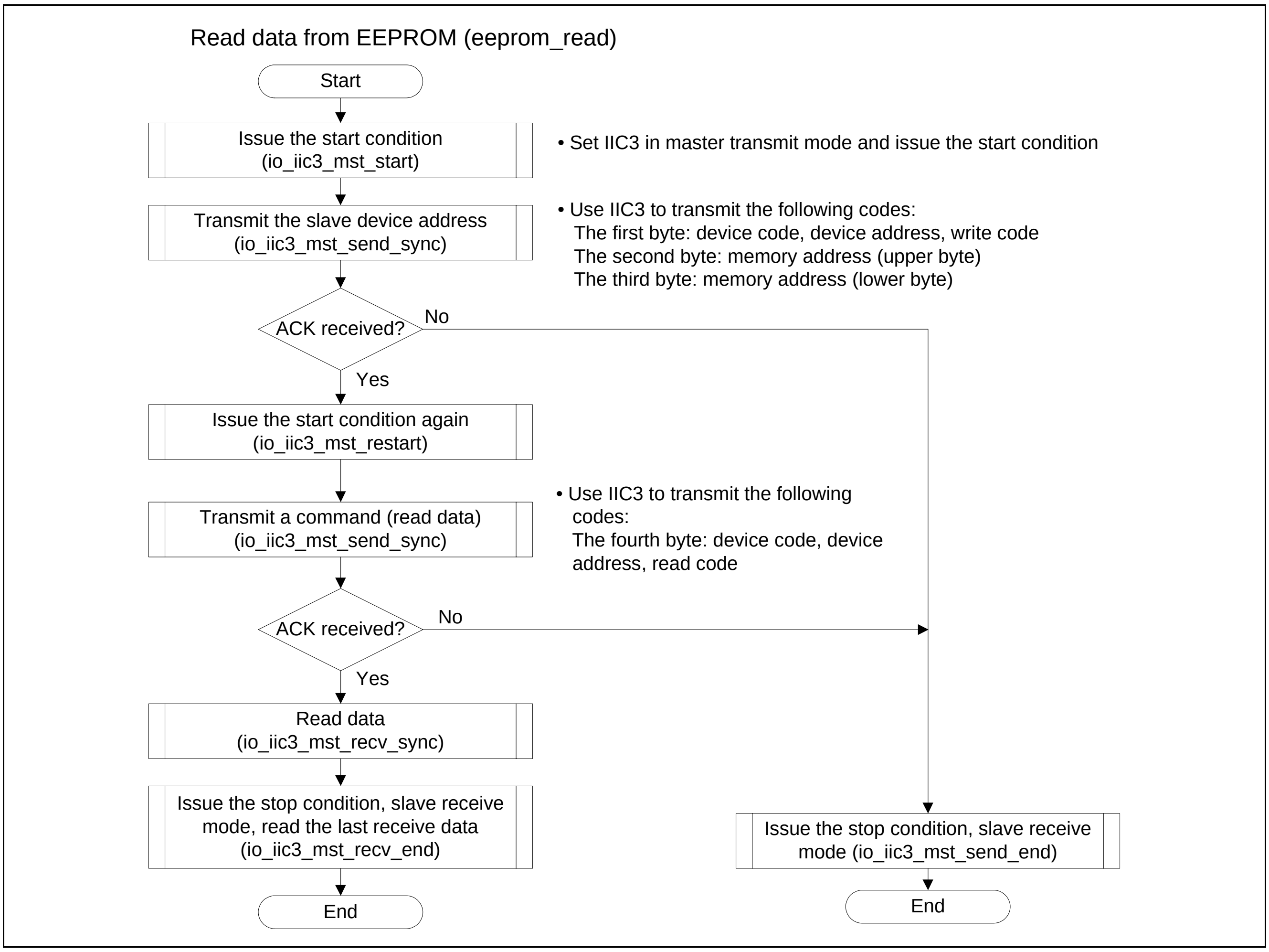

Read data from EEPROM (eeprom_read)
Start
Issue the start condition (io_iic3_mst_start)
Transmit the slave device address (io_iic3_mst_send_sync)
ACK received?
No
Yes
Issue the start condition again (io_iic3_mst_restart)
Transmit a command (read data) (io_iic3_mst_send_sync)
ACK received?
No
Yes
Read data (io_iic3_mst_recv_sync)
Issue the stop condition, slave receive mode, read the last receive data (io_iic3_mst_recv_end)
End
Issue the stop condition, slave receive mode (io_iic3_mst_send_end)
End
• Set IIC3 in master transmit mode and issue the start condition
• Use IIC3 to transmit the following codes:
The first byte: device code, device address, write code
The second byte: memory address (upper byte)
The third byte: memory address (lower byte)
• Use IIC3 to transmit the following codes:
The fourth byte: device code, device address, read code

Figure 9 Sample Program Flow Chart (3/11)

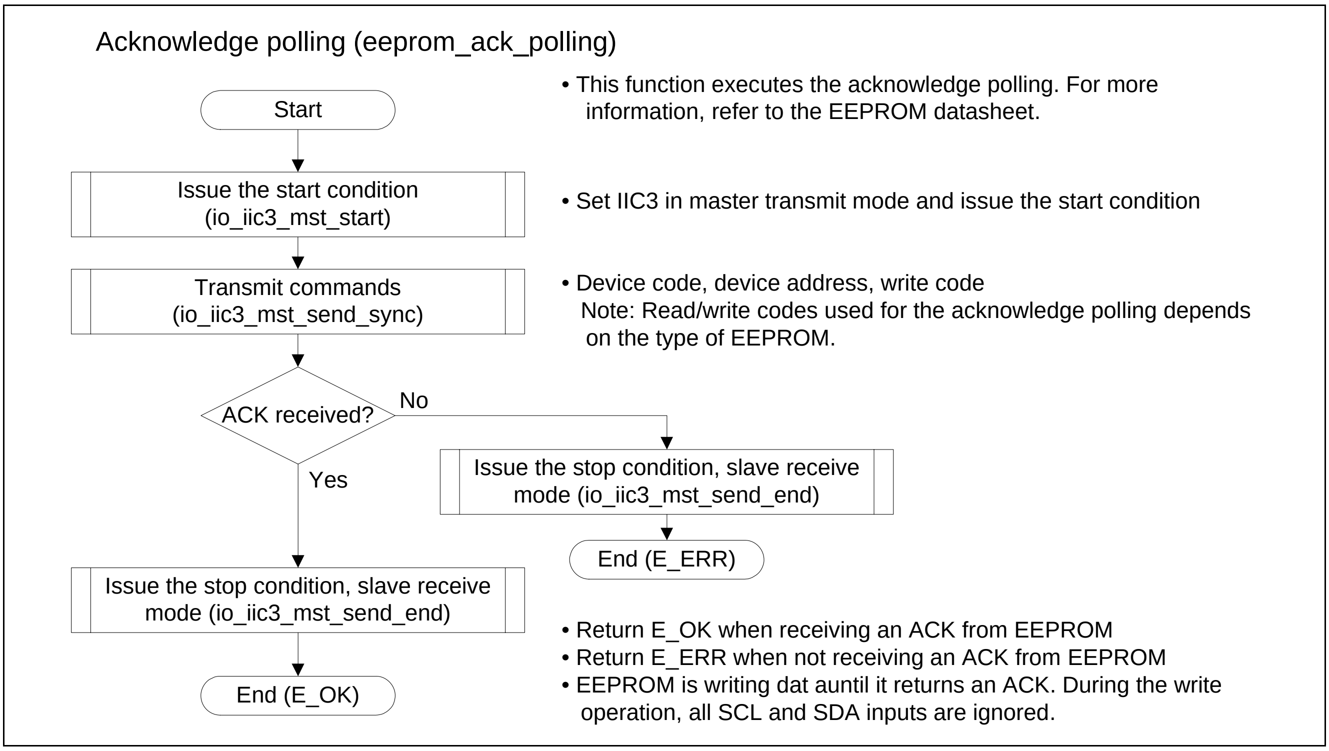

Acknowledge polling (eeprom_ack_polling)
Start
Issue the start condition (io_iic3_mst_start)
Transmit commands (io_iic3_mst_send_sync)
ACK received?
No
Yes
Issue the stop condition, slave receive mode (io_iic3_mst_send_end)
End (E_ERR)
Issue the stop condition, slave receive mode (io_iic3_mst_send_end)
End (E_OK)
• This function executes the acknowledge polling. For more information, refer to the EEPROM datasheet.
• Set IIC3 in master transmit mode and issue the start condition
• Device code, device address, write code
Note: Read/write codes used for the acknowledge polling depends on the type of EEPROM.
• Return E_OK when receiving an ACK from EEPROM
• Return E_ERR when not receiving an ACK from EEPROM
• EEPROM is writing dat auntil it returns an ACK. During the write operation, all SCL and SDA inputs are ignored.

Figure 10 Sample Program Flow Chart (4/11)

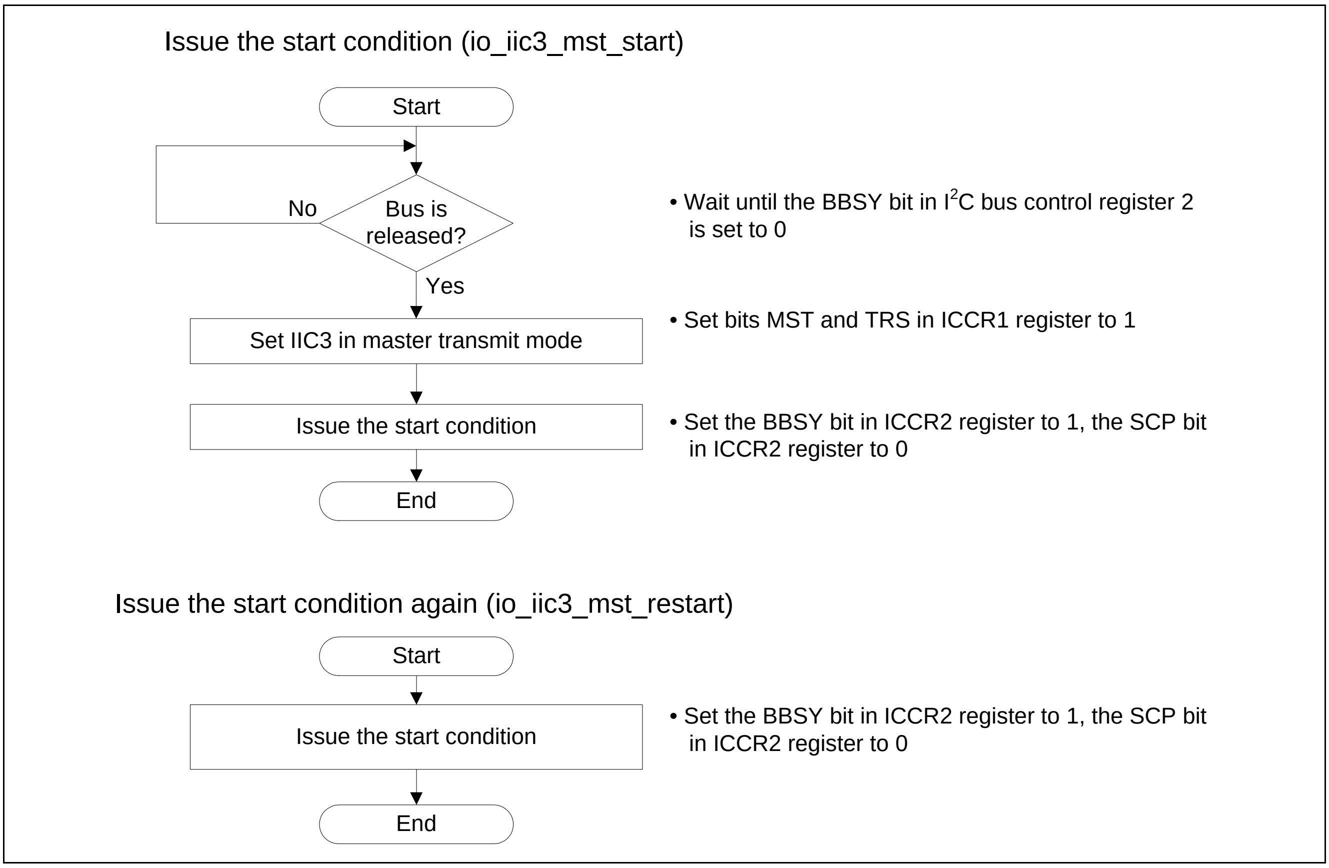
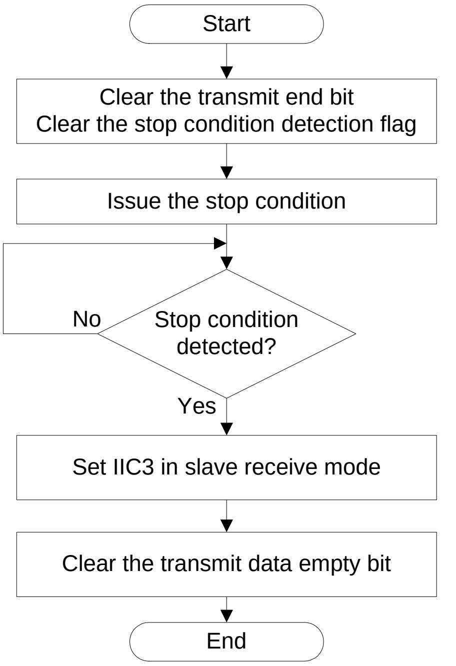

Issue the start condition (io_iic3_mst_start)
Start
No
Bus is released?
Yes
Set IIC3 in master transmit mode
Issue the start condition
End
• Wait until the BBSY bit in I2C bus control register 2 is set to 0
• Set bits MST and TRS in ICCR1 register to 1
• Set the BBSY bit in ICCR2 register to 1, the SCP bit in ICCR2 register to 0
Issue the start condition again (io_iic3_mst_restart)
Start
Issue the start condition
End
• Set the BBSY bit in ICCR2 register to 1, the SCP bit in ICCR2 register to 0

Figure 11 Sample Program Flow Chart (5/11)

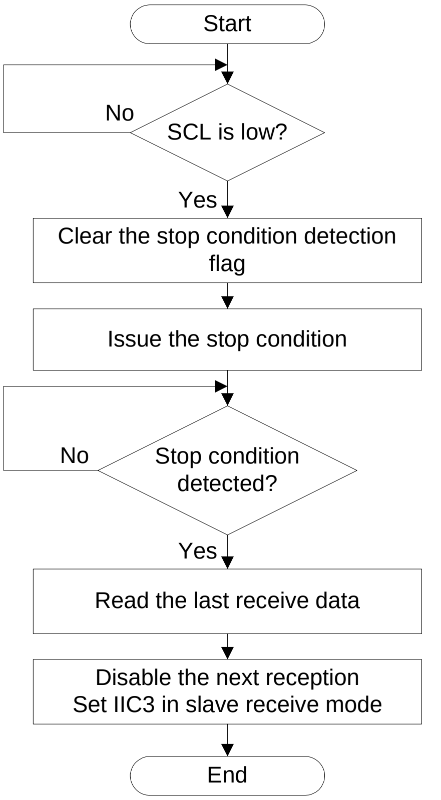
Issue the stop condition, slave receive mode (io_iic3_mst_send_end)


Start
Clear the transmit end bit
Clear the stop condition detection flag
Issue the stop condition
No
Stop condition detected?
Yes
Set IIC3 in slave receive mode
Clear the transmit data empty bit
End

- This function is used to issue the stop condition after transmitting data in master transmit mode.
- Clear bits TEND and STOP in the I^{2}C bus status register (ICSR)
- Set bits BBSY and SCP in ICCR2 register to 0
- Wait until the STOP bit is set to 1
- Set bits MST and TRS in ICCR1 register to 0
- Clear the TDRE bit in the ICSR register

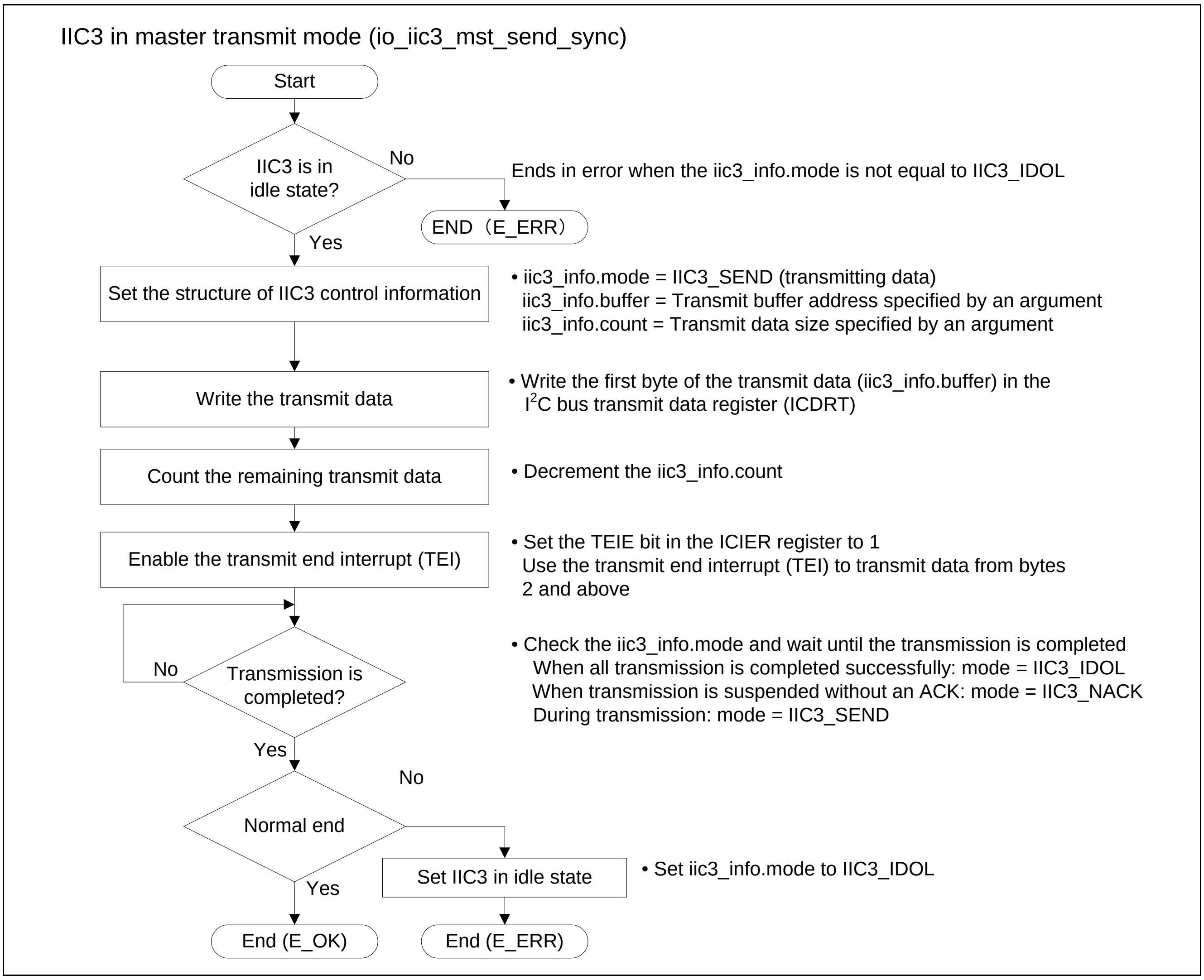
Figure 12 Sample Program Flow Chart (6/11)

Issue the stop condition, slave receive mode, read the last receive data (io_iic3_mst_recv_end)


Start
No
SCL is low?
Yes
Clear the stop condition detection flag
Issue the stop condition
No
Stop condition detected?
Yes
Read the last receive data
Disable the next reception
Set IIC3 in slave receive mode
End

- This function is used to issue the stop condition after receiving data in master receive mode.
- Wait for the falling edge of the 9th clock of SCL (Refer to the Renesas technical update [TN-MC*-A020A/E])
- Clear bits TEND and STOP in the I^{2}C bus status register (ICSR)
- Set bits BBSY and SCP in ICCR2 register to 0
- Wait until the STOP bit is set to 1
- Set bits RCVD, MST, and TRS in ICCR1 register to 0

Figure 13 Sample Program Flow Chart (7/11)


IIC3 in master transmit mode (io_iic3_mst_send_sync)
Start
IIC3 is in idle state?
No
Ends in error when the iic3_info.mode is not equal to IIC3_IDOL
END（E_ERR）
Yes
Set the structure of IIC3 control information
Write the transmit data
Count the remaining transmit data
Enable the transmit end interrupt (TEI)
No
Transmission is completed?
Yes
No
Normal end
Set IIC3 in idle state
Yes
End (E_OK)
End (E_ERR)

- iic3_info.mode = IIC3_SEND (transmitting data)
  iic3_info.buffer = Transmit buffer address specified by an argument
  iic3_info.count = Transmit data size specified by an argument
- Write the first byte of the transmit data (iic3_info.buffer) in the I^2C bus transmit data register (ICDRT)
- Decrement the iic3_info.count
- Set the TEIE bit in the ICIER register to 1
  Use the transmit end interrupt (TEI) to transmit data from bytes 2 and above
- Check the iic3_info.mode and wait until the transmission is completed
  When all transmission is completed successfully: mode = IIC3_IDOL
  When transmission is suspended without an ACK: mode = IIC3_NACK
  During transmission: mode = IIC3_SEND
- Set iic3_info.mode to IIC3_IDOL

Figure 14 Sample Program Flow Chart (8/11)

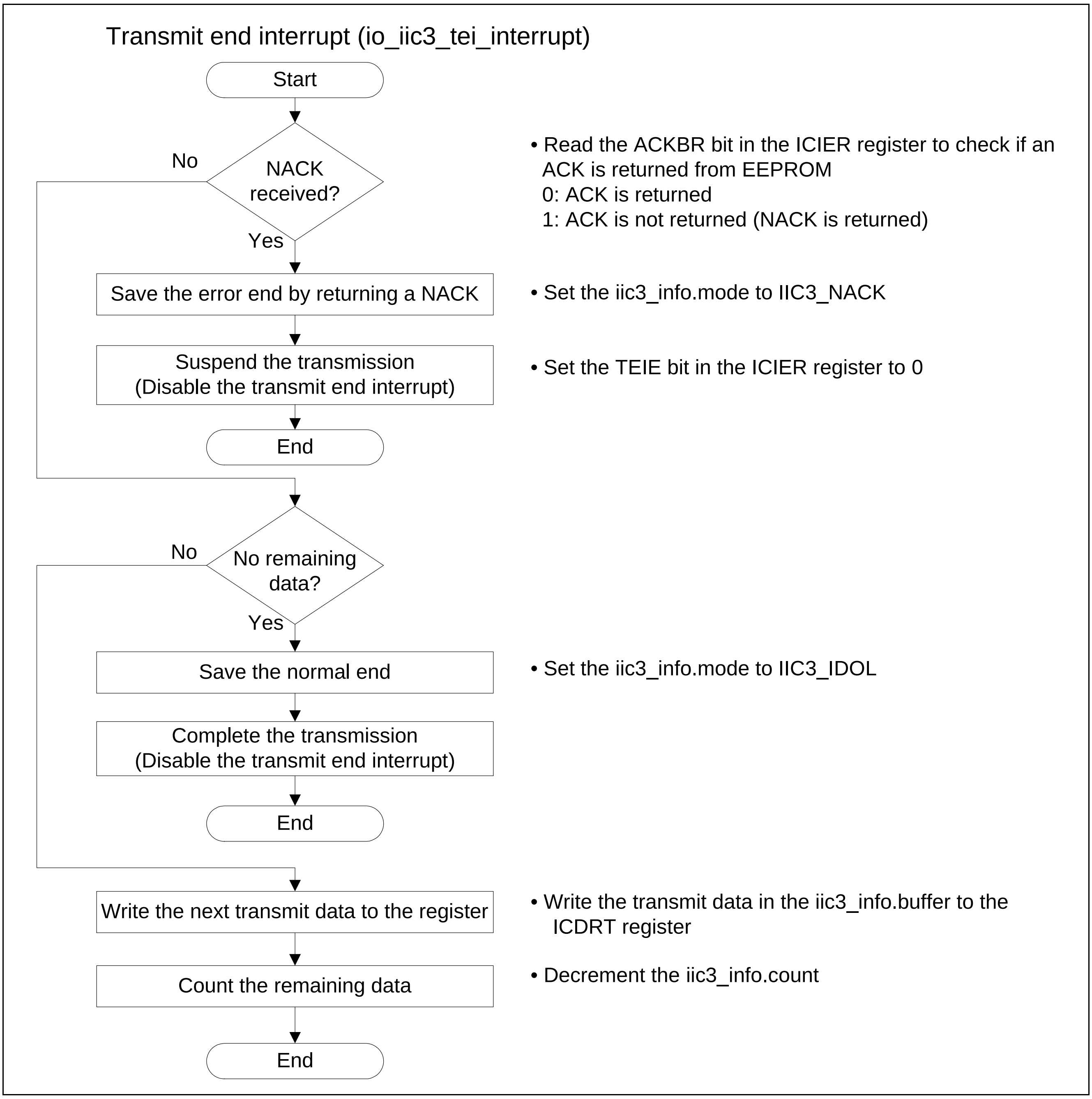
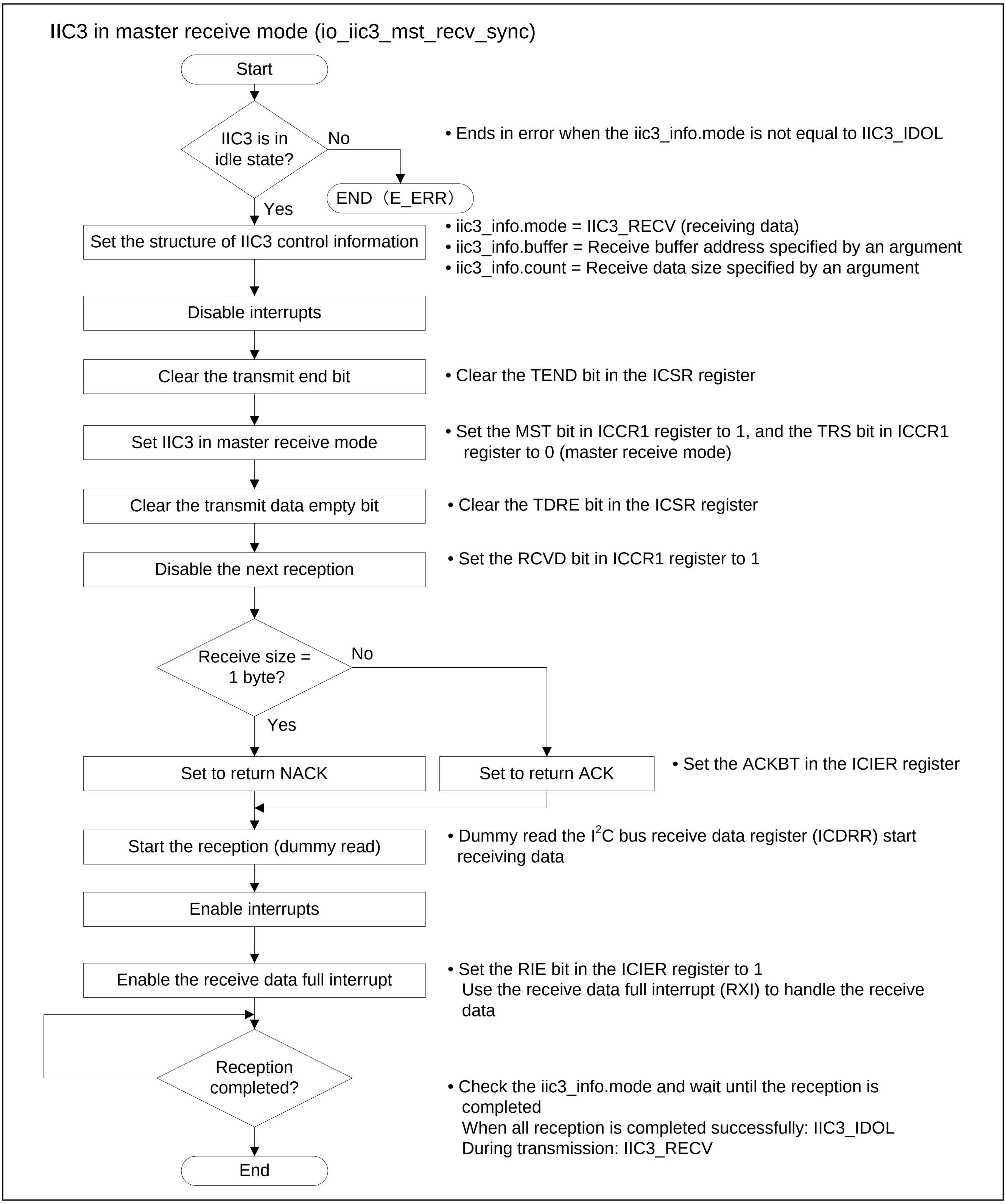

Transmit end interrupt (io_iic3_tei_interrupt)
Start
No
NACK received?
Yes
Save the error end by returning a NACK
Suspend the transmission (Disable the transmit end interrupt)
End
No
No remaining data?
Yes
Save the normal end
Complete the transmission (Disable the transmit end interrupt)
End
Write the next transmit data to the register
Count the remaining data
End
• Read the ACKBR bit in the ICIER register to check if an ACK is returned from EEPROM
0: ACK is returned
1: ACK is not returned (NACK is returned)
• Set the iic3_info.mode to IIC3_NACK
• Set the TEIE bit in the ICIER register to 0
• Set the iic3_info.mode to IIC3_IDOL
• Write the transmit data in the iic3_info.buffer to the ICDRT register
• Decrement the iic3_info.count

Figure 15 Sample Program Flow Chart (9/11)


Start
IIC3 is in idle state?
No
END（E_ERR）
Yes
Set the structure of IIC3 control information
Disable interrupts
Clear the transmit end bit
Set IIC3 in master receive mode
Clear the transmit data empty bit
Disable the next reception
Receive size = 1 byte?
No
Yes
Set to return NACK
Set to return ACK
Start the reception (dummy read)
Enable interrupts
Enable the receive data full interrupt
Reception completed?
End

IIC3 in master receive mode (io_iic3_mst_recv_sync)

- Ends in error when the iic3_info.mode is not equal to IIC3_IDOL

- iic3_info.mode = IIC3_RECV (receiving data)
- iic3_info.buffer = Receive buffer address specified by an argument
- iic3_info.count = Receive data size specified by an argument

- Clear the TEND bit in the ICSR register

- Set the MST bit in ICCR1 register to 1, and the TRS bit in ICCR1 register to 0 (master receive mode)

- Clear the TDRE bit in the ICSR register

- Set the RCVD bit in ICCR1 register to 1

- Set the ACKBT in the ICIER register

- Dummy read the I^{2}C bus receive data register (ICDRR) start receiving data

- Set the RIE bit in the ICIER register to 1
  Use the receive data full interrupt (RXI) to handle the receive data

- Check the iic3_info.mode and wait until the reception is completed
  When all reception is completed successfully: IIC3_IDOL
  During transmission: IIC3_RECV

Figure 16 Sample Program Flow Chart (10/11)

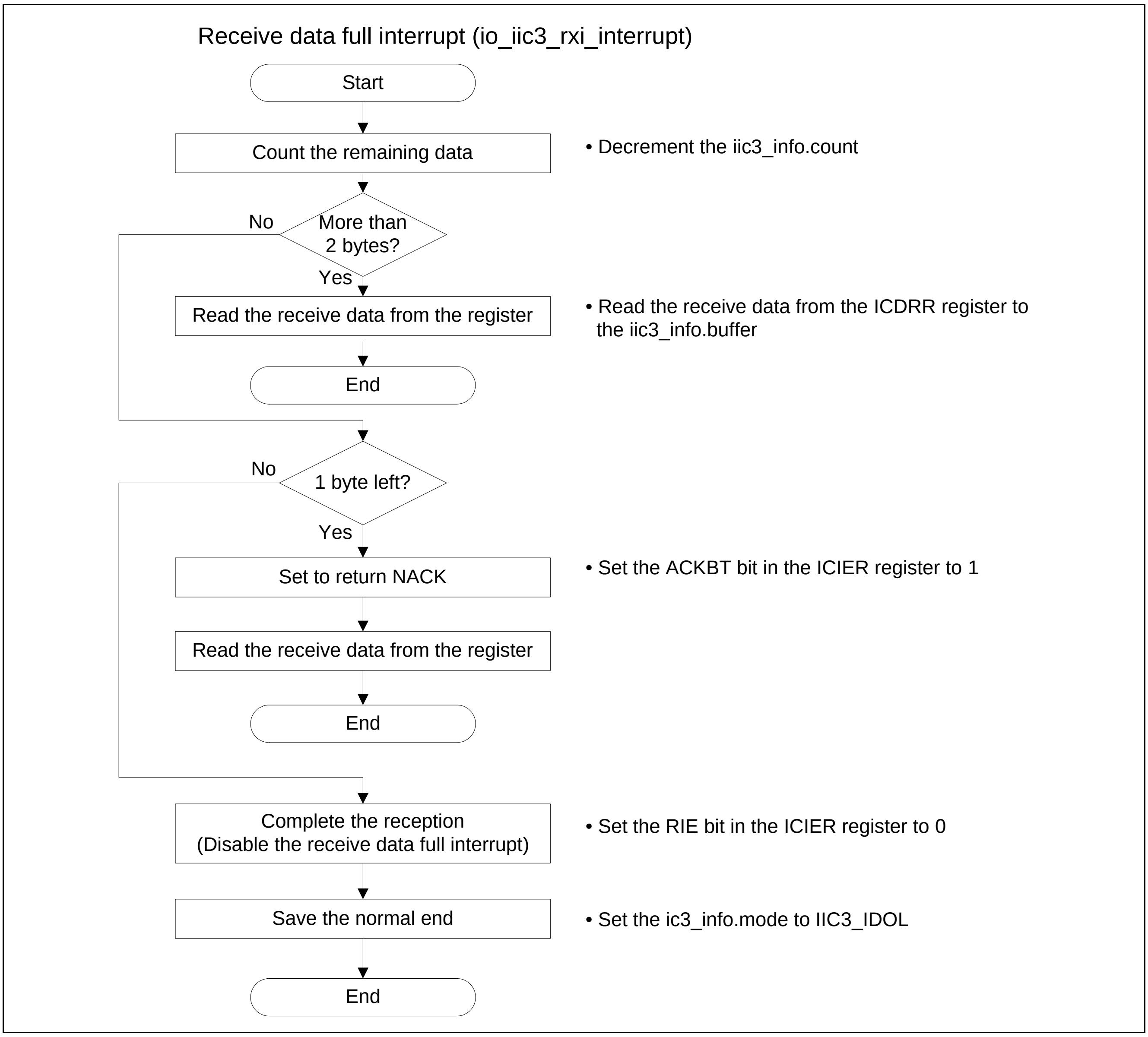

Receive data full interrupt (io_iic3_rxi_interrupt)
Start
Count the remaining data
• Decrement the iic3_info.count
No
More than 2 bytes?
Yes
Read the receive data from the register
• Read the receive data from the ICDRR register to the iic3_info.buffer
End
No
1 byte left?
Yes
Set to return NACK
• Set the ACKBT bit in the ICIER register to 1
Read the receive data from the register
End
Complete the reception (Disable the receive data full interrupt)
• Set the RIE bit in the ICIER register to 0
Save the normal end
• Set the ic3_info.mode to IIC3_IDOL
End

Figure 17 Sample Program Flow Chart (11/11)

2.5 Notes for Master Receive Mode

When reading the I^{2}C bus receive data register (ICDRR) at the falling edge of around the 8th clock, the receive data may not be retrieved.

When the receive buffer is full and specifying the receive disable bit (RCVD) in the ICDRR at the falling edge of around the 8th clock, STOP condition may not be issued. Use either one of the methods below.

The sample program sets the RCVD bit to 1 to receive a single byte at a time.

1. Read the ICDRR in master receive mode before the rising edge of the 8th clock.
2. Set the RCVD bit to 1 and receive a single byte at a time in master receive mode.

2.6 Notes for Setting the ACKBT Bit in Master Receive Mode

When IIC3 is in master receive mode, set the ACKBT bit before falling the 8th SCL signal of the last data which is continuously transferred. Otherwise, a slave device may overrun.

As the sample program sets the RCVD bit to 1 to receive a single byte at a time, this note is not applicable.

2.7 Notes for Issuing the Stop Condition or Start Condition Again in Master Receive Mode

When issuing the stop condition or start condition again at the falling edge of the SCL 9th clock, an additional cycle is output after the 9th clock. Make sure to issue the stop condition or start condition again after receiving data in master receive mode, and the falling of the SCL 9th clock.

How to make sure the falling of the SCL 9th clock:

- Check the RDRF (receive data register full) bit in the ICSR register is set to 1, and then check the SCLO bit (SCL monitor) in the ICCR2 register is set to 0 (SCL pin is low).

For more information, refer to the Renesas Technical Update (document number: TN-MC*-A020A/E).

2.8 Notes for Using the IICRST Bit

When writing 0 to the ICE bit in ICCR1 register or writing 1 to the IICRST bit in ICCR2 register while I^{2}C bus is operating, the BBSY bit in ICCR2 register and STOP bit in the ICSR register are not defined.

For more information, refer to the Renesas Technical Update (document number: TN-MC*-A022A/E).

3. Sample Program Listing

3.1 Supplement to the Sample Program

As the capacity of the SH7264 large-capacity internal RAM varies as 1 MB or 640 KB, depending on the MCU type, the section alignment and register setting must be partly altered. To support both MCU types, this application note provides two types of sample programs (workspaces) for 1-MB RAM and 640-KB RAM.

As the MCU with 640-KB RAM must be write-enabled before writing data in the data-retention RAM, the System control register 5 (SYSCR5) is set to write-enable the RAM in the sample program for 640-KB RAM.

Review your product and use the appropriate workspace.

3.2 Sample Program Listing "main.c" (1/6)

```
/*******************************************************************************
*   DISCLAIMER
*
*   This software is supplied by Renesas Electronics Corporation and is only
*   intended for use with Renesas products.  No other uses are authorized.
*
*   This software is owned by Renesas Electronics Corporation and is protected under
*   all applicable laws, including copyright laws.
*
*   THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
*   REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
*   INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
*   PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE EXPRESSLY
*   DISCLAIMED.
*
*   TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
*   ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
*   FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
*   FOR ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS
*   AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
*
*   Renesas reserves the right, without notice, to make changes to this
*   software and to discontinue the availability of this software.
*   By using this software, you agree to the additional terms and
*   conditions found by accessing the following link:
*   http://www.renesas.com/disclaimer
********************************************************************************
*   Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
*""FILE COMMENT""*********** Technical reference data **************************
*   System Name : SH7264 Sample Program
*   File Name   : main.c
*   Abstract    : Reading/Writing EEPROM Using IIC3 interrupt
*   Version     : 1.00.00
*   Device      : SH7262/SH7264
*   Tool-Chain  : High-performance Embedded Workshop (Ver.4.07.00).
*               : C/C++ compiler package for the SuperH RISC engine family
*               :                             (Ver.9.03 Release00).
*   OS          : None
*   H/W Platform: M3A-HS64G50(CPU board)
*   Description :
********************************************************************************
*   History     : Aug.17,2010 Ver.1.00.00
*""FILE COMMENT END""**********************************************************/
#include <machine.h>
#include "iodefine.h"      /* SH7264 iodefine  */

```

3.3 Sample Program Listing "main.c" (2/6)

```
/* ==== symbol definition ==== */
#define EEPROM_MEM_ADDR 0x0000
#define DEVICE_CODE 0xA0  /* EEPROM device code   :b'1010       */
#define DEVICE_ADDR 0x00  /* EEPROM device address:b'000    */
#define IIC_DATA_WR 0x00  /* Data write code      :b'0      */
#define IIC_DATA_RD 0x01  /* Data read code       :b'1      */
#define DATA_LENGTH 10

#define E_OK 0
#define E_ERR -1

/* ==== RAM allocation variable declaration ==== */
unsigned char ReadData[DATA_LENGTH];
unsigned char WriteData[DATA_LENGTH];

/* ==== prototype declaration ==== */
void main(void);
int eeprom_write(unsigned char d_code,unsigned char d_adr,unsigned short w_adr,
                                      unsigned int w_size,unsigned char* w_buf);
int eeprom_read(unsigned char d_code,unsigned char d_adr,unsigned short r_adr,
                                     unsigned int r_size,unsigned char* r_buf);
int eeprom_ack_polling(unsigned char d_code,unsigned char d_adr);
int  io_iic3_init(void);
void io_iic3_mst_start(void);
void io_iic3_mst_restart(void);
void io_iic3_mst_send_end(void);
unsigned char io_iic3_mst_recv_end(void);
int  io_iic3_mst_send_sync( unsigned char *buffer, int size);
int  io_iic3_mst_send( unsigned char *buffer, int size);
int  io_iic3_mst_recv_sync( unsigned char *buffer, int size);
int  io_iic3_mst_recv( unsigned char *buffer, int size);

/*""FUNC COMMENT""**************************************************************
 * ID            :
 * Outline       : Sample program main
 *------------------------------------------------------------------------------
 * Include       :
 *------------------------------------------------------------------------------
 * Declaration   : void main(void);
 *------------------------------------------------------------------------------
 * Description   : Writes data to EEPROM using IIC3 master transmit mode.
 *               : Reads data from EEPROM using IIC3 master receive mode.
 *               : Use the transmit end interrupt and receive data full interrupt
 *               : to transmit or receive data.
 *------------------------------------------------------------------------------
 * Argument      : void
 *------------------------------------------------------------------------------
 * Return Value  : void
 *------------------------------------------------------------------------------
 * Note          : None
 *""FUNC COMMENT END""*********************************************************/
```

3.4 Sample Program Listing "main.c" (3/6)

```
void main(void)
{
  int i,ack;

  /* ==== Clears the buffer storing the data ==== */
  for(i=0;i<DATA_LENGTH;i++){
    ReadData[i] = 0x00;
  }
  /* ==== Creates the write data ==== */
  for(i=0;i<DATA_LENGTH;i++){
    WriteData[i] = DATA_LENGTH+i;
  }

  /* ==== Configures IIC3 ==== */
  io_iic3_init();

  /* ==== Writes data to EEPROM ==== */
  eeprom_write(DEVICE_CODE,       /* Device code */
           DEVICE_ADDR,       /* Device address */
           0x0000,            /* Write start address */
           sizeof(WriteData), /* Write data size */
           WriteData);        /* Buffer storing data */

  /* ==== Acknowledge Polling ==== */
  while( eeprom_ack_polling(DEVICE_CODE, DEVICE_ADDR) != E_OK){
    /* Waits until reprogramming EEPROM internally is completed */
  }

  /* ==== Reads data from EEPROM ==== */
  eeprom_read(DEVICE_CODE,    /* Device code */
           DEVICE_ADDR,       /* Device address */
           0x0000,            /* Read start address */
           sizeof(ReadData),  /* Read data size */
           ReadData);         /* Buffer storing data */

  /* ==== Compares the result ==== */
  for(i=0; i<DATA_LENGTH; i++){
    if( WriteData[i] != ReadData[i] ){
        while(1){
            /* error */
        }
    }
  }
  while(1){
    /* Loop */
  }
}
```

3.5 Sample Program Listing "main.c" (4/6)

```
/*""FUNC COMMENT""**************************************************************
 * ID            :
 * Outline       : Write data to EEPROM
 *------------------------------------------------------------------------------
 * Include       :
 *------------------------------------------------------------------------------
 * Declaration   : int eeprom_write(unsigned char d_code,unsigned char d_adr,
 *               : unsigned short w_adr,unsigned int w_size,unsigned char* w_buf);
 *------------------------------------------------------------------------------
 * Description   : Writes the w_size bytes of data stored in the buffer specified
 *               : by the w_buf to EEPROM specified by the device code (d_code),
 *               : device address (d_adr). Specify the memory address of EEPROM by
 *               : the w_adr.
 *------------------------------------------------------------------------------
 * Argument      : unsigned char d_code ; I : Device code
 *               : unsigned char d_adr  ; I : Device address
 *               : unsigned short w_adr ; I : Write start address
 *               : unsigned int w_size  ; I : Write data size
 *               : unsigned char* w_buf ; I : Buffer storing the write data
 *------------------------------------------------------------------------------
 * Return Value  : ACK received: E_OK
 *               : ACK not received: E_ERR
 *------------------------------------------------------------------------------
 * Note          : None
 *""FUNC COMMENT END""**********************************************************/
int eeprom_write(unsigned char d_code,unsigned char d_adr,unsigned short w_adr,
                                      unsigned int w_size,unsigned char* w_buf)
{
  int ack = E_OK;
  unsigned char send[3];

  send[0] = (unsigned char)(d_code|((d_adr & 0x7)<<1)|IIC_DATA_WR);
  send[1] = (unsigned char)((w_adr>>8) & 0x00ff);
  send[2] = (unsigned char)(w_adr & 0x00ff);

  /* ==== Issues the start condition ==== */
  io_iic3_mst_start();

  /* ==== Transmits the slave device address ==== */
  ack = io_iic3_mst_send_sync( send, 3);         /* Returns an ACK after */
                                             /* transmission is completed */
  if( ack != E_OK ){
    io_iic3_mst_send_end();
    return ack;
  }
  /* ==== Writes data ==== */
  ack = io_iic3_mst_send_sync( w_buf, w_size );  /* Returns an ACK after */
                                             /* transmission is completed */

  /* ==== Issues the stop condition ==== */
  io_iic3_mst_send_end();

  return ack;
}
```

3.6 Sample Program Listing "main.c" (5/6)

```
/*""FUNC COMMENT""**************************************************************
 * ID            :
 * Outline       : Read data from EEPROM
 *------------------------------------------------------------------------------
 * Include       :
 *------------------------------------------------------------------------------
 * Declaration  : int eeprom_read(unsigned char d_code,unsigned char d_adr,
 *               : unsigned short r_adr,unsigned int r_size,unsigned char* r_buf);
 *------------------------------------------------------------------------------
 * Description  : Reads the r_size bytes of data from EEPROM specified by the
 *               : device code (d_code), device address (d_adr), and stores the
 *               : read data in the buffer specified by the r_buf. Specify the
 *               : EEPROM memory address by the r_adr.
 *------------------------------------------------------------------------------
 * Argument      : unsigned char d_code ; I : Device code
 *               : unsigned char d_adr  ; I : Device address
 *               : unsigned short r_adr ; I : Read start address
 *               : unsigned int r_size  ; I : Read data size
 *               : unsigned char* r_buf ; O : Buffer storing the read data
 *------------------------------------------------------------------------------
 * Return Value : ACK received: E_OK
 *               : ACK not received: E_ERR
 *------------------------------------------------------------------------------
 * Note          : None
 *""FUNC COMMENT END""**********************************************************/
int eeprom_read(unsigned char d_code,unsigned char d_adr,unsigned short r_adr,
                                     unsigned int r_size,unsigned char* r_buf)
{
  int ack = E_OK;
  unsigned char send[4];

  send[0] = (unsigned char)(d_code|((d_adr & 0x7)<<1)|IIC_DATA_WR);
  send[1] = (unsigned char)((r_adr>>8) & 0x00ff);
  send[2] = (unsigned char)(r_adr & 0x00ff);
  send[3] = (unsigned char)(d_code|((d_adr & 0x7)<<1)|IIC_DATA_RD);

  /* ==== Issues the start condition ==== */
  io_iic3_mst_start();

  /* ==== Transmits the slave device address ==== */
  ack = io_iic3_mst_send_sync( send, 3);         /* Returns an ACK after */
                                             /* transmission is completed */
  if( ack != E_OK ){
    io_iic3_mst_send_end();
    return ack;
  }
  /* ==== Issues the start condition again ==== */
  io_iic3_mst_restart();

  /* ==== Transmits a command (read data) ==== */
  ack = io_iic3_mst_send_sync( &send[3], 1);     /* Returns an ACK after */
                                             /* transmission is completed */
```

3.7 Sample Program Listing "main.c" (6/6)

```
  if( ack != E_OK ){
    io_iic3_mst_send_end();
    return ack;
  }
  /* ==== Reads data ==== */
  ack = io_iic3_mst_recv_sync( r_buf, r_size);   /* Returns an ACK after */
                                             /* reception is completed */

  /* ==== Issues the stop condition (reads the last data) ==== */
  r_buf[r_size - 1] = io_iic3_mst_recv_end();

  return ack;
}
/*""FUNC COMMENT""**************************************************************
 * ID            :
 * Outline       : Acknowledge Polling
 *------------------------------------------------------------------------------
 * Include       : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration   : int eeprom_ack_polling(unsigned char d_code,unsigned char d_adr);
 *------------------------------------------------------------------------------
 * Description   : This function checks if the write cycle of EEPROM is finished
 *               : or not. When the write cycle is not finished, EEPROM ignores
 *               : the input command and does not return an ACK. Make sure that
 *               : the write cycle of EEPROM is finished by this function before
 *               : accessing EEPROM. Read/Write codes to transmit upon the
 *               : Acknowledge Polling depends on the type of EEPROM. For more
 *               : information, refer to the EEPROM datasheet.
 *------------------------------------------------------------------------------
 * Argument      : unsigned char d_code ; I : Device code
 *               : unsigned char d_adr  ; I : Device address
 *------------------------------------------------------------------------------
 * Return Value  : E_OK  : NOT_BUSY
 *               : E_ERR : BUSY (EEPROM is in the write cycle).
 *------------------------------------------------------------------------------
 * Note          : None
 *""FUNC COMMENT END""**********************************************************/
int eeprom_ack_polling(unsigned char d_code,unsigned char d_adr)
{
  int ack = E_OK;
  unsigned char send[1];

  send[0] = (unsigned char)(d_code|((d_adr & 0x7)<<1)|IIC_DATA_WR);

  /* ==== Executes the Acknowledge Polling ==== */
  io_iic3_mst_start();                   /* Issues the start condition */
  ack = io_iic3_mst_send_sync( send, 1);     /* Returns an ACK after */
                                         /* transmission is completed */
  io_iic3_mst_send_end();                    /* Issues the stop condition */

  return ack;
}
/* End of File */
```

3.8 Sample Program Listing "iic3.c" (1/12)

```
/*******************************************************************************
*   DISCLAIMER
*
*   This software is supplied by Renesas Electronics Corporation and is only
*   intended for use with Renesas products.  No other uses are authorized.
*
*   This software is owned by Renesas Electronics Corporation and is protected under
*   all applicable laws, including copyright laws.
*
*   THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES
*   REGARDING THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY,
*   INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
*   PARTICULAR PURPOSE AND NON-INFRINGEMENT.  ALL SUCH WARRANTIES ARE EXPRESSLY
*   DISCLAIMED.
*
*   TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
*   ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
*   FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES
*   FOR ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS
*   AFFILIATES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
*
*   Renesas reserves the right, without notice, to make changes to this
*   software and to discontinue the availability of this software.
*   By using this software, you agree to the additional terms and
*   conditions found by accessing the following link:
*   http://www.renesas.com/disclaimer
********************************************************************************
*   Copyright (C) 2010 Renesas Electronics Corporation. All rights reserved.
*""FILE COMMENT""*********** Technical reference data **************************
*   System Name : SH7264 Sample Program
*   File Name   : iic3.c
*   Abstract    : Reading/Writing EEPROM Using IIC3 interrupt
*   Version     : 1.00.00
*   Device      : SH7262/SH7264
*   Tool-Chain  : High-performance Embedded Workshop (Ver.4.07.00).
*               : C/C++ compiler package for the SuperH RISC engine family
*               :                             (Ver.9.03 Release00).
*   OS          : None
*   H/W Platform: M3A-HS64G50(CPU board)
*   Description :
********************************************************************************
*   History     : Dec.07,2010 Ver.1.00.00
*""FILE COMMENT END""**********************************************************/
#include <machine.h>
#include "iodefine.h"        /* SH7264 iodefine  */

```

3.9 Sample Program Listing "iic3.c" (2/12)

```
/* ==== symbol definition ==== */
typedef enum{
  IIC3_IDOL=0,            /* IIC3 is in idle state */
                          /* (transmission/reception is completed) */
  IIC3_NACK,                  /* IIC3 ends with receiving NACK */
  IIC3_SEND,                  /* IIC3 is transmitting data (using transmit interrupt) */
  IIC3_RECV,                  /* IIC3 is receiving data (using receive interrupt) */
}IIC3_MODE;

typedef struct{
  volatile IIC3_MODE mode;/* IIC3 state */
  unsigned char *buffer;      /* Pointer to control the destination when transmitting or */
                          /* receiving data continuously */
  int count;                  /* Remaining data size when transmitting or */
                          /* receiving data */
}IIC3_INFO;

#define E_OK 0
#define E_ERR -1

/* ==== RAM allocation variable declaration ==== */
static IIC3_INFO iic3_info;

/* ==== prototype declaration ==== */
int  io_iic3_init(void);
void io_iic3_mst_start(void);
void io_iic3_mst_restart(void);
void io_iic3_mst_send_end(void);
unsigned char io_iic3_mst_recv_end(void);
int  io_iic3_mst_send_sync( unsigned char *buffer, int size);
int  io_iic3_mst_send( unsigned char *buffer, int size);
int  io_iic3_mst_recv_sync( unsigned char *buffer, int size);
int  io_iic3_mst_recv( unsigned char *buffer, int size);
void io_iic3_tei_interrupt(void);
void io_iic3_rxi_interrupt(void);
```

3.10 Sample Program Listing "iic3.c" (3/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID           :
 * Outline      : IIC3 configuration
 *------------------------------------------------------------------------------
 * Include      : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration  : int io_iic3_init(void);
 *------------------------------------------------------------------------------
 * Description  : Configures IIC3 channel 1.
 *------------------------------------------------------------------------------
 * Argument     : void
 *------------------------------------------------------------------------------
 * Return Value : E_OK
 *------------------------------------------------------------------------------
 * Note         : None
 *""FUNC COMMENT END""**********************************************************/
int io_iic3_init(void)
{
  /* ---- STBCR5 ---- */
  CPG.STBCR5.BIT.MSTP56   = 0;    /* IIC3 channel 1 is operating */

  /* ---- PORT ---- */
  PORT.PECR0.BIT.PE2MD = 0x01;/* SCL1 select */
  PORT.PECR0.BIT.PE3MD = 0x01;/* SDA1 select */


  /* ----IIC31 module operation enabled ---- */
  IIC3_1.ICCR1.BYTE = 0xB5;       /* IIC3 is enabled to operate */
                              /* Continues the next reception */
                              /* Master mode */
                              /* Transmit mode */
                              /* Transfer rate: P-clock/92 (391 kHz) */
  /*  ---IIC bus mode register (ICMR) setting --- */
  IIC3_1.ICMR.BYTE = 0x30;
           /*
                 bit 7: MLS:0 ---------------------  MSB first
                 bits 6 to 4: Reserve:1 -----------  Reserve bit
                 bit 3: BCWP:0---------------------  Unsetting
                 bits 2 to 0: BC0:0, BC1:0,BC0:0---  IIC format 9-bit
           */
  /* ---- Disables or enables interrupts ---- */
  IIC3_1.ICIER.BYTE = 0x00;       /* Disables all IIC3 interrupts */
  INTC.IPR16.BIT._IIC31 = 5;      /* Sets the interrupt priority to 5 */

  /* ---- Initializes the manage information ---- */
  iic3_info.mode   = IIC3_IDOL;
  iic3_info.buffer = (void *)0;
  iic3_info.count  = 0;

  return(E_OK);
}
```

3.11 Sample Program Listing "iic3.c" (4/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID             :
 * Outline        : Issues the start condition
 *------------------------------------------------------------------------------
 * Include        : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration    : void io_iic3_mst_start(void);
 *------------------------------------------------------------------------------
 * Description    : Issues the start condition.
 *                : Before issuing the start condition, it checks the bus is released
 *                : and sets IIC3 in master transmit mode.
 *------------------------------------------------------------------------------
 * Argument       : void
 *------------------------------------------------------------------------------
 * Return Value   : void
 *------------------------------------------------------------------------------
 * Note           : None
 *""FUNC COMMENT END""**********************************************************/
void io_iic3_mst_start(void)
{
  while(IIC3_1.ICCR2.BIT.BBSY == 1){
    /* Waits until the bus is released */
  }
  IIC3_1.ICCR1.BYTE |= 0x30;                     /* Sets IIC3 in master transmit mode */
  IIC3_1.ICCR2.BYTE = ((IIC3_1.ICCR2.BYTE & 0xbf)|0x80); /* Issues the start condition */
}
/*""FUNC COMMENT""**************************************************************
 * ID             :
 * Outline        : Issues the start condition again
 *------------------------------------------------------------------------------
 * Include        : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration    : void io_iic3_mst_restart(void);
 *------------------------------------------------------------------------------
 * Description    : Issues the start condition again. Other processing is not included.
 *------------------------------------------------------------------------------
 * Argument       : void
 *------------------------------------------------------------------------------
 * Return Value   : void
 *------------------------------------------------------------------------------
 * Note           : None
 *""FUNC COMMENT END""**********************************************************/
void io_iic3_mst_restart(void)
{
  IIC3_1.ICCR2.BYTE = ((IIC3_1.ICCR2.BYTE & 0xbf)|0x80); /* Issues the start condition */
}
```

3.12 Sample Program Listing "iic3.c" (5/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID            :
 * Outline       : Issue the stop condition
 *------------------------------------------------------------------------------
 * Include       : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration   : void io_iic3_mst_send_end(void);
 *------------------------------------------------------------------------------
 * Description   : Issues the stop condition and sets IIC3 in slave receive mode.
 *               : This function is used to issue the stop condition after
 *               : transmitting data in master transmit mode.
 *------------------------------------------------------------------------------
 * Argument      : void
 *------------------------------------------------------------------------------
 * Return Value  : void
 *------------------------------------------------------------------------------
 * Note          : None
 *""FUNC COMMENT END""**********************************************************/
void io_iic3_mst_send_end(void)
{
  /* ==== Issues the stop condition ==== */
  IIC3_1.ICSR.BIT.TEND = 0;       /* Clears the TEND flag */
  IIC3_1.ICSR.BIT.STOP = 0;       /* Clears the STOP flag */
  IIC3_1.ICCR2.BYTE &= 0x3f;      /* Issues the stop condition */

  /* ==== Waits until the bus is released ==== */
  while(IIC3_1.ICSR.BIT.STOP == 0){
    /* wait */
  }
  /* ==== Sets IIC3 in slave receive mode ==== */
  IIC3_1.ICCR1.BYTE &= 0xcf;      /* Slave receive mode */
  IIC3_1.ICSR.BIT.TDRE = 0;       /* Clears the TDRE bit */
}
```

3.13 Sample Program Listing "iic3.c" (6/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID          :
 * Outline     : Issue the stop condition
 *------------------------------------------------------------------------------
 * Include     : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration : unsigned char io_iic3_mst_recv_end(void);
 *------------------------------------------------------------------------------
 * Description : Issues the stop condition and sets IIC3 in slave receive mode.
 *             : This function is used to issue the stop condition after
 *             : receiving data in master receive mode.
 *------------------------------------------------------------------------------
 * Argument    : void
 *------------------------------------------------------------------------------
 * Return Value : The last receive data
 *------------------------------------------------------------------------------
 * Note        : None
 *""FUNC COMMENT END""**********************************************************/
unsigned char io_iic3_mst_recv_end(void)
{
  unsigned char data;

  /* ==== Waits for the falling edge of the SCL 9th clock ==== */
  while(IIC3_1.ICCR2.BIT.SCLO == 1){  /* Technical Update [TN-MC*-A020A/E] */
    /* wait */
  }
  /* ==== Issues the stop condition ==== */
  IIC3_1.ICSR.BIT.STOP = 0;           /* Clears the STOP flag */
  IIC3_1.ICCR2.BYTE &= 0x3f;          /* Issues the stop condition */

  /* ==== Waits until the bus is released ==== */
  while(IIC3_1.ICSR.BIT.STOP == 0){
    /* wait */
  }
  /* ==== Reads the last byte of data ==== */
  data = IIC3_1.ICDRR;            /* Read the last data from the register */

  /* ==== Sets IIC3 in slave receive mode again ==== */
  IIC3_1.ICCR1.BIT.RCVD = 0;          /* Clears the RCVD bit */
  IIC3_1.ICCR1.BYTE &= 0xcf;          /* Slave receive mode */

  return data;
}
```

3.14 Sample Program Listing "iic3.c" (7/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID             :
 * Outline        : Transmission in master mode (synchronous).
 *------------------------------------------------------------------------------
 * Include        : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration    : int io_iic3_mst_send_sync( unsigned char *buffer, int size);
 *------------------------------------------------------------------------------
 * Description    : Transmits the number of bytes of data specified by the argument
 *                : size from the address specified by the argument buffer in master
 *                : transmit mode. Uses the transmit end interrupt (TEI) to transmit
 *                : data. After transmitting data is completed, it returns to the caller.
 *------------------------------------------------------------------------------
 * Argument       : unsigned char *buffer ; I : Buffer storing the transmit data
 *                : int            size   ; I : Transmit data size
 *------------------------------------------------------------------------------
 * Return Value : ACK received: E_OK
 *                : ACK not received: E_ERR
 *------------------------------------------------------------------------------
 * Note           : None
 *""FUNC COMMENT END""**********************************************************/
int io_iic3_mst_send_sync( unsigned char *buffer, int size)
{
  int ack = E_OK;

  /* ==== Starts transmission in master transmit mode ==== */
  ack = io_iic3_mst_send( buffer, size);
  if( ack == E_OK ){
    /* ==== Waits until the transmission is completed ==== */
    while( iic3_info.mode == IIC3_SEND ){
        /* wait */
    }
    /* ==== Ends in error when receiving a NACK ==== */
    if( iic3_info.mode == IIC3_NACK ){
        iic3_info.mode = IIC3_IDOL;
        ack = E_ERR;
    }
  }
  return ack;
}
```

3.15 Sample Program Listing "iic3.c" (8/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID             :
 * Outline        : Transmission in master mode (asynchronous).
 *-------------------------------------------------------------------------------
 * Include        : iodefine.h
 *-------------------------------------------------------------------------------
 * Declaration    : int io_iic3_mst_send( unsigned char *buffer, int size);
 *-------------------------------------------------------------------------------
 * Description    : Transmits the number of bytes of data specified by the argument
 *                : size from the address specified by the argument buffer in master
 *                : transmit mode. Uses the transmit end interrupt (TEI) to transmit
 *                : data. It returns the caller before the transmission is completed.
 *-------------------------------------------------------------------------------
 * Argument       : unsigned char *buffer ; I : Buffer storing the transmit data
 *                : int            size   ; I : Transmit data size
 *-------------------------------------------------------------------------------
 * Return Value :  ACK received: E_OK
 *              :  ACK not received: E_ERR
 *-------------------------------------------------------------------------------
 * Note           : None
 *""FUNC COMMENT END""**********************************************************/
int io_iic3_mst_send( unsigned char *buffer, int size)
{
  /* ==== Make sure that IIC3 is not transferring data ==== */
  if( iic3_info.mode != IIC3_IDOL ){
    return E_ERR;
  }
  /* ==== Sets the structure of IIC3 control information ==== */
  iic3_info.mode   = IIC3_SEND;
  iic3_info.buffer = buffer;
  iic3_info.count  = size;

  /* ==== Starts transmitting data ==== */
  IIC3_1.ICDRT = *(iic3_info.buffer)++;
  iic3_info.count--;

  /* ==== Enables the transmit end interrupt ==== */
  IIC3_1.ICIER.BIT.TEIE = 1;

  return E_OK;
}
```

3.16 Sample Program Listing "iic3.c" (9/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID           :
 * Outline      : Reception in master mode (synchronous).
 *------------------------------------------------------------------------------
 * Include      : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration  : int io_iic3_mst_recv_sync( unsigned char *buffer, int size);
 *------------------------------------------------------------------------------
 * Description  : Receives the number of bytes of data specified by the argument
 *              : size to the address specified by the argument buffer in master
 *              : receive mode. Uses the receive data full interrupt (RXI).
 *              : After receiving data is completed, it returns to the caller.
 *------------------------------------------------------------------------------
 * Argument     : unsigned char *buffer ; O : Buffer storing the receive data
 *              : int            size   ; I : Receive data size
 *------------------------------------------------------------------------------
 * Return Value : Normal end: E_OK
 *              : Error end: E_ERR
 *------------------------------------------------------------------------------
 * Note         : None
 *""FUNC COMMENT END""**********************************************************/
int io_iic3_mst_recv_sync( unsigned char *buffer, int size)
{
  int ack = E_OK;

  /* ==== Starts reception in master receive mode ==== */
  ack = io_iic3_mst_recv( buffer, size);
  if( ack == E_OK ){
    /* ==== Waits until the reception is completed ==== */
    while( iic3_info.mode == IIC3_RECV ){
        /* wait */
    }
  }
  return ack;
}
```

3.17 Sample Program Listing "iic3.c" (10/12)

```
/*""FUNC COMMENT""**************************************************************
 * ID           :
 * Outline      : Reception in master mode (asynchronous).
 *------------------------------------------------------------------------------
 * Include      : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration  : int io_iic3_mst_recv( unsigned char *buffer, int size);
 *------------------------------------------------------------------------------
 * Description  : Receives the number of bytes of data specified by the argument
 *              : size to the address specified by the argument buffer in master
 *              : receive mode. Uses the receive data full interrupt (RXI).
 *              : It returns the caller before the reception is completed.
 *------------------------------------------------------------------------------
 * Argument     : unsigned char *buffer ; O : Buffer storing the receive data
 *              : int            size   ; I : Receive data size
 *------------------------------------------------------------------------------
 * Return Value : Normal end: E_OK
 *              : Error end: E_ERR
 *------------------------------------------------------------------------------
 * Note         : None
 *""FUNC COMMENT END""*********************************************************/
int io_iic3_mst_recv( unsigned char *buffer, int size)
{
  int mask;
  unsigned char dummy;

  /* ==== Make sure that IIC3 is not transferring data ==== */
  if( iic3_info.mode != IIC3_IDOL ){
    return E_ERR;
  }
  /* ==== Sets the structure of IIC3 control information ==== */
  iic3_info.mode   = IIC3_RECV;
  iic3_info.buffer = buffer;
  iic3_info.count  = size;

  mask = get_imask();
  set_imask(15);                      /* Interrupts are disabled */

  /* ==== Sets IIC3 in master receive mode (noncontiguous reception) ==== */
  IIC3_1.ICSR.BIT.TEND = 0;           /* Clears the TEND bit */
  IIC3_1.ICCR1.BIT.MST = 1;           /* Master mode */
  IIC3_1.ICCR1.BIT.TRS = 0;           /* Receive mode */
  IIC3_1.ICSR.BIT.TDRE = 0;           /* Clears the TDRE bit */
  IIC3_1.ICCR1.BIT.RCVD = 1;          /* Disables the next reception */

```

3.18 Sample Program Listing "iic3.c" (11/12)

```
  /* ==== Sets an acknowledge ==== */
  if(iic3_info.count == 1){           /* When receiving a single byte */
    IIC3_1.ICIER.BIT.ACKBT = 1;       /* sets acknowledge to high */
  }
  else{
    IIC3_1.ICIER.BIT.ACKBT = 0;       /* Sets acknowledge to low */
  }
  /* ==== Starts receiving data ==== */
  dummy = IIC3_1.ICDRR;               /* Dummy read */
  set_imask(mask);                /* Interrupts are enabled */

  /* ==== Enables the receive data full interrupt ==== */
  IIC3_1.ICIER.BIT.RIE = 1;

  return E_OK;
}
/*""FUNC COMMENT""**************************************************************
 * ID           :
 * Outline      : Transmit end interrupt (TEI).
 *------------------------------------------------------------------------------
 * Include      : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration  : void io_iic3_tei_interrupt(void);
 *------------------------------------------------------------------------------
 * Description  : Execute this function when a transmit end interrupt (TEI) occurs.
 *              : When an ACK is not returned, sets IIC3_NACK to finish the
 *              : transmission. When all transmission is completed successfully,
 *              : sets IIC3_IDOL to finish the transmission.
 *              : Continues transmitting data when the transmit data is left.
 *------------------------------------------------------------------------------
 * Argument     : void
 *------------------------------------------------------------------------------
 * Return Value : void
 *------------------------------------------------------------------------------
 * Note         : None
 *""FUNC COMMENT END""**********************************************************/
void io_iic3_tei_interrupt(void)
{
  unsigned char dummy;
  /* ==== Transmission end when receiving a NACK ==== */
  if(IIC3_1.ICIER.BIT.ACKBR == 1){
    iic3_info.mode = IIC3_NACK;
    IIC3_1.ICIER.BIT.TEIE = 0;            /* Disables the transmit end interrupt */
  }
  /* ==== Continues transmission when there is remaining data ==== */
  else if( iic3_info.count > 0 ){
    IIC3_1.ICDRT = *(iic3_info.buffer)++;
    iic3_info.count--;
  }
```

3.19 Sample Program Listing "iic3.c" (12/12)

```
  /* ==== Normal end when all transmission is completed ==== */
  else{
    IIC3_1.ICIER.BIT.TEIE = 0;            /* Disables the transmit end interrupt */
    iic3_info.mode = IIC3_IDOL;
  }
  dummy = IIC3_1.ICSR.BYTE;
}
/*""FUNC COMMENT""**************************************************************
 * ID            :
 * Outline       : Receive data full interrupt (RXI).
 *------------------------------------------------------------------------------
 * Include       : iodefine.h
 *------------------------------------------------------------------------------
 * Declaration   : void io_iic3_rxi_interrupt(void);
 *------------------------------------------------------------------------------
 * Description   : Execute this function when the receive data full interrupt (RXI)
 *               : occurs. When there are more than 2 bytes of remaining data,
 *               : it continues receiving data. When there is a single byte of data
 *               : left, it sets to return a NACK and continues receiving data.
 *               : It finishes reception when receiving all data is completed.
 *               : As this function does not read the last data, read the last data
 *               : after issuing the stop condition.
 *------------------------------------------------------------------------------
 * Argument      : void
 *------------------------------------------------------------------------------
 * Return Value  : void
 *------------------------------------------------------------------------------
 * Note          : None
 *""FUNC COMMENT END""**********************************************************/
void io_iic3_rxi_interrupt(void)
{
  unsigned char dummy;
  /* ==== Counts the remaining data ==== */
  iic3_info.count--;

  /* ==== More than 2 bytes of remaining data ==== */
  if( iic3_info.count >= 2 ){
    *(iic3_info.buffer)++ = IIC3_1.ICDRR; /* Reads the receive data from the register */
  }
  /* ==== Only a single byte of data left ==== */
  else if( iic3_info.count == 1 ){
    IIC3_1.ICIER.BIT.ACKBT = 1;               /* Sets acknowledge to high */
    *(iic3_info.buffer)++ = IIC3_1.ICDRR; /* Reads the receive data from the register */
  }
  /* ====  Normal end when all transmission is completed ==== */
  else{
    IIC3_1.ICIER.BIT.RIE = 0;             /* Disables the receive data full interrupt */
    iic3_info.mode = IIC3_IDOL;
  }
  dummy = IIC3_1.ICSR.BYTE;
}
/* End of File */
```

4. References

- Software Manual
  SH-2A/SH2A-FPU Software Manual Rev. 3.00
  The latest version of the software manual can be downloaded from the Renesas Electronics website.

- Hardware Manual
  SH7262 Group, SH7264 Group Hardware User’s Manual Rev. 2.00
  The latest version of the hardware manual can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Dec.7.10	—	First edition issued
1.01	Feb.10.12	17	Description amended 2.5 Notes for Master Receive Mode 1. Read the ICDRR bit in master receive mode before the ***rising*** edge of the 8th clock

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this manual, refer to the relevant sections of the manual. If the descriptions under General Precautions in the Handling of MPU/MCU Products and in the body of the manual differ from each other, the description in the body of the manual takes precedence.

1. Handling of Unused Pins
   Handle unused pins in accord 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 one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of 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. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. 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 of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may 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.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, 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, please evaluate the safety of the final products or system 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. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be 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, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

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

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.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

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

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, 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 Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.1