To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

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

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

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.

H8/300H Tiny Series

Application Examples for Reading from/Writing to Serial EEPROM

Introduction

This application note gives examples of reading from/writing to two-wire serial (I^2C bus) EEPROM via the on-chip I^2C module 2 (I^2C) of the H8/3694 or H8/3687.

Target Device

H8/3694, H8/3687

Contents

1. Specifications

This application is for the H8/3694 and H8/3687.

From now on in this text, the H8/3694 is used as a representative target product name.

This application note introduces specific application examples for reading from/writing to two-wire serial (I^2C bus) EEPROM using the on-chip I^2C module 2 (IIC2) of the H8/3694 (here, the H8/3694 is assumed as the master, and EEPROM as a slave).

Please make use of this application note as a reference in addition to the hardware manual for programming with the I^2C module 2 (IIC2).

The sample programs for this application note are written in C language.

The functions in the sample programs are those typically required for using I^2C module 2 (IIC2) and have been created for versatile use; so they can be reused as service modules.

It is assumed that the serial (I^2C bus) EEPROM in this application requires its memory address to be specified in two bytes.

The operation of this application has been confirmed with the EEPROM products shown in table 1.1.1.

Since a standard bus format is employed, it is considered that this application may run with other I^2C bus-compliant products.

Figure 1.1.1 shows the operating environment (connection diagram) for this application.

Table 1.1.1 Operation-Confirmed EEPROM Products with This Application

No.	Operation-Confirmed Products	Manufacturer	Capacity [byte]	Page Size [byte]
1	H8/3694N on-chip EEPROM	RENESAS	512	8
2	H8/3687N on-chip EEPROM	RENESAS	512	8
3	HN58X2432I	RENESAS	4,096	32
4	HN58X2464I	RENESAS	8,192	32

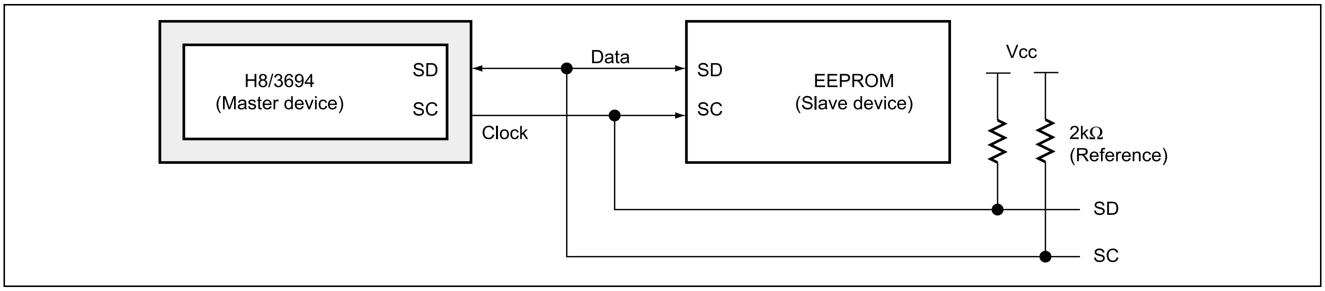


Figure 1.1.1 Operating Environment for this Application (Connection Diagram)

2. Bus Specifications

1. On-chip I^{2}C bus interface 2 (IIC2)
   The on-chip I^{2}C bus interface (IIC2) of the H8/3694 conforms to the I^{2}C bus (inter IC bus) interface method propounded by Philips, and embeds sub-set functions.
   For details, refer to I^{2}C Bus Interface 2 (IIC2) section in the Hardware Manual.

2. I^{2}C bus interface of EEPROM
   For the I^{2}C bus interface of EEPROM, refer to the Hardware Manual of the EEPROM.

3. Bus mode
   This application performs communications through the I^{2}C bus with the configuration of one master device (H8/3694), on which this application program runs, and one slave device (EEPROM) (see figure 1.1.1).
   Among transmission/reception modes of the I^{2}C bus interface 2, only the master transmission/reception mode is used in this application.

3. EEPROM Basic Specifications

For the I^{2}C bus interface specifications of the on-chip EEPROM of the H8/3694N, refer to the Hardware Manual of the product.

According to the I^{2}C bus interface specifications, there are following four types of standard specifications (read/write specifications) for accessing to EEPROM having an I^{2}C bus interface.

This application note describes operations of the four types of basic specifications (read/write specifications) through functionalization with versatility.

1. Read specifications
   A. Random read (1-byte read)
      Data is read out from a given address.
   B. Sequential read (N-byte read)
      N-byte data is sequentially read out starting from a given address.

2. Write specifications
   A. Byte write (1-byte write)
      Data is written to a given address.
   B. Page write (N-byte write)
      One page of (N-byte) data is sequentially written to starting from a given address.
      In writing, after confirming the completion of the writing operation by acknowledgement polling, the next data write is performed.

4. Software Specifications

4.1 Software Specifications

The software specifications of this application are described below.

4.1.1 Basic functions

The basic functions listed in table 4.1.1 have been provided for reading from/writing to serial EEPROM via the I^{2}C bus interface 2 (IIC2) embedded in the H8/3694.

Table 4.1.1 Basic Functions for Read/Write Access to Serial EEPROM

No.	Function	Function Name
1	1-byte write	Master_byte_write(Device_id, Addr, Data)
2	1-byte read	Master_read_byte_random(Device_id, Addr)
3	1-page write (N-byte write)	Master_page_write(Device_id, Addr, Length, *buff_ptr)
4	Sequential read (N-byte read)	Master_read_sequential(Device_id, Addr, Length, *buff_ptr)

4.1.2 Functional policy of this application

To perform data communications via the I^{2}C bus interface 2 (IIC2), processing is done through reading/writing of flags and data in the internal registers by software.

In this application, read/write processing of flags and data of internal registers is grouped into functional units and made into functions for each functional unit to describe the operation of the I^{2}C bus interface 2 (IIC2).

4.1.3 Slave address of the EEPROM

The slave address (7 bits) of the EEPROM to be connected as a slave device is assigned to be 0xa0 (upper 7 bits are used). For details, see table 4.1.2.

Table 4.1.2 Slave Address (7 Bits) Assignment of the EEPROM to be Connected

Bit	Bit Name	Setting
7	Device code D3	1
6	Device code D2	0
5	Device code D1	1
4	Device code D0	0
3	Slave address code A2	0
2	Slave address code A1	0
1	Slave address code A0	0

Note: Bit 0 is used as a R/W code.

4.1.4 Transfer method

Data transfer through the I^{2}C bus is carried out by performing communications while confirming the transfer through an acknowledgement for every one-byte transmission/reception.

Alternatively, transfer processing can be performed using interrupts. For details, see table 4.1.4 and the Hardware Manual of the target product.

4.1.5 Timeout processing for individual control flags

In the control of I^{2}C bus interface 2 (IIC2), it is necessary to check individual control flags repeatedly until they become active.

In this application, a timeout function is added as part of control flag checking functionality.

When the flag check count exceeds the limit value, it is regarded that a timeout error has occurred, so that the process exits from the flag check loop and error processing is performed.

When returning to the upper function, a timeout error code should be returned.

For details on the timeout limit counter value and error code, see table 4.3.2, Constants 2. and 3. .

In the checking of respective control flags, the minimum time for the flag becoming active is determined by the internal logic.

The minimum time until a timeout occurs for each flag is shown in table 4.1.3 for reference. When performing timeout processing, the limit count (retry count) values should be set referring to these minimum times as a guide and then be adjusted.

Note that the minimum time is a value when the transfer clock does not go rusty. When the transfer clock dulls, the value may be larger. Therefore, a margin should be taken according to the environment for use.

In this application, the values shown in table 4.3.2, Constants 2., are used as assumed values for explanation.

Table 4.1.3 Minimum Time till Timeout of Each Processing (when transfer clock does not dull)

No.	Control Flag Check	Target Control Flag Name where Timeout Processing is Required	Minimum Time till Timeout (Transfer Clock Count)
1	Timeout of bus busy check	IIC2.ICCR2.BIT.BBSY	2
2	Timeout of ACK check	IIC2.ICIER.BIT.ACKBR	21
3	Timeout of TEND check	IIC2.ICSR.BIT.TEND	21
4	Timeout of RDRF check	IIC2.ICSR.BIT.RDRF	21
5	Timeout of STOP check	IIC2.ICSR.BIT.STOP	2

Note: One transfer clock cycle = (1/transfer clock frequency) [time](When transfer clock frequency = 400 kHz, one transfer clock cycle = 2.5 μs)

4.1.6 Main processing

To present usage examples of basic functions, a simple main processing that uses the four basic functions (see table 4.1.1) have been created.

The following processing 1. and 2. are performed in the main processing.

Note that it is assumed that the on-chip EEPROM of the H8/3694N is used as a slave device.

1. Test data (the lower 8 bits of an EEPROM address) is written to EEPROM by using "1-byte write" function (byte write); the data that has been written is then read by "1-byte read" function (byte read), and the write/read operation is checked by verification. This processing is repeated for all addresses in a 512-byte area in the EEPROM. The result of the processing is output to port 1 (P12, P11, and P10).

2. To all addresses in a 512-byte area of EEPROM, test data (the lower 8 bits of the EEPROM address + 1) is written using "1-page write" function; the block of data that has been written is then read by using "sequential read" function, and the write/read operations are checked by verification. The result of the processing is output to port 1 (P12, P11, and P10).

For details on main processing, see the module configuration diagram, flowchart, and program.

4.1.7 Language used in this application

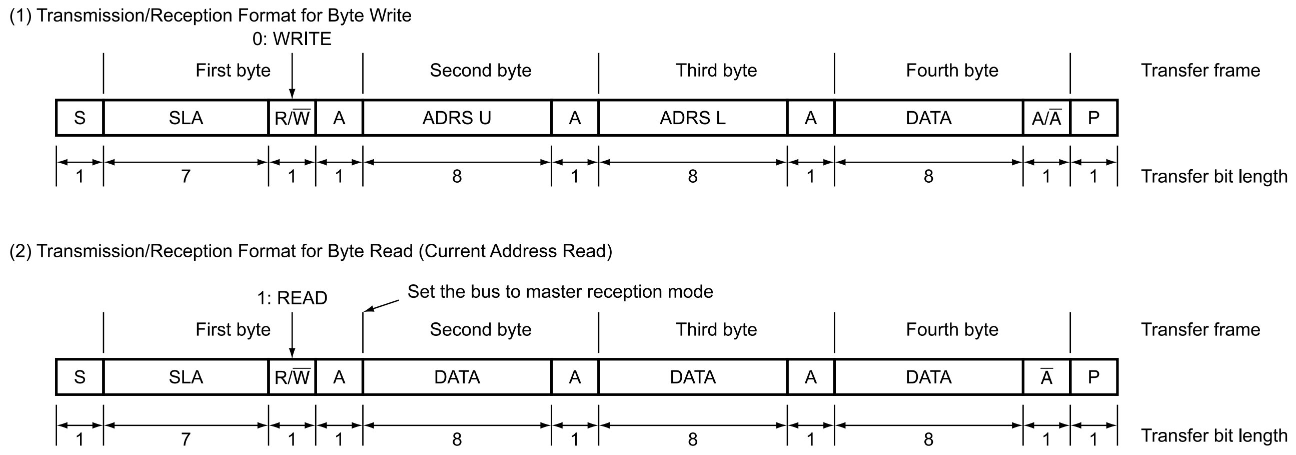
In this application, C language is used for programming.

The program-developing processes from the source program creation to the object code generation is carried out by using the integrated development environment HEW2 (product of Renesas Technology Corp.).

4.1.8 Representation of I^{2}C bus transmission/reception formats

In this application note, the I^{2}C bus transmission/reception formats are described using the representation shown in figure 4.1.1.

Applicable EEPROM types are those for which the memory address is specified in two bytes.



[Legends]

S:	Start condition
SLA:	Slave address (7 bits) of the EEPROM
$R/\overline{W}$:	Transmission/reception (1: read, 0: write)
A:	Acknowledgement (SDA LOW)
$\overline{A}$:	No acknowledgement (SDA HI)
ADRS U:	Upper 8 bits of an EEPROM memory address
ADRS L:	Lower 8 bits of an EEPROM memory address
DATA:	Data (for transmission/reception)
P:	Stop condition

*: For details on the slave address, see table 4.1.2

Figure 4.1.1 Representation of I^{2}C Bus Transmission/Reception Format

4.1.9 Functions of control flags and registers used in this application

Table 4.1.4 summarizes the functions of the control flags and registers used in this application.

Table 4.1.4 Functions of Control Flags and Registers

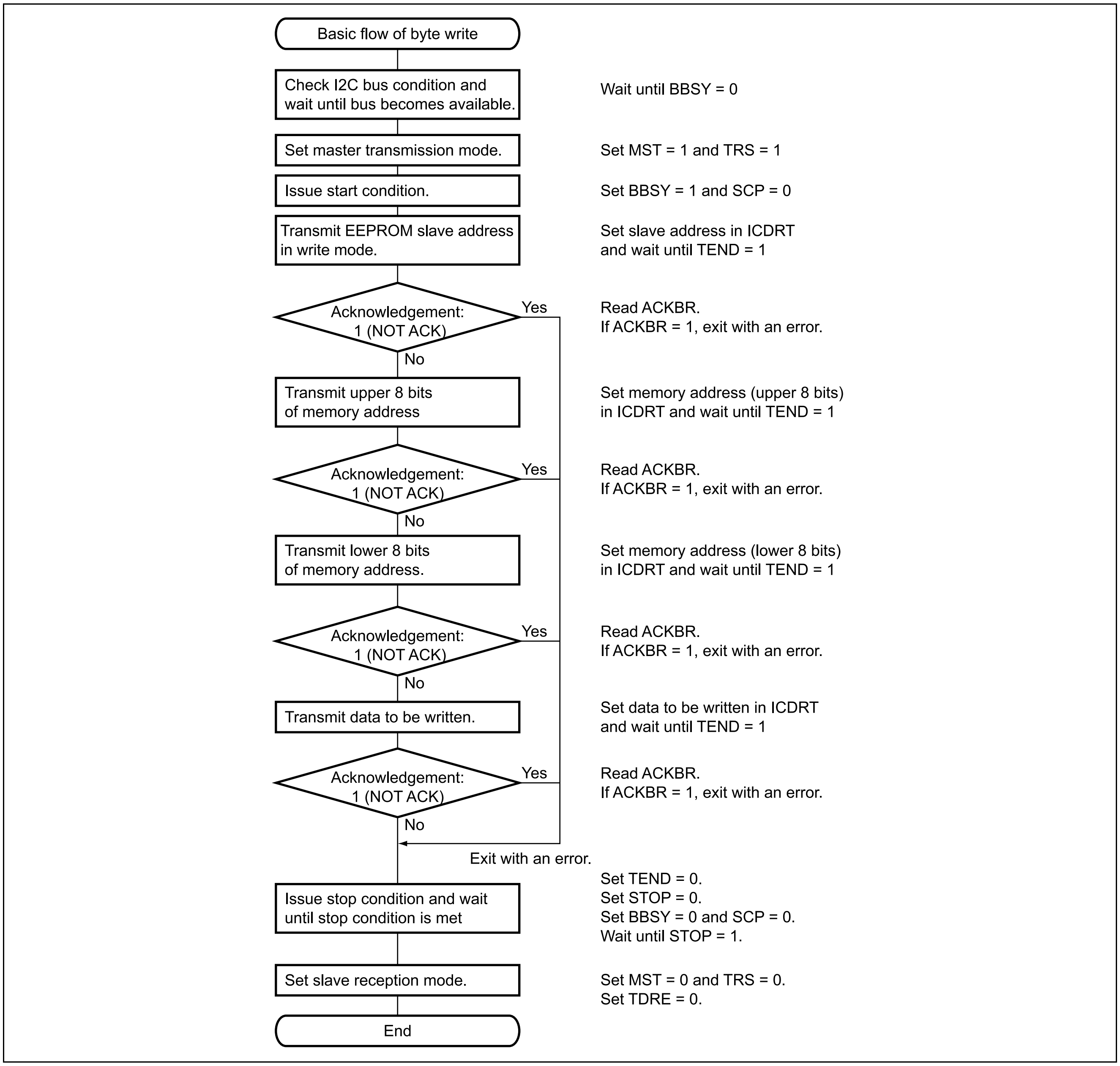
No.	Control Flag/Register Name	Description
1	ICDRR	Receive data register When read, the next reception is performed (the clock output is started).
2	RDRF	Receive data register full This flag is set when data is placed in ICDRR after completion of reception. If the receive-data-full interrupt is enabled (RIE = 1), an RXI interrupt is generated.
4	RCVD	Receive disable 0: Enables subsequent reception operation. (When ICDDR is read, clock for subsequent reception is output.) 1: Disables subsequent reception operation. (When ICDDR is read, clock for subsequent reception is not output.) RCVD should be set prior to reading of ICDRR.
5	ACKBT	Transmission acknowledgement Sets the acknowledgment to be transmitted in reception mode. 0: ACK 1: NOT ACK ACKBT should be set prior to reading of ICDRR.
6	ICDRT	Transmit data register Data transmission is started when data is written to this register and the data is transferred to the shift register (ICDRS). Sequential transmission is possible by writing the subsequent data to ICDRT during data
7	TEND	Transmit end This flag is set on the rising edge of the ninth clock of SCL when TDRE = 1. When TDRE = 1, transmission has been completed and acknowledgement is being received. The TEND flag can be used if data should be transferred while confirming the acknowledgement. (TEND is used in this application.) After transmission of a slave address or final data, completion of transmission can be securely detected by using the TEND flag. If the transmit-end interrupt is enabled (TEIE = 1), a TEI interrupt is generated.
8	TDRE	Transmit data empty This flag is set when data is transferred from ICDRT to ICDRS. The TDRE flag can be used when data transfer is done without confirming the acknowledgement. (Detection of an acknowledgement error in this case can be performed by means of NACK receive interrupts (NACKI). If the transmit data empty interrupt is enabled (TIE = 1), a TXI interrupt is generated.
9	ACKBR	Reception acknowledgement In transmission mode, an acknowledgement from a receiving device is stored to this bit. 0: ACK, 1: NOT ACK

4.1.10 Basic Flow of the EEPROM Write/Read Processing

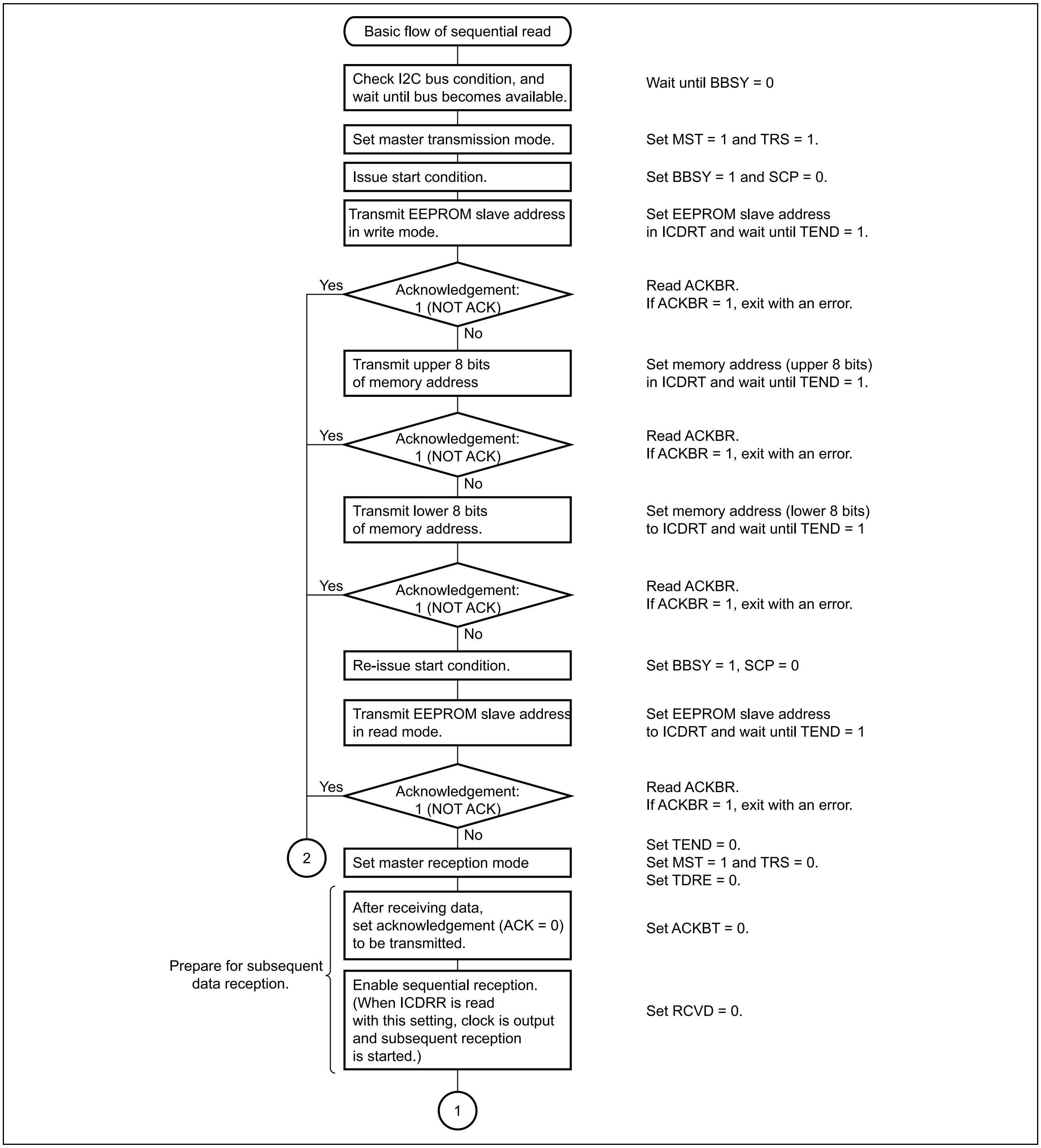
The basic flow of EEPROM write/read processing via the I^2C bus interface 2 (IIC2) are described below with reference to the examples: 1. byte write and 2. sequential read.

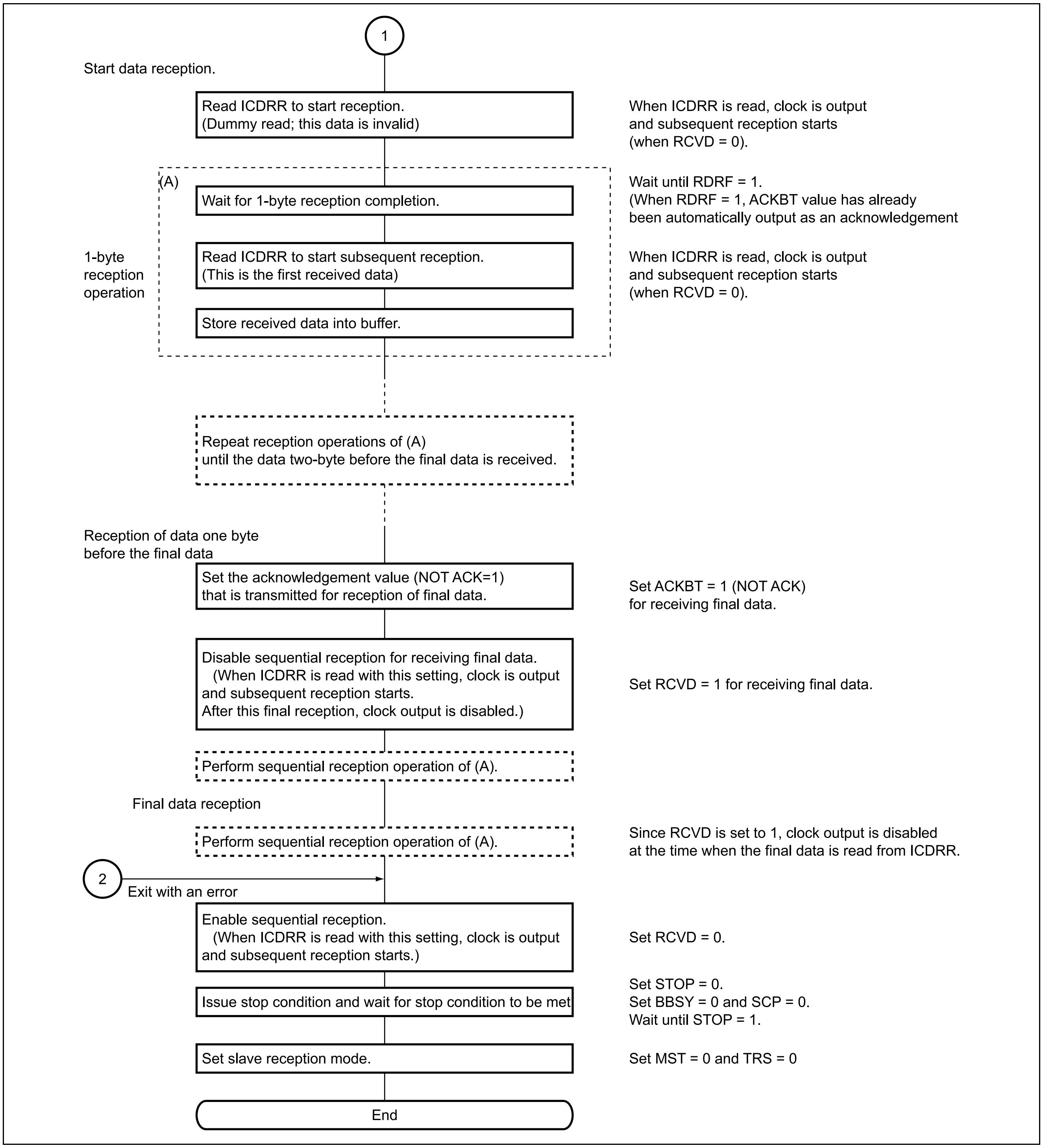
Since these flows are only for describing the basic flow of processing, they might not be consistent with the sample application program at some minor points. Thus, please use this flow in addition to the Hardware Manual as a reference for dealing with the flags and registers.

1. Standard write processing flow for EEPROM (Byte write)


Basic flow of byte write
Check I2C bus condition and wait until bus becomes available.
Wait until BBSY = 0
Set master transmission mode.
Set MST = 1 and TRS = 1
Issue start condition.
Set BBSY = 1 and SCP = 0
Transmit EEPROM slave address in write mode.
Set slave address in ICDRT and wait until TEND = 1
Acknowledgement: 1 (NOT ACK)
Yes
No
Read ACKBR.
If ACKBR = 1, exit with an error.
Transmit upper 8 bits of memory address
Set memory address (upper 8 bits) in ICDRT and wait until TEND = 1
Acknowledgement: 1 (NOT ACK)
Yes
No
Read ACKBR.
If ACKBR = 1, exit with an error.
Transmit lower 8 bits of memory address.
Set memory address (lower 8 bits) in ICDRT and wait until TEND = 1
Acknowledgement: 1 (NOT ACK)
Yes
No
Read ACKBR.
If ACKBR = 1, exit with an error.
Transmit data to be written.
Set data to be written in ICDRT and wait until TEND = 1
Acknowledgement: 1 (NOT ACK)
Yes
No
Read ACKBR.
If ACKBR = 1, exit with an error.
Exit with an error.
Issue stop condition and wait until stop condition is met
Set TEND = 0.
Set STOP = 0.
Set BBSY = 0 and SCP = 0.
Wait until STOP = 1.
Set slave reception mode.
Set MST = 0 and TRS = 0.
Set TDRE = 0.
End

2. Standard read processing flow for EEPROM (Sequential read)


Basic flow of sequential read
Check I2C bus condition, and wait until bus becomes available.
Wait until BBSY = 0
Set master transmission mode.
Set MST = 1 and TRS = 1.
Issue start condition.
Set BBSY = 1 and SCP = 0.
Transmit EEPROM slave address in write mode.
Set EEPROM slave address in ICDRT and wait until TEND = 1.
Yes
Acknowledgement: 1 (NOT ACK)
No
Read ACKBR.
If ACKBR = 1, exit with an error.
Transmit upper 8 bits of memory address
Set memory address (upper 8 bits) in ICDRT and wait until TEND = 1.
Yes
Acknowledgement: 1 (NOT ACK)
No
Read ACKBR.
If ACKBR = 1, exit with an error.
Transmit lower 8 bits of memory address.
Set memory address (lower 8 bits) to ICDRT and wait until TEND = 1
Yes
Acknowledgement: 1 (NOT ACK)
No
Read ACKBR.
If ACKBR = 1, exit with an error.
Re-issue start condition.
Set BBSY = 1, SCP = 0
Transmit EEPROM slave address in read mode.
Set EEPROM slave address to ICDRT and wait until TEND = 1
Yes
Acknowledgement: 1 (NOT ACK)
No
Read ACKBR.
If ACKBR = 1, exit with an error.
2
Set master reception mode
Set TEND = 0.
Set MST = 1 and TRS = 0.
Set TDRE = 0.
Prepare for subsequent data reception.
After receiving data, set acknowledgement (ACK = 0) to be transmitted.
Set ACKBT = 0.
Enable sequential reception. (When ICDRR is read with this setting, clock is output and subsequent reception is started.)
Set RCVD = 0.
1


1
Start data reception.
Read ICDRR to start reception.
(Dummy read; this data is invalid)
When ICDRR is read, clock is output
and subsequent reception starts
(when RCVD = 0).
(A)
Wait for 1-byte reception completion.
Wait until RDRF = 1.
(When RDRF = 1, ACKBT value has already
been automatically output as an acknowledgement
1-byte
reception
operation
Read ICDRR to start subsequent reception.
(This is the first received data)
When ICDRR is read, clock is output
and subsequent reception starts
(when RCVD = 0).
Store received data into buffer.
Repeat reception operations of (A)
until the data two-byte before the final data is received.
Reception of data one byte
before the final data
Set the acknowledgement value (NOT ACK=1)
that is transmitted for reception of final data.
Set ACKBT = 1 (NOT ACK)
for receiving final data.
Disable sequential reception for receiving final data.
(When ICDRR is read with this setting, clock is output
and subsequent reception starts.
After this final reception, clock output is disabled.)
Set RCVD = 1 for receiving final data.
Perform sequential reception operation of (A).
Final data reception
Perform sequential reception operation of (A).
Since RCVD is set to 1, clock output is disabled
at the time when the final data is read from ICDRR.
2
Exit with an error
Enable sequential reception.
(When ICDRR is read with this setting, clock is output
and subsequent reception starts.)
Set RCVD = 0.
Issue stop condition and wait for stop condition to be met
Set STOP = 0.
Set BBSY = 0 and SCP = 0.
Wait until STOP = 1.
Set slave reception mode.
Set MST = 0 and TRS = 0
End

4.2 Function list

The functions created for this application and their functionalities are shown in tables 4.2.1 to 4.2.5.

Table 4.2.1 IIC2 Main Function

No.	Function Name	Description
0	Main(void)	Functions No. 1 to No. 4 are called to show their usage examples.
1	Master_byte_write(Device_id, Addr, Data)	1-byte write
2	Master_read_byte_random(Device_id, Addr)	1-byte read
3	Master_page_write(Device_id, Addr, Length, *buff_ptr)	1-page write (N-byte write)
4	Master_read_sequential(Device_id, Addr, Length, *buff_ptr)	Sequential read (N-byte read)

Table 4.2.2 IIC2 Sub Function

No.	Function Name	Description
5	Master_address_set(Device_id, Addr)	Transmits slave address and memory address.
6	Check_bus_condition(void)	Checks bus condition.
7	Send_start_condition(void)	Issues start condition.
8	Send_stop_condition(void)	Issues stop condition.
9	Send_byte_data(Byte_data)	Transmits 1-byte data.
10	Receive_byte_data(void)	Receives 1-byte data.
11	Receive_byte_data_fin(void)	Receives final 1-byte data
12	Receive_byte_data_many(Length, *buff_ptr)	Receives byte data sequentially (for read byte length).
13	Set_slave_read_mode(Device_id)	Transmits slave address (read mode).
14	Set_slave_write_mode(Device_id)	Transmits slave address (write mode).
15	Set_receive_mode(Ackbt_flag, Rcvd_flag)	Sets ACKBT and RCVD flags.

Table 4.2.3 IIC2 Initialization Function

No.	Function Name	Description
16	Init_iic2 (void)	Initializes IIC2 I/F.
17	Set_iic_bus_mode(MLS, WAIT, BC210)	Sets master device bus mode.
18	Set_iic_mode(MODE)	Sets master device transmission/reception mode.
19	Set_iic_rate(RATE)	Sets master device transfer rate.
20	Set_iic2_if_enable(ICE)	Sets IIC2 I/F module operation status.

Table 4.2.4 EEPROM Initialization Function (Controls IIC Bus through Port Processing and Initializes EEPROM)

No.	Function Name	Description
21	Init_eeprom(void)	Initializes EEPROM bus condition.
22	I2c_start (void)	Issues start condition to IIC device. (port processing)
23	I2c_stop (void)	Issues stop condition to IIC device. (port processing)
24	I2c_set (Scl, Sda)	Outputs SCL and SDA to IIC device. (port processing)
25	I2c_bytesend (Tx_data)	Transmits byte data to IIC device. (port processing)
26	I2c_bitsend (Tx_data, Ckbit)	Transmits bit data to IIC device. (port processing)
27	I2c_send (Bit_data)	Transmits bit data synchronized with a clock from IIC port. (port processing)
28	I2c_ackck (void)	Obtains acknowledgements from IIC device. (port processing)
29	I2c_sda_in (void)	Sets SDA (P56) of IIC port as input and SCL (P57) as output.
30	I2c_sda_out (Data)	Sets SDA (P56) and SCL (P57) of IIC port as output and outputs data.
31	Wait_timer (Wait_cnt)	Wait for the specified time (countdown until the wait count reaches 0).

Note: The EEPROM initialization functions are used to forcibly bring the SDA bus of EEPROM into input mode when the SDA bus of EEPROM still remains in output mode (this might be caused when, for example, communication is halted during data reception from EEPROM) and reception processing from the master device cannot be performed. These functions are only for reference and do not have direct relation to the main function.

Table 4.2.5 Initialization (Startup) Function

No	Function Name	Description
32	INIT(void)	Initialization (startup) routine (created in INIT.SRC file)

4.3 Global variables and constants

Tables 4.3.1 and 4.3.2 show the global variable and constants (defined using #define directives) used in this application program.

Table 4.3.1 Global Variable

No.	Variable Name	Type	Size	Application
1	eeprom_buf	unsigned char	513	EEPROM read/write data storage buffer

Table 4.3.2 Constants

1. Device code definition

No.	Variable Name	Type	Size	Application
1	DEVICE_CODE	unsigned char	0xa0	EEPROM device code: 1010 (D3, D2, D1, D0)
2	SLAVE_ADRS	unsigned char	0x00	Slave address code: 000 (A2, A1, A0)
3	RW_CODE_W	unsigned char	0x00	R/W code: 0 (data write)
4	RW_CODE_R	unsigned char	0x01	R/W code: 1 (data read)
5	DEVICE_ADDRESS_WORD_W	unsigned char	See right. (DEVICE_CODE | SLAVE_ADRS |RW_CODE_W)	Slave address (for writing)
6	DEVICE_ADDRESS_WORD_R	unsigned char	See right. (DEVICE_CODE | SLAVE_ADRS | RW_CODE_R)	Slave address (for reading)

Note: For slave address, see table 4.1.2.

2. Definition of timeout limit counts (number of times for terminating respective flag check processings: assumed value)

No.	Variable Name	Type	Size	Application
1	TIMEOUT_LIMIT_BBSY	unsigned char	1000	Bus busy check timeout limit
2	TIMEOUT_LIMIT_ACK	unsigned char	1000	ACK check timeout limit
3	TIMEOUT_LIMIT_TEND	unsigned char	1000	TEND check timeout limit
4	TIMEOUT_LIMIT_RDRF	unsigned char	1000	RDRF check timeout limit
5	TIMEOUT_LIMIT_STOP	unsigned char	1000	STOP check timeout limit

3. Definition of error codes

No.	Variable Name	Type	Size	Application
1	IMEOUT_ERR_BUS_BUSY	unsigned char	200	Bus busy timeout error
2	TIMEOUT_ERR_ACK	unsigned char	201	Acknowledgement polling timeout error
3	TIMEOUT_ERR_TEND	unsigned char	202	Transmit end timeout error
4	TIMEOUT_ERR_RDRF	unsigned char	203	Receive end timeout error
5	TIMEOUT_ERR_STOP	unsigned char	204	Stop condition issue timeout error
6	ERR_ACK	unsigned char	205	Acknowledgement error

4. Other definition

No.	Variable Name	Type	Size	Application
1	TM	unsigned char	100 (assumed value)	Wait count for the wait timer (Used for clock output of EEPROM initialization function)

4.4 Section definitions

Table 4.4.1 shows section (memory allocation address) definitions of this application program.

Table 4.4.1 Section Definitions

Allocated Address	Section name	Allocated as
H'0000	C	Vector area
H'1000	P	Program area
H'FB80	B	RAM area

4.5 Description of Functions

Description of the major functions are provided in sections 4.5.1 through 4.5.7.

The major functions described are listed in table 4.5.1. (Section numbers 4.5.1 through 4.5.7 correspond to the major function numbers in this table.)

Table 4.5.1 Major Functions Described in This Section

No.	Function Name	Description
1	Master_byte_write(Device_id, Addr, Data)	1-byte write
2	Master_read_byte_random(Device_id, Addr)	1-byte read
3	Master_page_write(Device_id, Addr, Length, *buff_ptr)	1-page write
4	Master_read_sequential(Device_id, Addr, Length, *buff_ptr)	Sequential read
5	Master_address_set (Device_id, Addr)	Transmission of slave address and memory address
6	Receive_byte_data_many (Length, *buff_ptr)	Sequential data reception
7	Init_eeprom(void)	Initialization of EEPROM bus condition

4.5.1 1-byte write

1.Function Name
Master_byte_write(Device_id, Addr, Data)
1-byte write

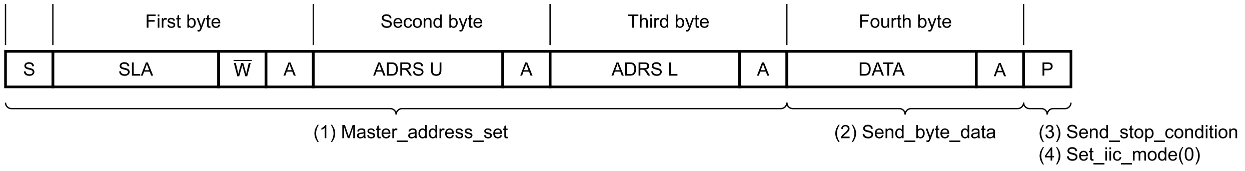
2. Argument
A. Entry
(1) Device_id (unsigned char) ; Slave address code: 0 to 7 (A2, A1, A0)
(2) Addr (unsigned short) ; Memory address (2 bytes)
(3) Data (unsigned char) ; Write data (1 byte)
B. Return
(1) Err_code (unsigned char); Error code
0: Normal termination
2: Second byte acknowledgement error (upper 8 bits of memory address)
3: Third byte acknowledgement error (lower 8 bits of memory address)
4: Fourth byte acknowledgement error (data)
TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error
TIMEOUT_ERR_ACK: Acknowledgement polling timeout error
TIMEOUT_ERR_STOP: Stop condition detection timeout error

Note: The transmit end timeout error is returned as 2 to 4 of Err_code.

3. Processing
Transmits to the slave device the slave address (1 byte), memory address (2 bytes), and write data (1 byte) according to the transmission/reception format, and writes the write data to the memory address.

4. Functions used
(1) Master_address_set(Device_id, Addr); Transmits slave address and memory address.
(2) Send_byte_data(Data); Transmits data.
(3) Send_stop_condition(); Issues stop condition.
(4) Set_iic_mode(0); Sets slave reception mode.

Transmission format



4.5.2 1-byte read

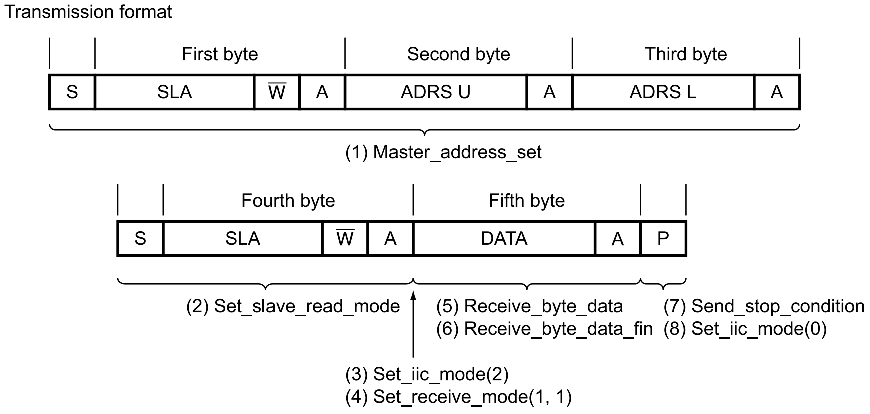
1. Function Name
   Master_read_byte_random(Device_id, Addr)
   1-byte read
2. Arguments
   A. Entry
      (1) Device_id (unsigned char) ; slave address code: 0 to 7 (A2,A1,A0)
      (2) Addr (unsigned short) ; memory address (2 bytes)
   B Return
      (1) Err_code (unsigned char); error code (upper 8 bits)
         0: Normal termination
         2: Second byte acknowledgement error (upper 8 bits of memory address)
         3: Third byte acknowledgement error (lower 8 bits of memory address)
         ERR_ACK: Slave address (read) acknowledgement error
         TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error
         TIMEOUT_ERR_ACK: Acknowledgement polling timeout error
         TIMEOUT_ERR_RDRF: Receive end timeout error
         TIMEOUT_ERR_STOP: Stop condition detection timeout error

Note: The transmit end timeout error is returned as 2 to 3 of Err_code.

      (2) Data (unsigned char) ; received data (lower 8 bits)

3. Processing
   Transmits to the slave device the slave address (1 byte) and memory address (2 bytes) according to the transmission/reception format, retransmits the slave address (1 byte) in read mode, and reads the data (1 byte) from the memory address in master reception mode.

4. Functions used
   (1) Master_address_set(Device_id, Addr); Transmits slave address and memory address.
   (2) Set_slave_read_mode(Device_id); Retransmits slave address in read mode.
   (3) Set_iic_mode(2); Sets master reception mode.
   (4) Set_receive_mode(1, 1); Sets reception mode (ACKBT = 1, RCVD = 1).
   (5) Receive_byte_data(); Receives data
   (6) Receive_byte_data_fin(); Gets the finally received data.
   (7) Send_stop_condition(); Issues stop condition.
   (8) Set_iic_mode(0); Sets slave reception mode.



(4) Since this is the 1-byte reception for the final byte,
set ACKBT = 1 (Nack is output in the subsequent reception)
and RCVD = 1 (the subsequent clock output is disabled.

(5) Since this is a dummy read for starting reception operation (receive clock output),
the received data should be regarded as invalid.

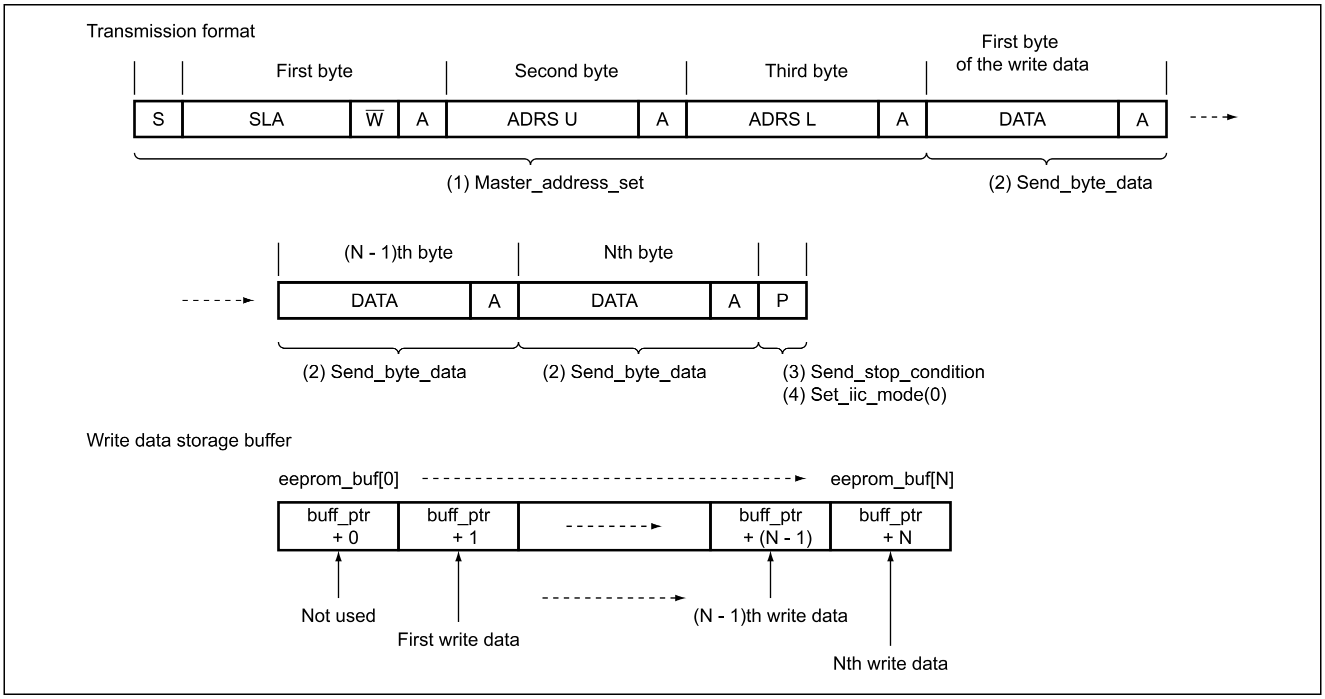
4.5.3 1-page write (N-byte write)

1. Function Name
   Master_page_write(Device_id, Addr, Length, *buff_ptr)
   1-page write (N-byte write)
2. Arguments
   A. Entry
      (1) Device_id (unsigned char); Slave address code: 0 to 7 (A2, A1, A0)
      (2) Addr (unsigned short); Write start memory address (2 bytes)
      (3) Length (unsigned char); Write byte length N (specify within 1 page)
      (4) *buff_ptr (unsigned char); First address of the write data storage buffer
   B. Return
      (1) Err_code (unsigned char); error code
         0: Normal termination
         2: Second byte acknowledgement error (upper 8 bits of memory address)
         3: Third byte acknowledgement error (lower 8 bits of memory address)
         4 to 131 (number of transmitted bytes + 3): Transmission acknowledgement error for the fourth and following bytes (data)
         TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error
         TIMEOUT_ERR_ACK: Acknowledgement polling timeout error
         TIMEOUT_ERR_STOP: Stop condition detection timeout error

Note: The transmit end timeout error is returned as 2 to 131 of Err_code.

3. Processing
   Transmits to the slave device the slave address (1 byte) and memory address (2 bytes) according to the transmission/reception format; after that, sequentially transmits the write data of the size specified by Length (N bytes), and writes the data through page writing.
   — Write start memory address (Addr) and write byte length (Length) should be set so that the write addresses fall within a page during writing process.
     When the address exceeds the range of a page, the write address pointer for EEPROM is rollovers, and overwriting occurs from the first address of the page.
   — Write data of the size specified by Length (N bytes) should be written in advance to the write data storage buffer, from its first address + 1 to the first address + N. (The location at the first address of the buffer is not used.)

4. Functions used
    (1) Master_address_set(Device_id, Addr); Transmits slave address and memory address.
    (2) Send_byte_data(Data); Transmits data.
    (3) Send_stop_condition(); Issues stop condition.
    (4) Set_iic_mode(0); Sets slave reception mode.



4.5.4 Sequential read (N-byte read)

1. Function Name

   Master_read_sequential(Device_id, Addr, Length, *buff_ptr)
   Sequential read (N-byte read)

2. Arguments

   A. Entry

      (1) Device_id (unsigned char) ; Slave address code: 0 to 7 (A2, A1, A0)
      (2) Addr (unsigned short) ; Read start memory address (2 bytes)
      (3) Length (unsigned short) ; Read byte length N (more than 2 bytes)
      (4) *buff_ptr (unsigned char) ; First address of the read data storage buffer

   B. Return

      (a) Err_code (unsigned char); error code
         0: Normal termination
         2: Second byte acknowledgement error (upper 8 bits of memory address)
         3: Third byte acknowledgement error (lower 8 bits of memory address)
         ERR_ACK: Slave address (read) acknowledgement error
         TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error
         TIMEOUT_ERR_ACK: Acknowledgement polling timeout error
         TIMEOUT_ERR_RDRF: Data receive end timeout error
         TIMEOUT_ERR_STOP: Stop condition detection timeout error

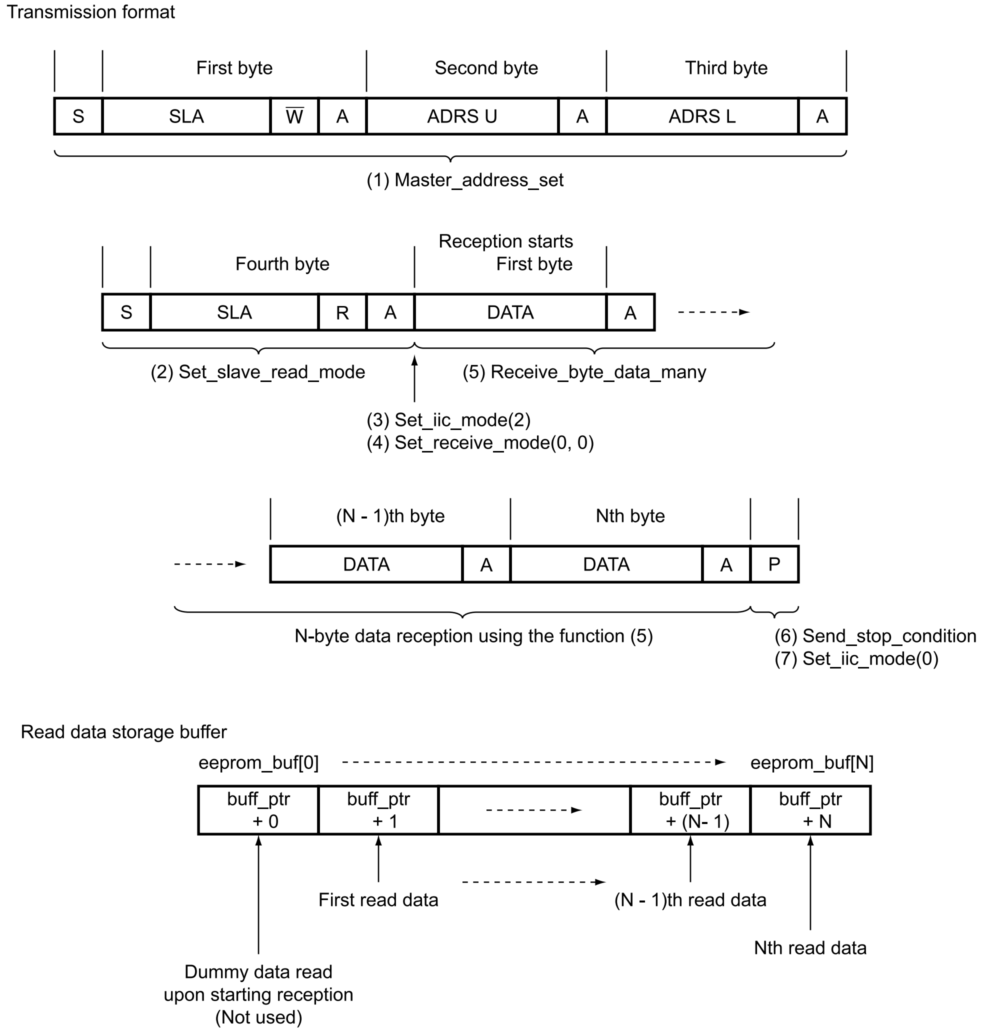
3. Processing

   Transmits to the slave device the slave address (1 byte) and memory address (2 bytes) according to the transmission/reception format; after that, retransmits the slave address (1 byte) in read mode, and sequentially reads data (N bytes) from the memory address in master reception mode.

   — The read data is sequentially stored in the read data storage buffer from its first address.
     In the location at the first address of the buffer, dummy data which has been read upon starting reception is stored. Therefore, valid received data is stored to the addresses from (first address of the buffer + 1) to (first address of the buffer + N).

4. Functions used

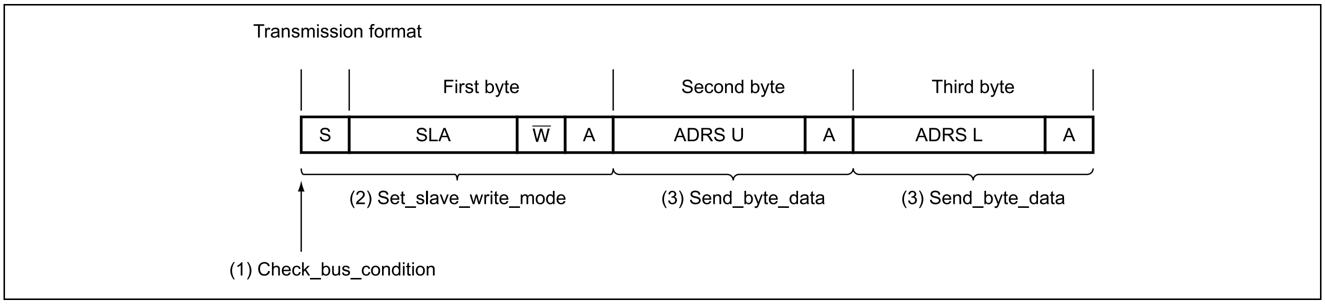
(1) Master_address_set(Device_id, Addr; Transmits slave address and memory address.
(2) Set_slave_read_mode(Device_id); Retransmits slave address in read mode.
(3) Set_iic_mode(2); Sets master reception mode.
(4) Set_receive_mode(0, 0); Sets reception mode (ACKBT = 0, RCVD = 0).
(5) Receive_byte_data_many(Length, buff_ptr); Receives data (N bytes).
(6) Send_stop_condition(); Issues stop condition.
(7) Set_iic_mode(0); Sets slave reception mode.



(4) To carry out sequential reception, set ACKBT = 0 (Ack is output in the subsequent reception) and RCVD = 0 (clock output for the subsequent reception is enabled).

4.5.5 Transmits slave address and memory address

1. Function Name
   Master_address_set (Device_id, Addr)
   Transmits slave address and memory address
2. Arguments
   A. Entry
      (1) Device_id (unsigned char); Slave address code: 0 to 7 (A2, A1, A0)
      (2) Addr (unsigned short) ; Memory address (2 bytes)
   B. Return
      (1) Err_code (unsigned char); error code
         0: Normal termination
         2: Second byte acknowledgement error (upper 8 bits of memory address)
         3: Third byte acknowledgement error (lower 8 bits of memory address)
         TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error
         TIMEOUT_ERR_ACK: Acknowledgement polling timeout error
3. Processing
   Transmits to the slave device the slave address (1 byte) and memory address (2 byte) according to the transmission/reception format.
4. Functions used
   (1) Check_bus_condition(); Checks bus condition.
   (2) Set_slave_write_mode(Device_id); Transmits slave address (write mode).
   (3) Send_byte_data(Data); Transmits data.



4.5.6 Sequential data reception (for N bytes)

1. Function Name

Receive_byte_data_many (Length, *buff_ptr)
Sequential data reception (for N bytes)

2. Arguments

A. Entry

(1) Length (unsigned short); Read byte length N (2 bytes or more)

(2) *buff_ptr (unsigned char); First address of read data storage buffer

Since dummy data, which is read upon start of reception, is stored to the first address (buff_ptr), valid data of the size specified by Length is stored to (buff_ptr + 1) and following addresses.

B. Return

(1) Err_code (unsigned char); error code

0: Normal termination

TIMEOUT_ERR_RDRF: Data receive end timeout error

3. Processing

Reads data (N bytes) from the slave device sequentially according to the transmission/reception format.

— The read data is sequentially stored in the read data storage buffer from its first address.
In the location at the first address of the buffer, dummy data which has been read upon starting reception is stored. Therefore, valid received data is stored to the addresses from (first address of the buffer + 1) to (first address of the buffer + N).

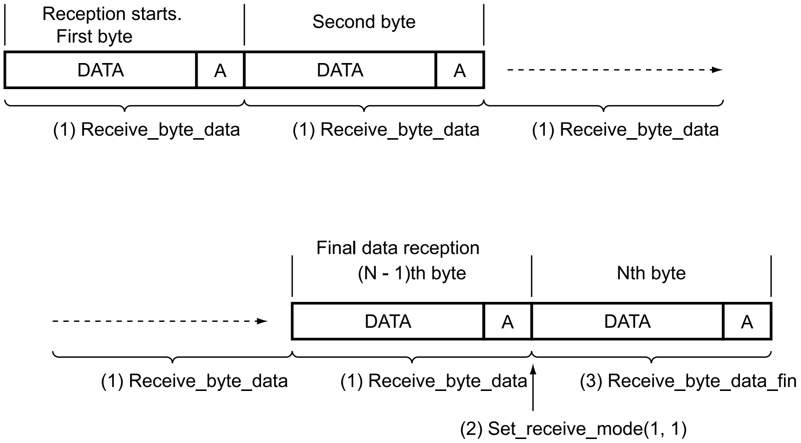
4. Functions used

(1) Receive_byte_data(); Receives 1-byte data.

(2) Set_receive_mode(1, 1); Sets reception mode (ACKBT = 1, RCVD = 1).

(3) Receive_byte_data_fin(); Receives final 1-byte data.

Transmission format



(1) The first data read (first byte) is dummy read for starting reception operation (receive clock output).
This invalid received data is stored in eeprombuf[0].

(2) Since this is the final reception for the N-th byte,
set ACKBT = 1 (Nack is output in the subsequent reception) and RCVD = 1
(the subsequent clock output is disabled).

4.5.7 Initializes EEPROM bus condition

1. Function Name

   Init_eeprom(void)

   Initializes EEPROM bus condition

2. Arguments

   A. Entry

      None; None

   B. Return

      None; None

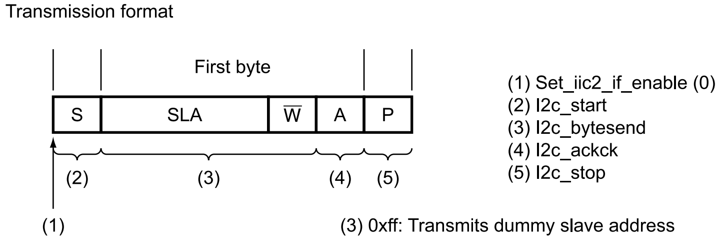
3. Processing

   Forcibly initializes EEPROM bus condition by first issuing start condition according to the transmission/reception format via a port, transmitting dummy slave address (0xff), then issuing stop condition.

   (This initialization processing is used to forcibly set the SDA bus of the EEPROM in input mode when reception processing from the master device cannot be performed because of the SDA bus of the EEPROM still remaining in output mode due to communication disconnection during data reception from EEPROM.)

4. Functions used

   (1) Set_iic2_if_enable (0); Sets IIC2 I/F module to a halt state.; (SCL/SDA pin to port function)
   (2) I2c_start () ; Issues start condition to the IIC device (port processing).
   (3) I2c_bytesend (Tx_data); Transmits byte data to the IIC device (port processing).
   (4) I2c_ackck (); Obtains acknowledgement from the IIC device (port processing).
   (5) I2c_stop (); Issues stop condition to the IIC device (port processing).
   (6) Wait_timer (); Waits for the specified time (countdown until the wait count reaches 0).



4.6 Program Files

Table 4.6.1 shows the program files involved in this application.

Table 4.6.1 Program Files Involved in This Application

No.	Function Name	Description
1	H8_3694_IIC2.H	Register map definitions
2	INIT.SRC	Initialization routine (start up)
3	eeprom.c	Main routine
4	iic2_eeprom.c	Processing functions

4.7 Module configuration diagram

Module configuration of this application program is shown in section 4.7 1. through 7. .

In the following module configuration diagrams, a function is represented as a box (figure 4.7.1) and the hierarchical structure of functions is depicted.

The function numbers correspond to the item numbers in the function lists (table 4.2.1 through 4.2.4). A function number containing a dot indicates that there is any functions called in that function.

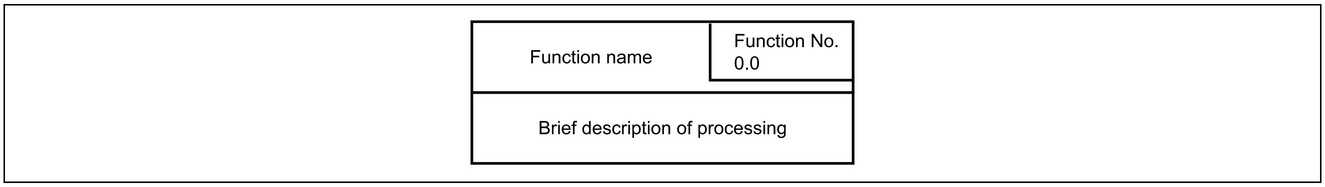
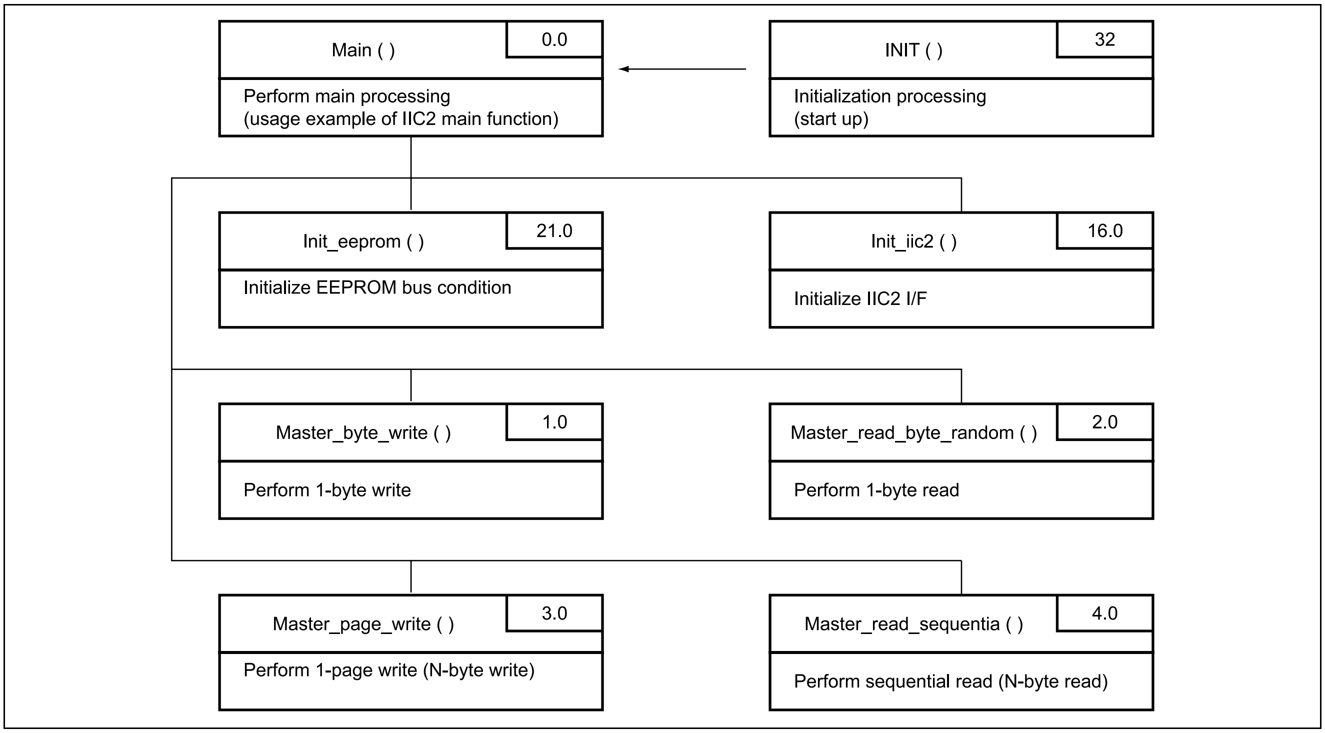
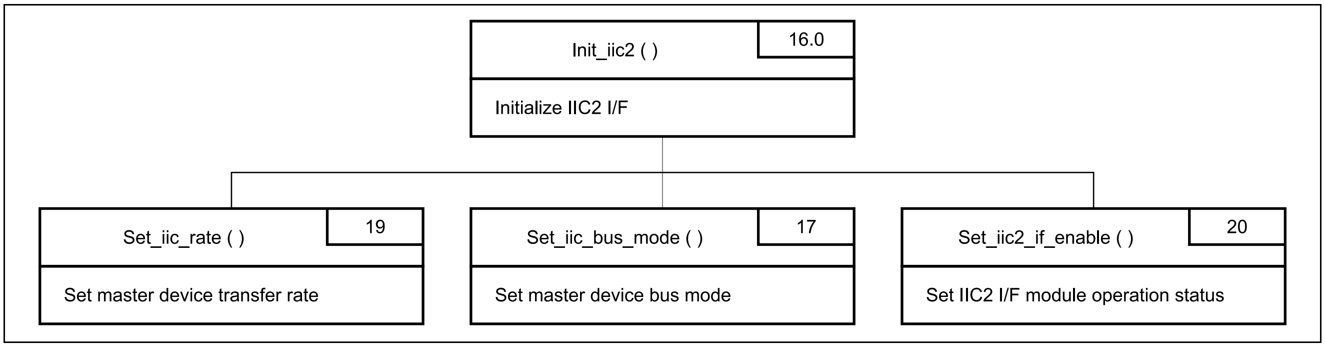


Figure 4.7.1 Representation of a Function

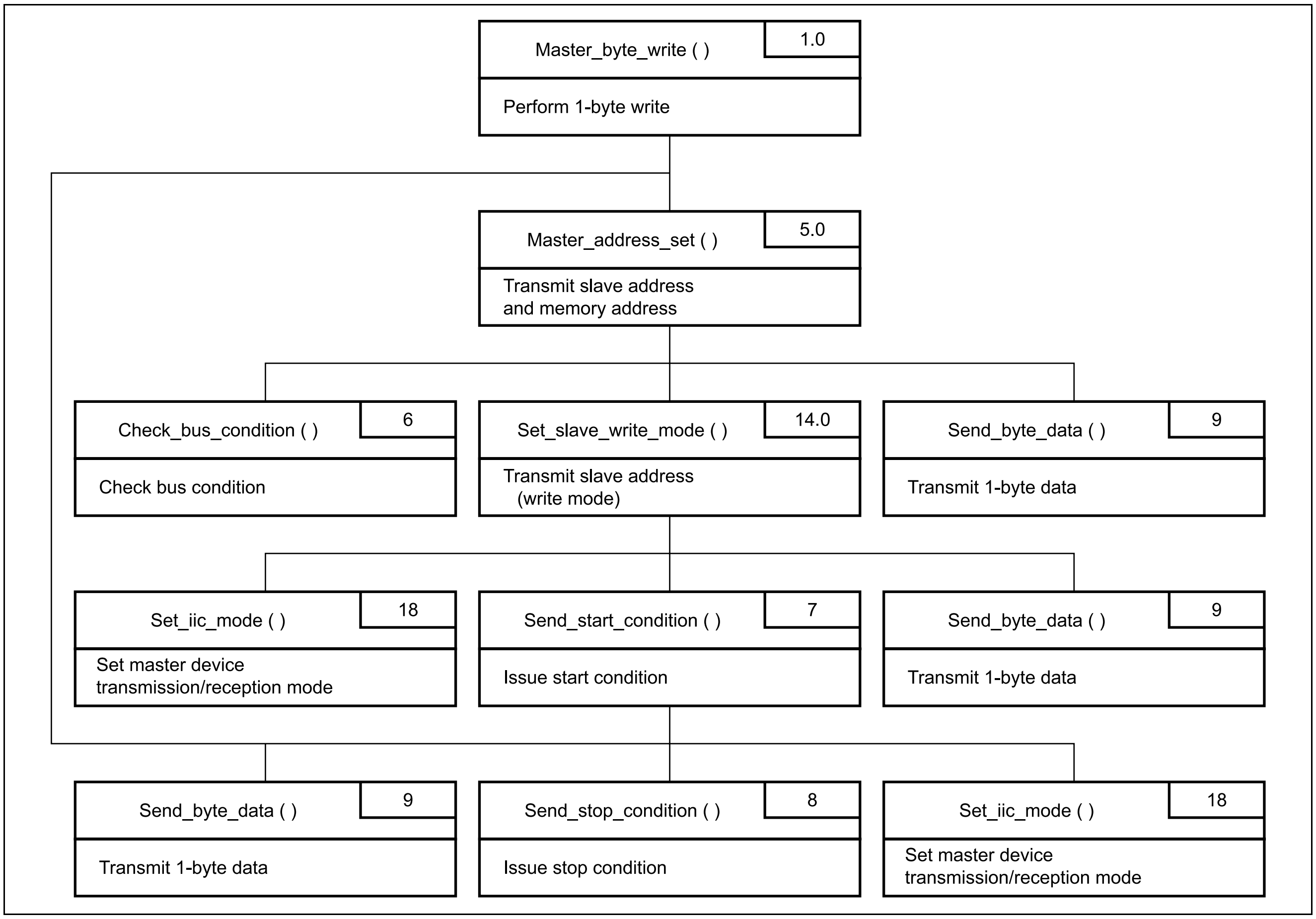
1. Main ()


Main ( )
0.0
Perform main processing
(usage example of IIC2 main function)
INIT ( )
32
Initialization processing
(start up)
Init_eeprom ( )
21.0
Initialize EEPROM bus condition
Init_iic2 ( )
16.0
Initialize IIC2 I/F
Master_byte_write ( )
1.0
Perform 1-byte write
Master_read_byte_random ( )
2.0
Perform 1-byte read
Master_page_write ( )
3.0
Perform 1-page write (N-byte write)
Master_read_sequentia ( )
4.0
Perform sequential read (N-byte read)

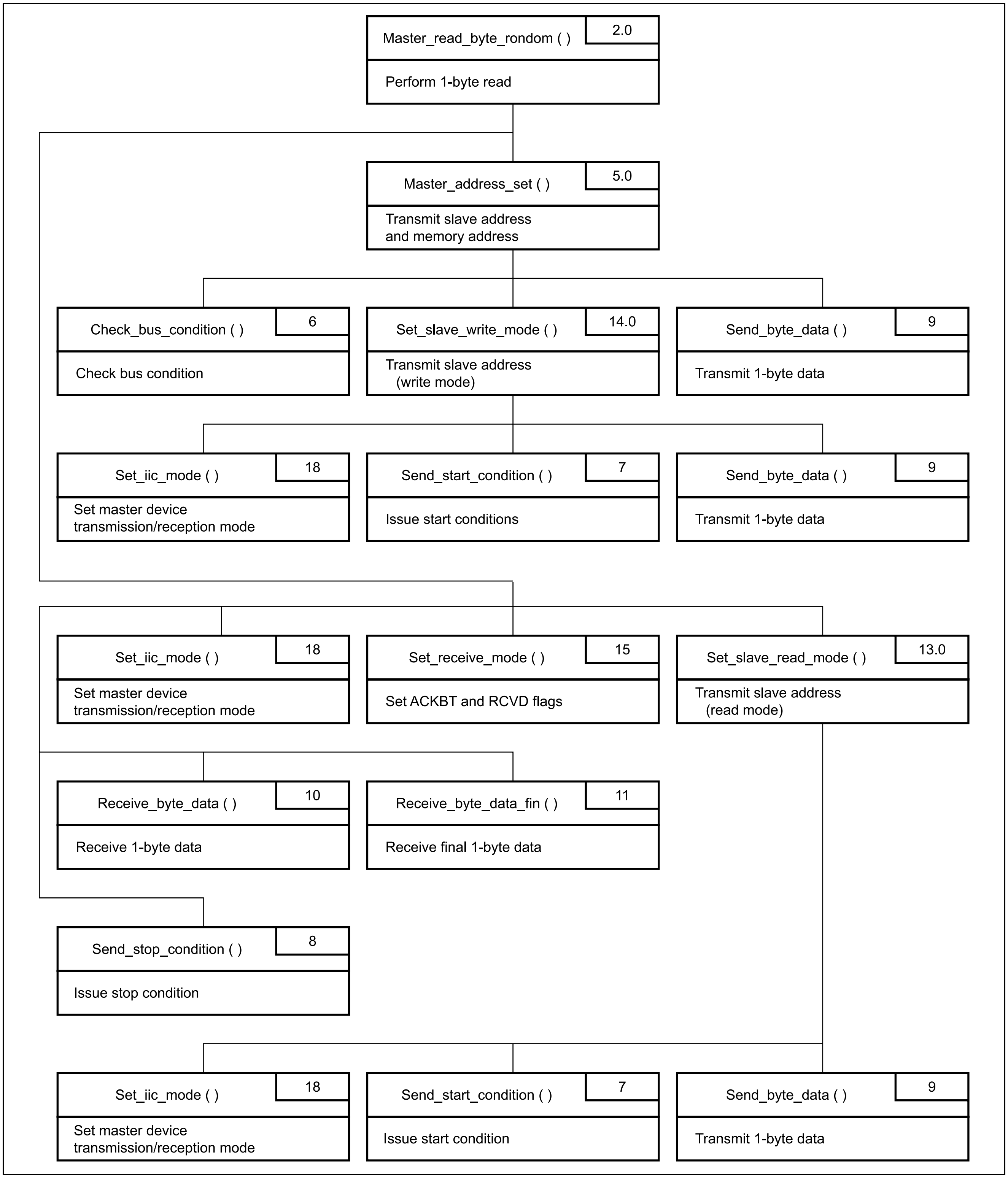
2. Init_iic2 ()



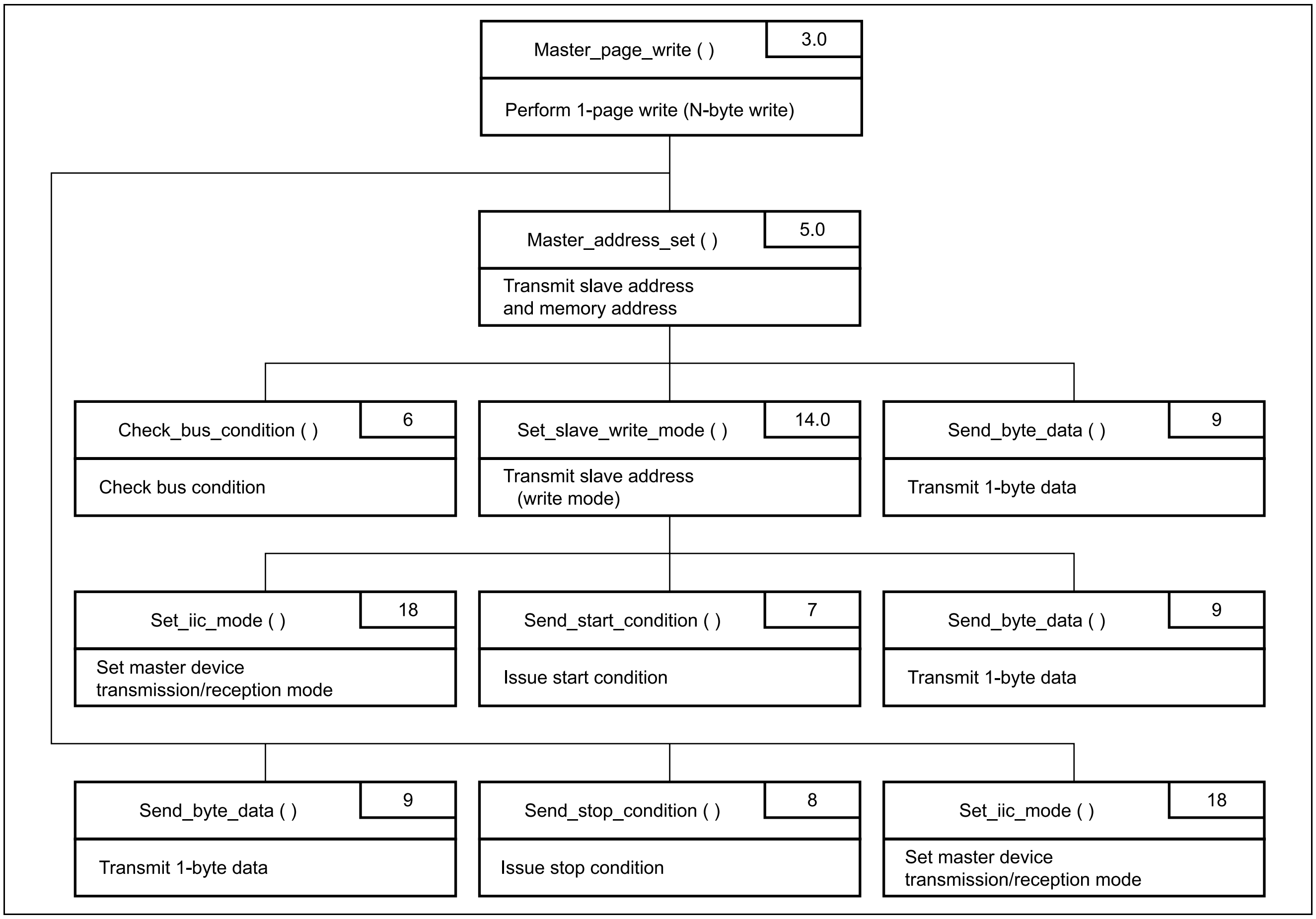
3. Master_byte_write ()



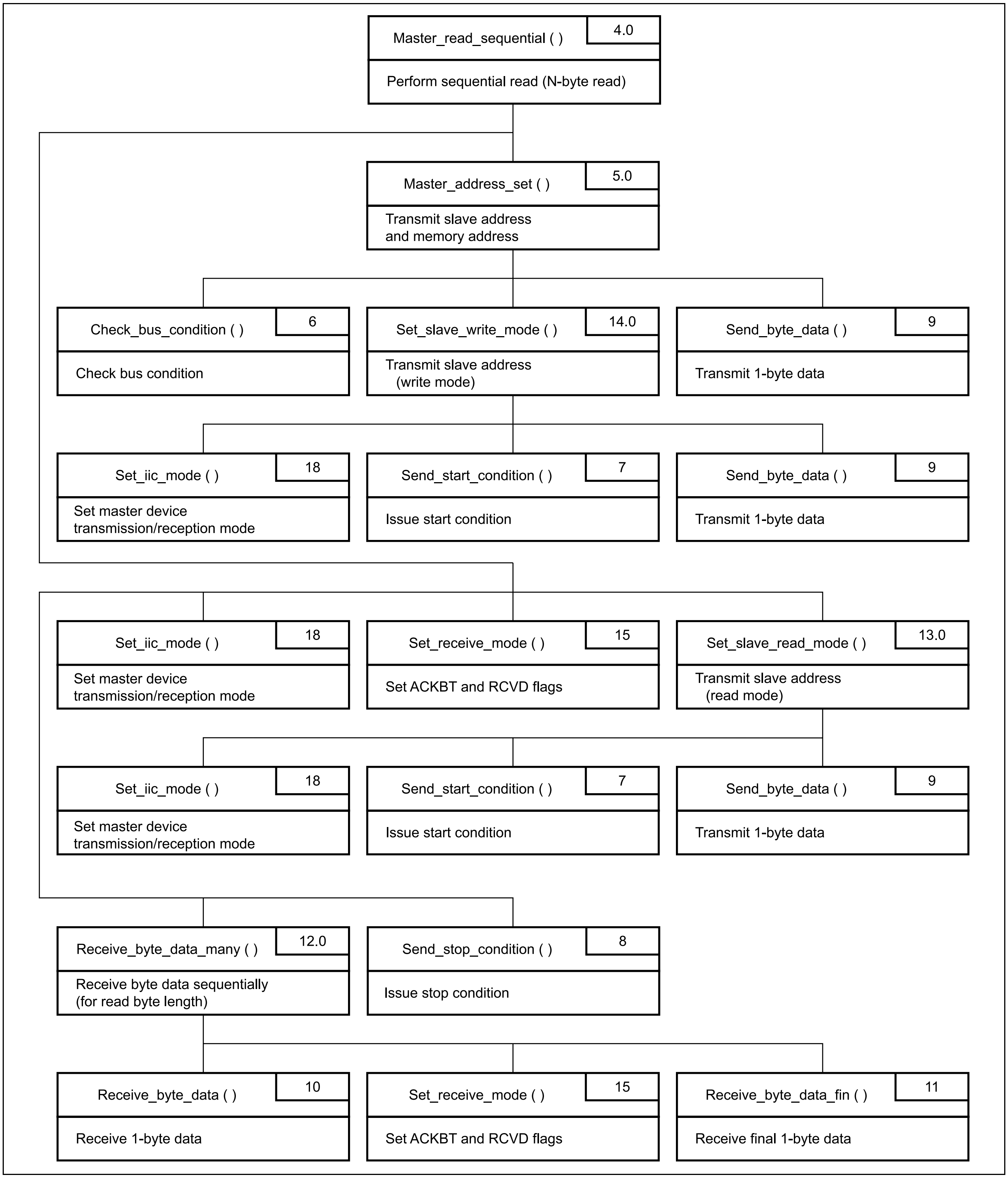
4. Master_read_byte_random ()



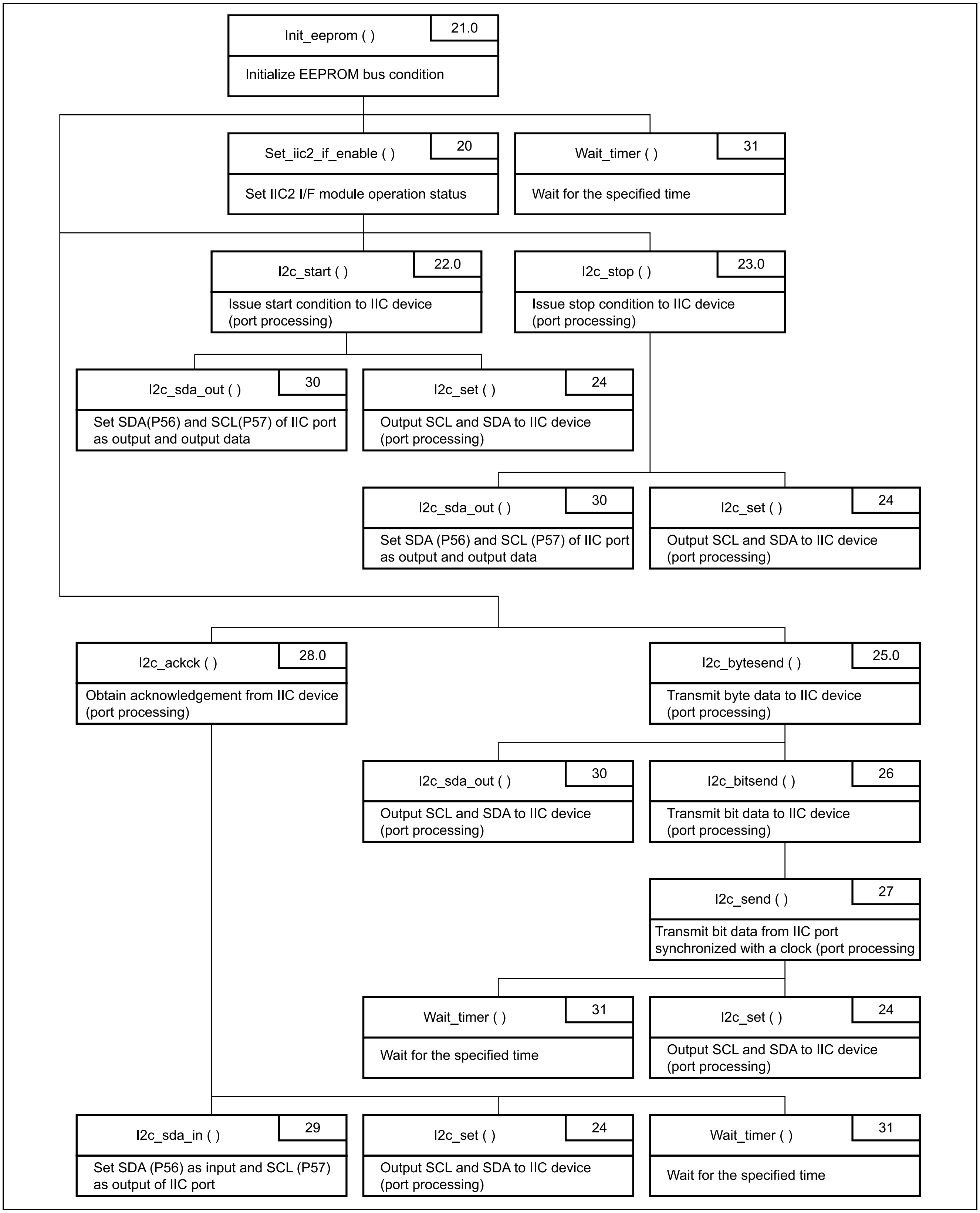
5. Master_page_write()



6. Master_read_sequential ()



7. Init_eeprom ()

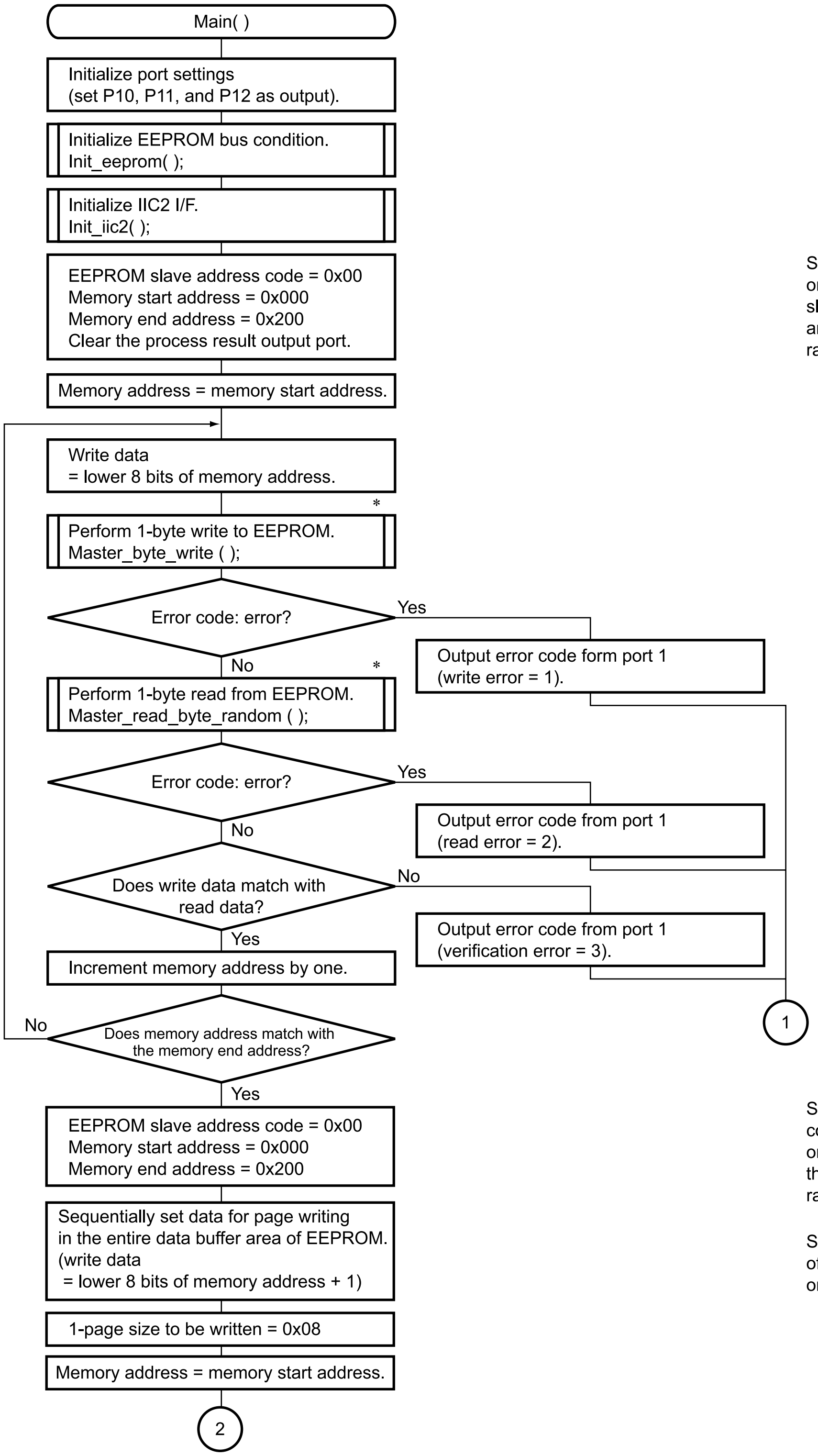

Init_eeprom ( )
21.0
Initialize EEPROM bus condition
Set_iic2_if_enable ( )
20
Set IIC2 I/F module operation status
Wait_timer ( )
31
Wait for the specified time
I2c_start ( )
22.0
Issue start condition to IIC device (port processing)
I2c_stop ( )
23.0
Issue stop condition to IIC device (port processing)
I2c_sda_out ( )
30
Set SDA(P56) and SCL(P57) of IIC port as output and output data
I2c_set ( )
24
Output SCL and SDA to IIC device (port processing)
I2c_sda_out ( )
30
Set SDA (P56) and SCL (P57) of IIC port as output and output data
I2c_set ( )
24
Output SCL and SDA to IIC device (port processing)
I2c_ackck ( )
28.0
Obtain acknowledgement from IIC device (port processing)
I2c_bytesend ( )
25.0
Transmit byte data to IIC device (port processing)
I2c_sda_out ( )
30
Output SCL and SDA to IIC device (port processing)
I2c_bitsend ( )
26
Transmit bit data to IIC device (port processing)
I2c_send ( )
27
Transmit bit data from IIC port synchronized with a clock (port processing
Wait_timer ( )
31
Wait for the specified time
I2c_set ( )
24
Output SCL and SDA to IIC device (port processing)
I2c_sda_in ( )
29
Set SDA (P56) as input and SCL (P57) as output of IIC port
I2c_set ( )
24
Output SCL and SDA to IIC device (port processing)
Wait_timer ( )
31
Wait for the specified time

5. Flowchart

Flowcharts of this application program are shown in [0] to [21] and [32].

Note that the numbers [0] to [21] and [32] correspond to function number shown in section 4.2, Function Lists (tables 4.2.1 to 4.2.5).

[0] Main () Function: Main processing

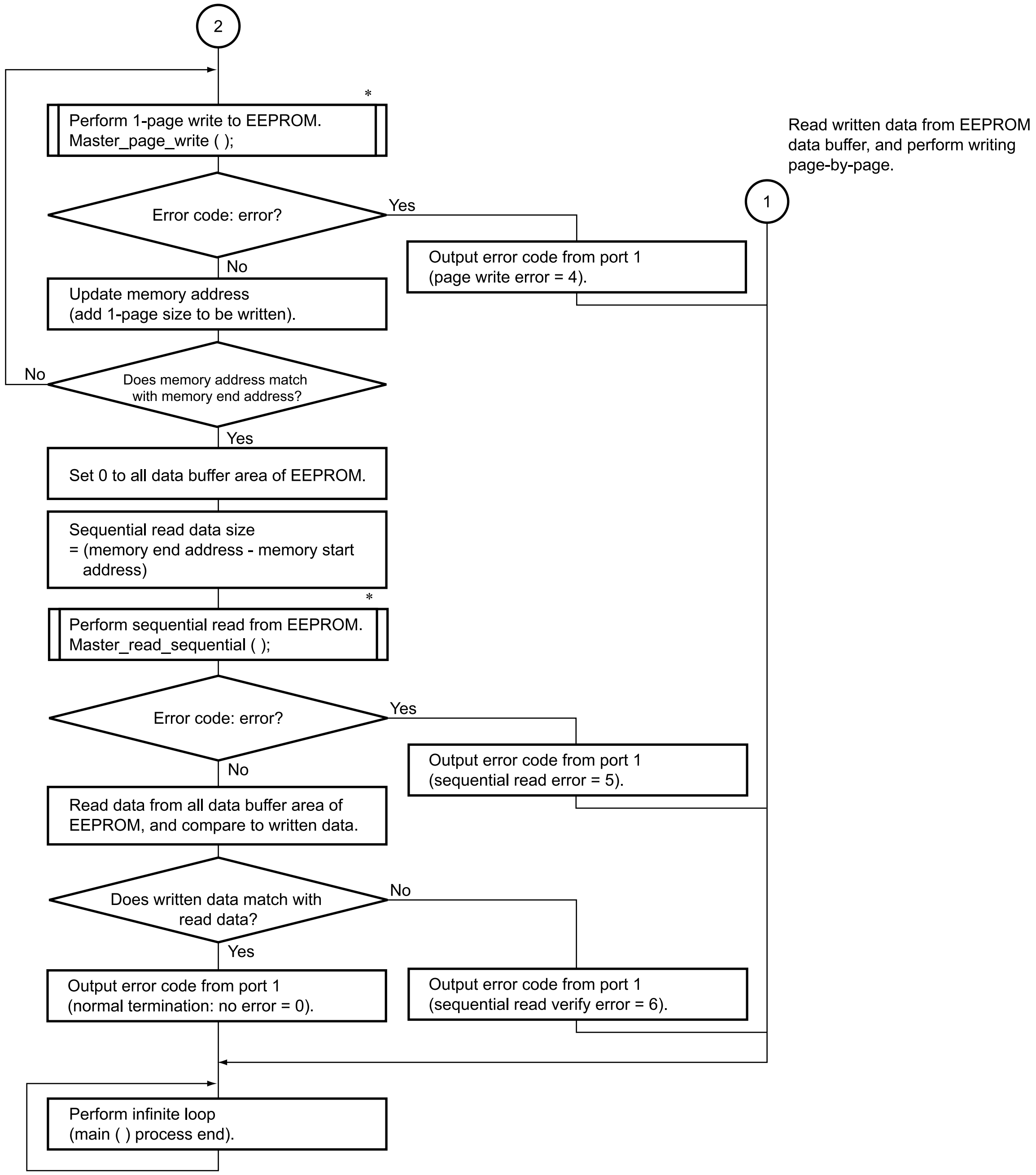

Main( )
Initialize port settings
(set P10, P11, and P12 as output).
Initialize EEPROM bus condition.
Init_eeprom( );
Initialize IIC2 I/F.
Init_iic2( );
EEPROM slave address code = 0x00
Memory start address = 0x000
Memory end address = 0x200
Clear the process result output port.
Memory address = memory start address.
Write data
= lower 8 bits of memory address.
*
Perform 1-byte write to EEPROM.
Master_byte_write ( );
Error code: error?
Yes
Output error code form port 1
(write error = 1).
No
*
Perform 1-byte read from EEPROM.
Master_read_byte_random ( );
Error code: error?
Yes
Output error code from port 1
(read error = 2).
No
Does write data match with
read data?
No
Output error code from port 1
(verification error = 3).
Yes
Increment memory address by one.
No
Does memory address match with
the memory end address?
1
Yes
EEPROM slave address code = 0x00
Memory start address = 0x000
Memory end address = 0x200
Sequentially set data for page writing
in the entire data buffer area of EEPROM.
(write data
= lower 8 bits of memory address + 1)
1-page size to be written = 0x08
Memory address = memory start address.
2

Set the H8/3694N's
on-chip EEPROM
slave address code (0)
and the memory address
range (512 bytes).

Set the slave address
code (0) of H8/3694N's
on-chip EEPROM and
the memory address
range (512 bytes).

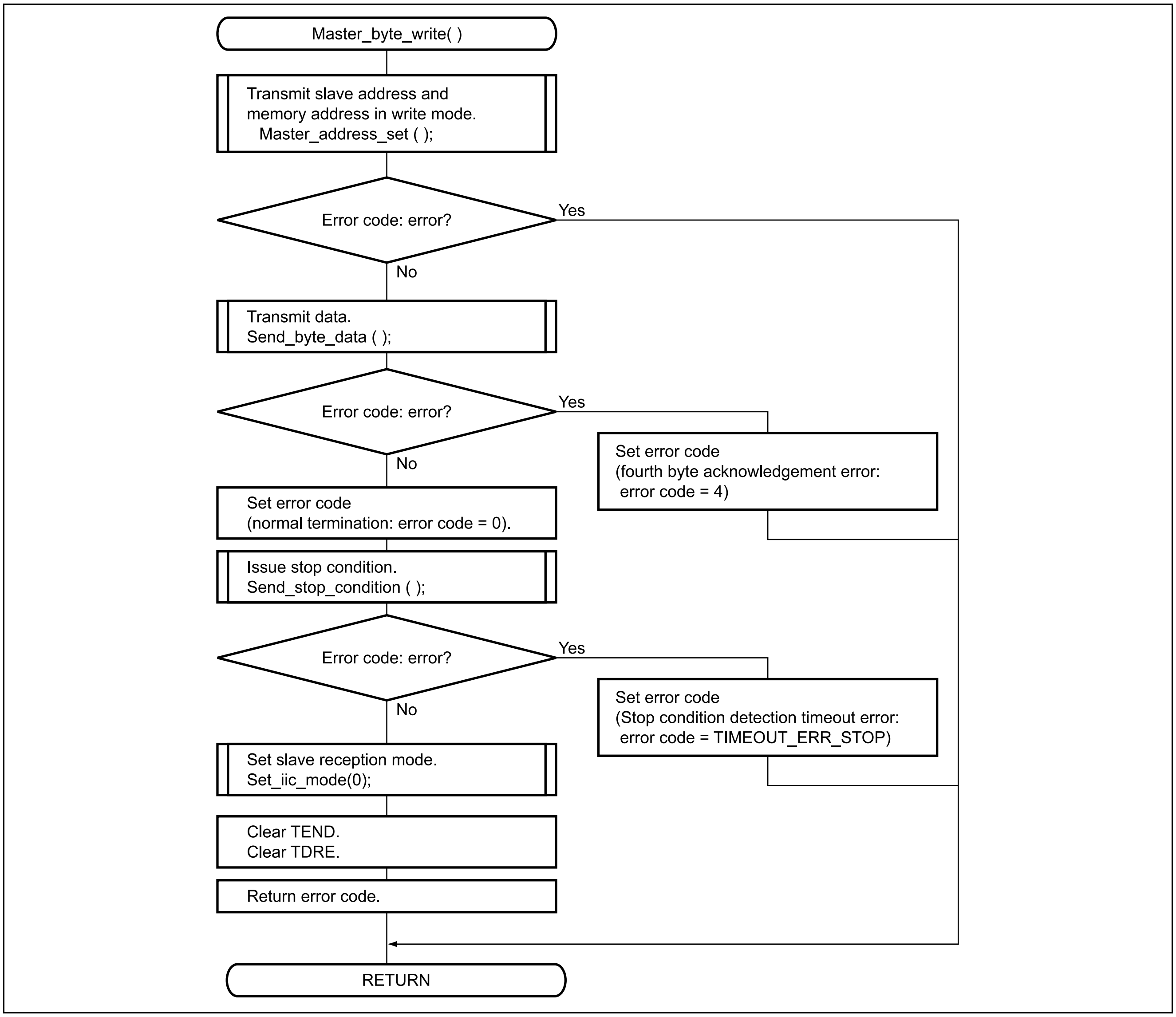
Set the page size (8 bytes)
of H8/3694N's
on-chip EEPROM.

Note: * Usage examples of 1-byte write (Master_byte_write) and 1-byte read (Master_read_byte_random)
Writes the lower 8-bit value of address to the corresponding EEPROM address (0x000 to 0x1ff), reads it, and verifies.
If an error occurs, output an error code from the process result output ports (P10, P11, and P12).

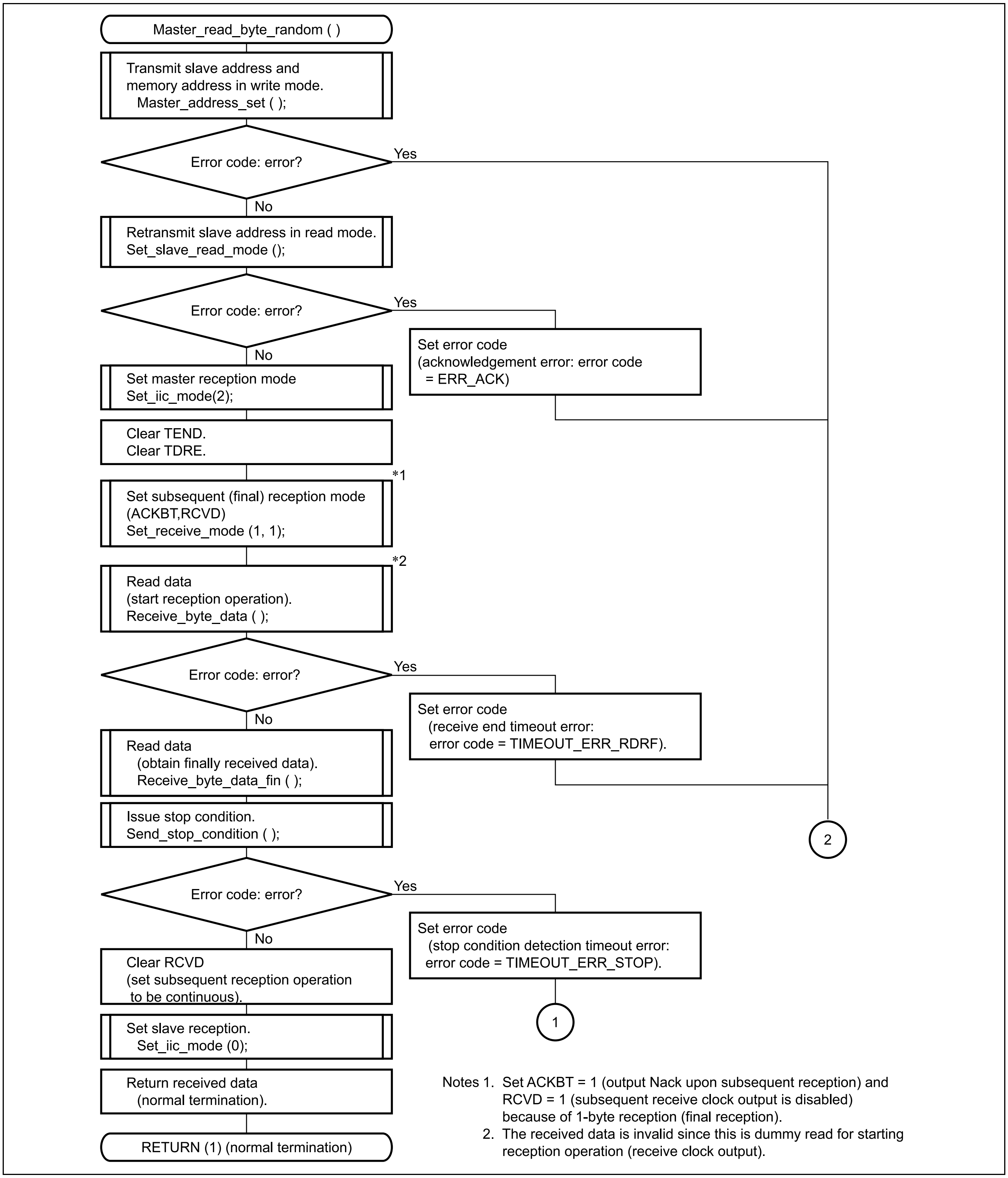

2
*
Perform 1-page write to EEPROM.
Master_page_write ( );
Read written data from EEPROM data buffer, and perform writing page-by-page.
1
Error code: error?
Yes
Output error code from port 1 (page write error = 4).
No
Update memory address (add 1-page size to be written).
No
Does memory address match with memory end address?
Yes
Set 0 to all data buffer area of EEPROM.
Sequential read data size = (memory end address - memory start address)
*
Perform sequential read from EEPROM.
Master_read_sequential ( );
Error code: error?
Yes
Output error code from port 1 (sequential read error = 5).
No
Read data from all data buffer area of EEPROM, and compare to written data.
Does written data match with read data?
No
Output error code from port 1 (sequential read verify error = 6).
Yes
Output error code from port 1 (normal termination: no error = 0).
Perform infinite loop (main ( ) process end).

Note: * Usage examples of page write (Master_page_write) and sequential read (Master_read_sequential)
Perform page write the value of lower 8 bits of the address + 1 to EEPROM addresses (0x000 to 0x1ff), perform sequential read, and verify.
If an error occurs, error code corresponding to processing is output from process result output ports (P10, P11, and P12).

[1] Master_byte_write () Function: 1-byte write

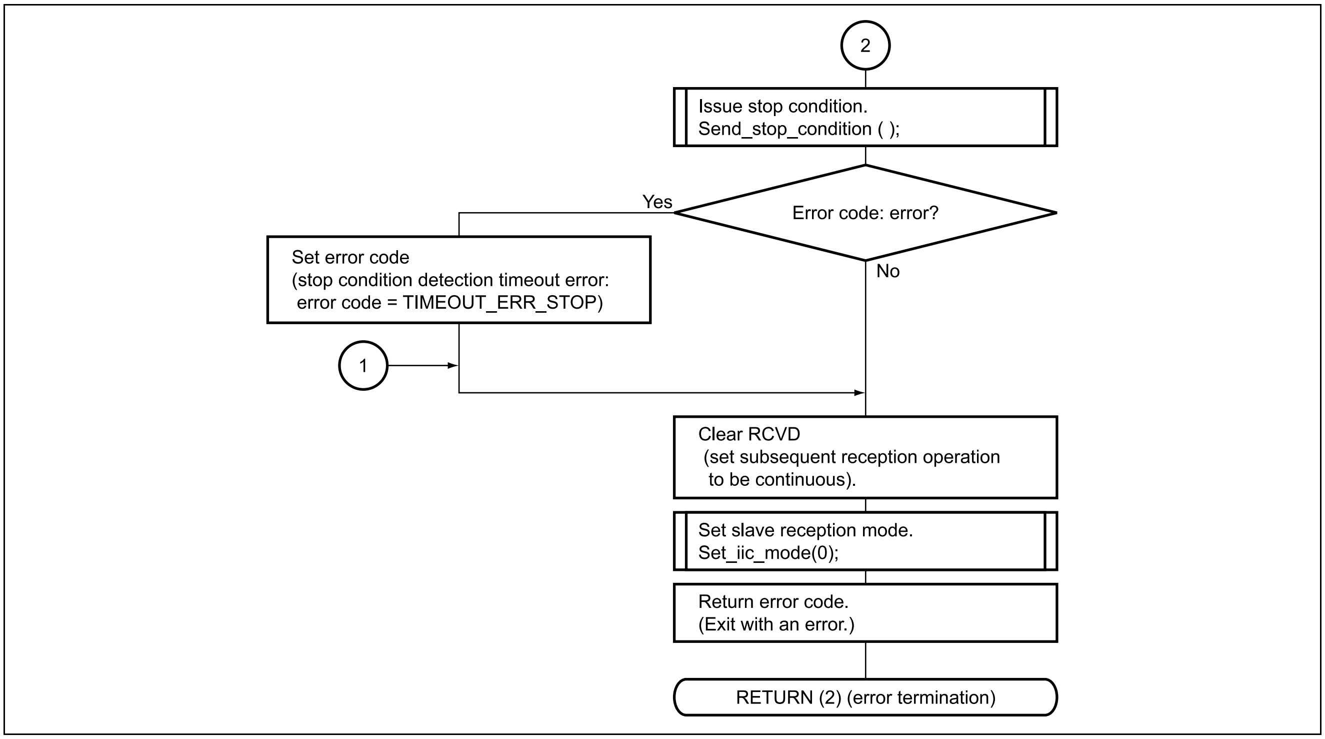

Master_byte_write( )
Transmit slave address and memory address in write mode. Master_address_set ( );
Error code: error?
Yes
No
Transmit data. Send_byte_data ( );
Error code: error?
Yes
No
Set error code (fourth byte acknowledgement error: error code = 4)
Set error code (normal termination: error code = 0).
Issue stop condition. Send_stop_condition ( );
Error code: error?
Yes
No
Set error code (Stop condition detection timeout error: error code = TIMEOUT_ERR_STOP)
Set slave reception mode. Set_iic_mode(0);
Clear TEND. Clear TDRE.
Return error code.
RETURN

[2] Master_read_byte_random () Function: 1-byte read

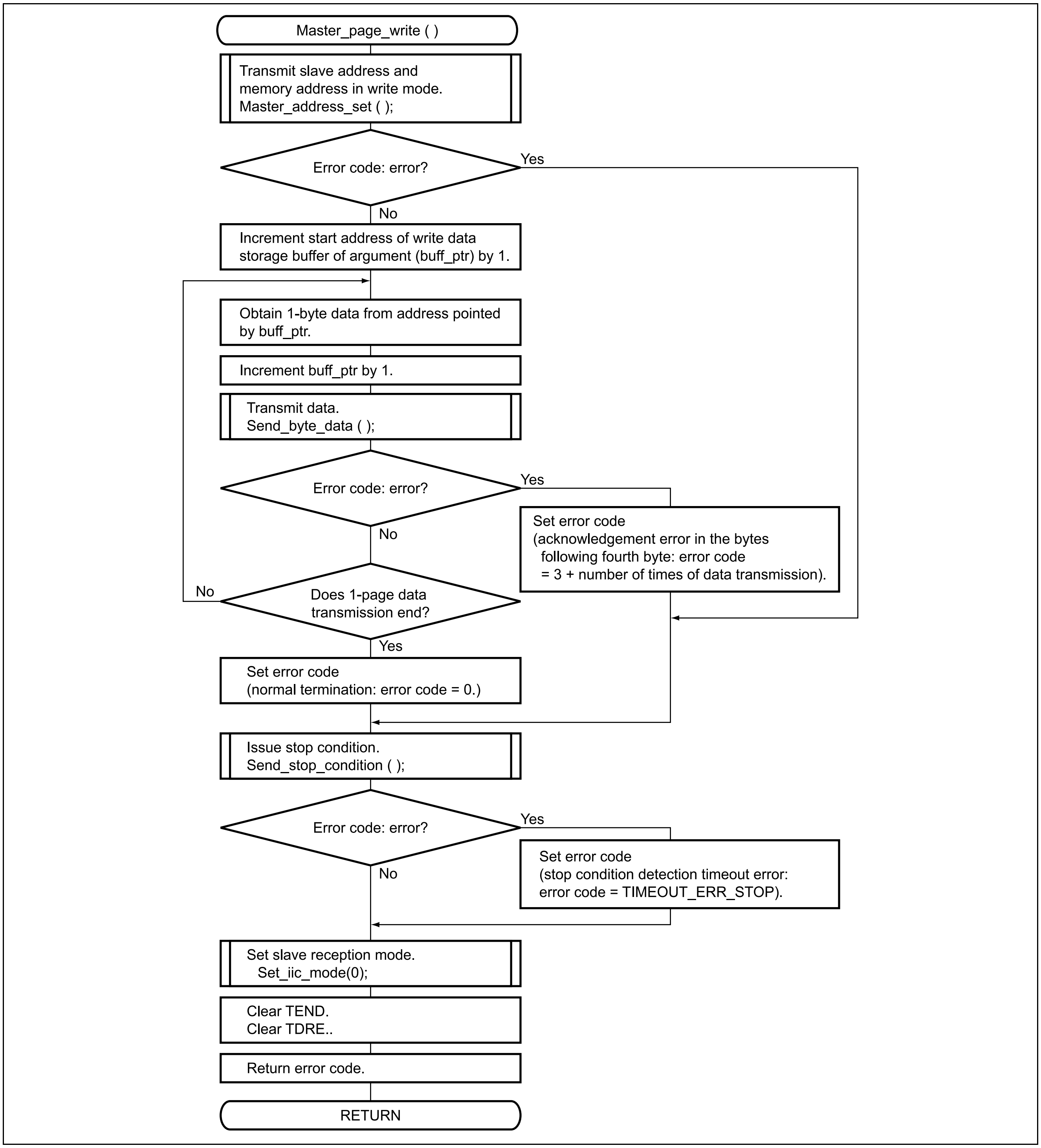

Master_read_byte_random ( )
Transmit slave address and memory address in write mode.
Master_address_set ( );
Error code: error?
Yes
No
Retransmit slave address in read mode.
Set_slave_read_mode ();
Error code: error?
Yes
No
Set error code (acknowledgement error: error code = ERR_ACK)
Set master reception mode
Set_iic_mode(2);
Clear TEND.
Clear TDRE.
*1
Set subsequent (final) reception mode (ACKBT,RCVD)
Set_receive_mode (1, 1);
*2
Read data (start reception operation).
Receive_byte_data ( );
Error code: error?
Yes
No
Set error code (receive end timeout error: error code = TIMEOUT_ERR_RDRF).
Read data (obtain finally received data).
Receive_byte_data_fin ( );
Issue stop condition.
Send_stop_condition ( );
2
Error code: error?
Yes
No
Set error code (stop condition detection timeout error: error code = TIMEOUT_ERR_STOP).
1
Clear RCVD (set subsequent reception operation to be continuous).
Set slave reception.
Set_iic_mode (0);
Return received data (normal termination).
RETURN (1) (normal termination)

Notes 1. Set ACKBT = 1 (output Nack upon subsequent reception) and RCVD = 1 (subsequent receive clock output is disabled) because of 1-byte reception (final reception).

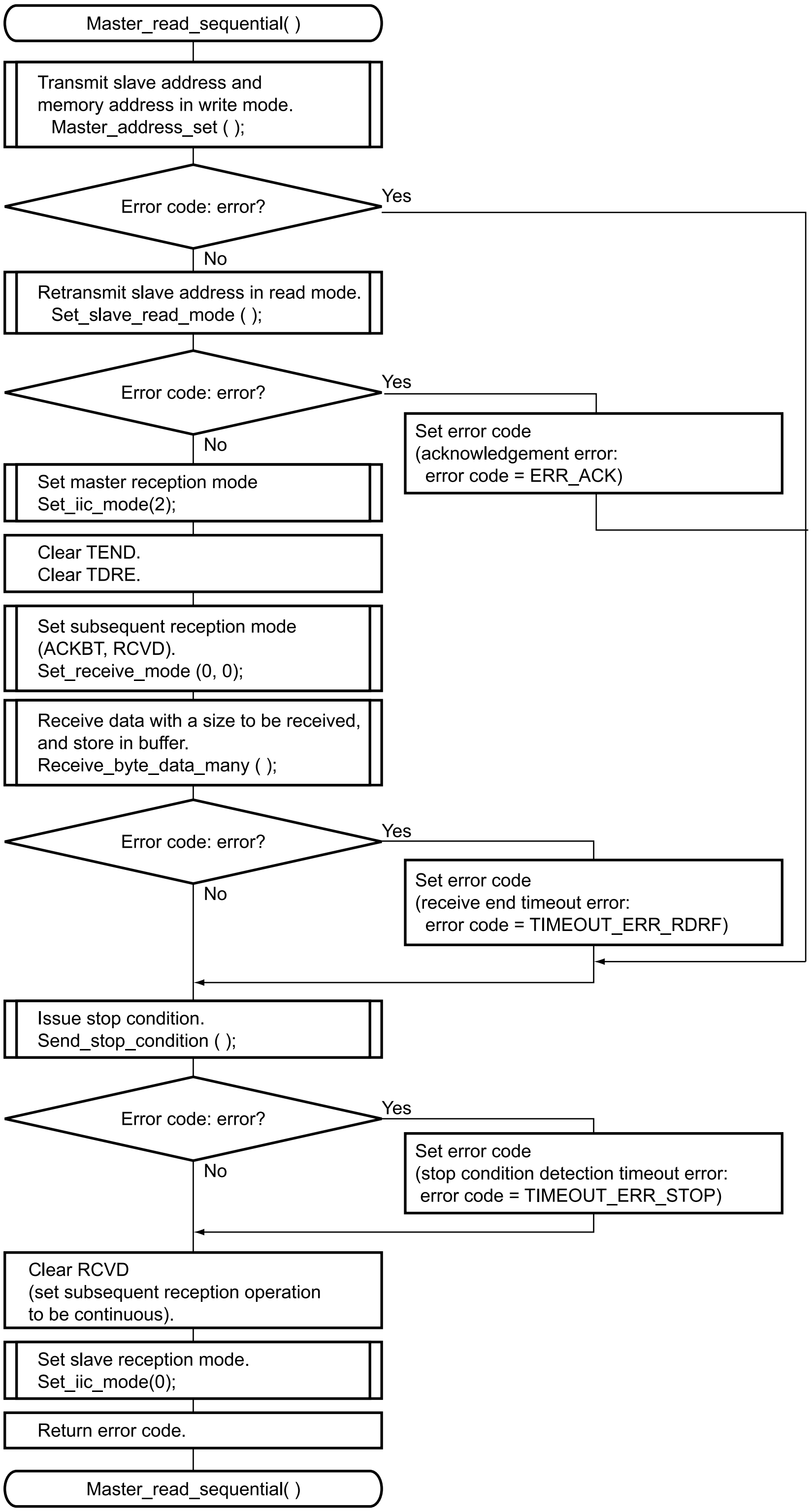
2. The received data is invalid since this is dummy read for starting reception operation (receive clock output).


2
Issue stop condition.
Send_stop_condition ( );
Error code: error?
Yes
No
Set error code
(stop condition detection timeout error:
error code = TIMEOUT_ERR_STOP)
1
Clear RCVD
(set subsequent reception operation
to be continuous).
Set slave reception mode.
Set_iic_mode(0);
Return error code.
(Exit with an error.)
RETURN (2) (error termination)

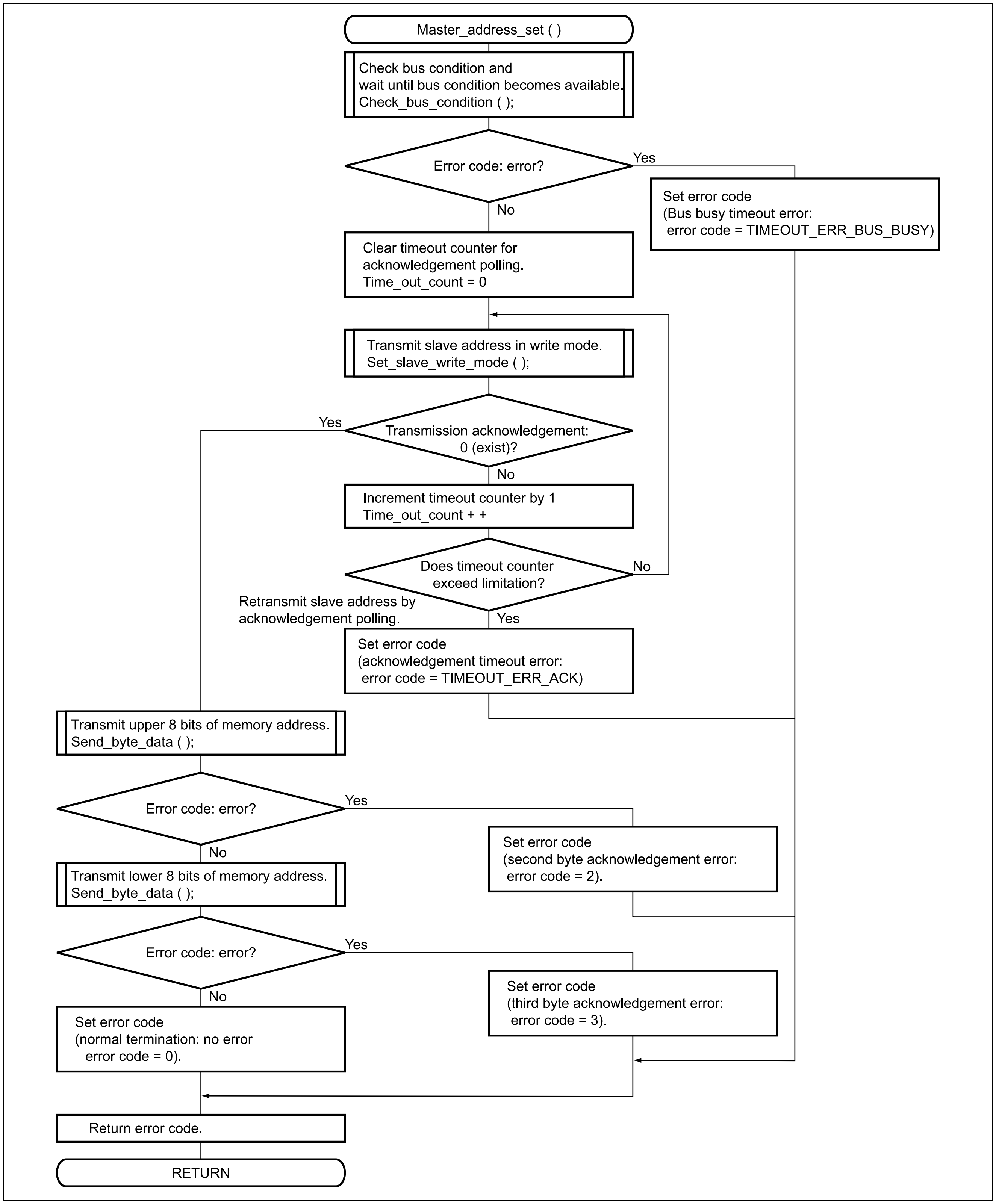
[3] Master_page_write () Function: 1-page write


Master_page_write ( )
Transmit slave address and memory address in write mode. Master_address_set ( );
Error code: error?
Yes
No
Increment start address of write data storage buffer of argument (buff_ptr) by 1.
Obtain 1-byte data from address pointed by buff_ptr.
Increment buff_ptr by 1.
Transmit data. Send_byte_data ( );
Error code: error?
Yes
No
Set error code (acknowledgement error in the bytes following fourth byte: error code = 3 + number of times of data transmission).
No
Does 1-page data transmission end?
Yes
Set error code (normal termination: error code = 0.)
Issue stop condition. Send_stop_condition ( );
Error code: error?
Yes
No
Set error code (stop condition detection timeout error: error code = TIMEOUT_ERR_STOP).
Set slave reception mode. Set_iic_mode(0);
Clear TEND. Clear TDRE..
Return error code.
RETURN

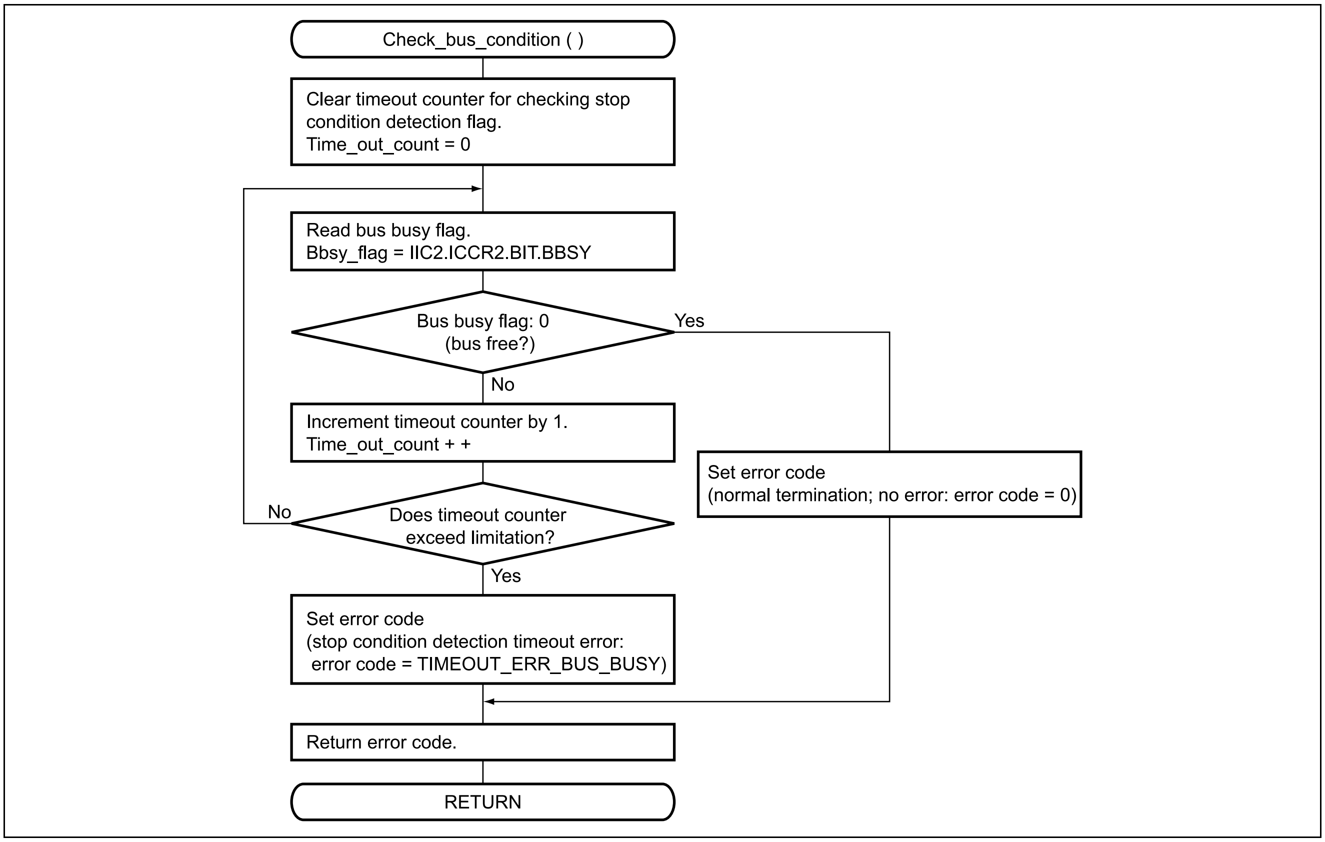
[4] Master_read_sequential () Function: sequential read


Master_read_sequential( )
Transmit slave address and memory address in write mode. Master_address_set ( );
Error code: error?
Yes
No
Retransmit slave address in read mode. Set_slave_read_mode ( );
Error code: error?
Yes
No
Set error code (acknowledgement error: error code = ERR_ACK)
Set master reception mode Set_iic_mode(2);
Clear TEND. Clear TDRE.
Set subsequent reception mode (ACKBT, RCVD). Set_receive_mode (0, 0);
Receive data with a size to be received, and store in buffer. Receive_byte_data_many ( );
Error code: error?
Yes
No
Set error code (receive end timeout error: error code = TIMEOUT_ERR_RDRF)
Issue stop condition. Send_stop_condition ( );
Error code: error?
Yes
No
Set error code (stop condition detection timeout error: error code = TIMEOUT_ERR_STOP)
Clear RCVD (set subsequent reception operation to be continuous).
Set slave reception mode. Set_iic_mode(0);
Return error code.
Master_read_sequential( )

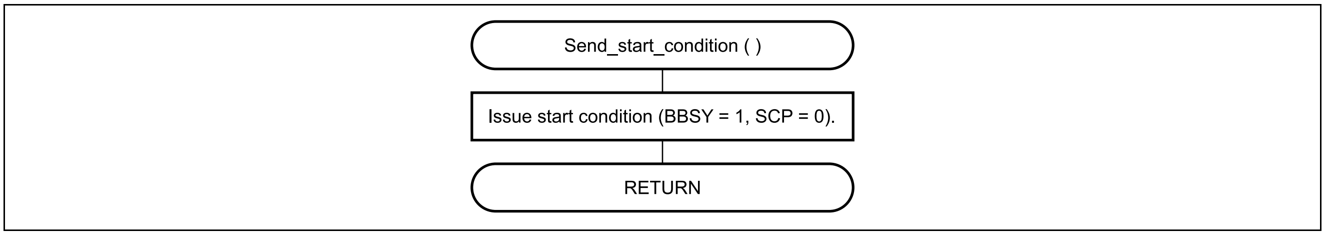
[5] Master_address_set () Function: Transmit slave address and memory address


Master_address_set ( )
Check bus condition and
wait until bus condition becomes available.
Check_bus_condition ( );
Error code: error?
Yes
Set error code
(Bus busy timeout error:
error code = TIMEOUT_ERR_BUS_BUSY)
No
Clear timeout counter for
acknowledgement polling.
Time_out_count = 0
Transmit slave address in write mode.
Set_slave_write_mode ( );
Yes
Transmission acknowledgement:
0 (exist)?
No
Increment timeout counter by 1
Time_out_count + +
Does timeout counter
exceed limitation?
No
Retransmit slave address by
acknowledgement polling.
Yes
Set error code
(acknowledgement timeout error:
error code = TIMEOUT_ERR_ACK)
Transmit upper 8 bits of memory address.
Send_byte_data ( );
Error code: error?
Yes
Set error code
(second byte acknowledgement error:
error code = 2).
No
Transmit lower 8 bits of memory address.
Send_byte_data ( );
Error code: error?
Yes
Set error code
(third byte acknowledgement error:
error code = 3).
No
Set error code
(normal termination: no error
error code = 0).
Return error code.
RETURN

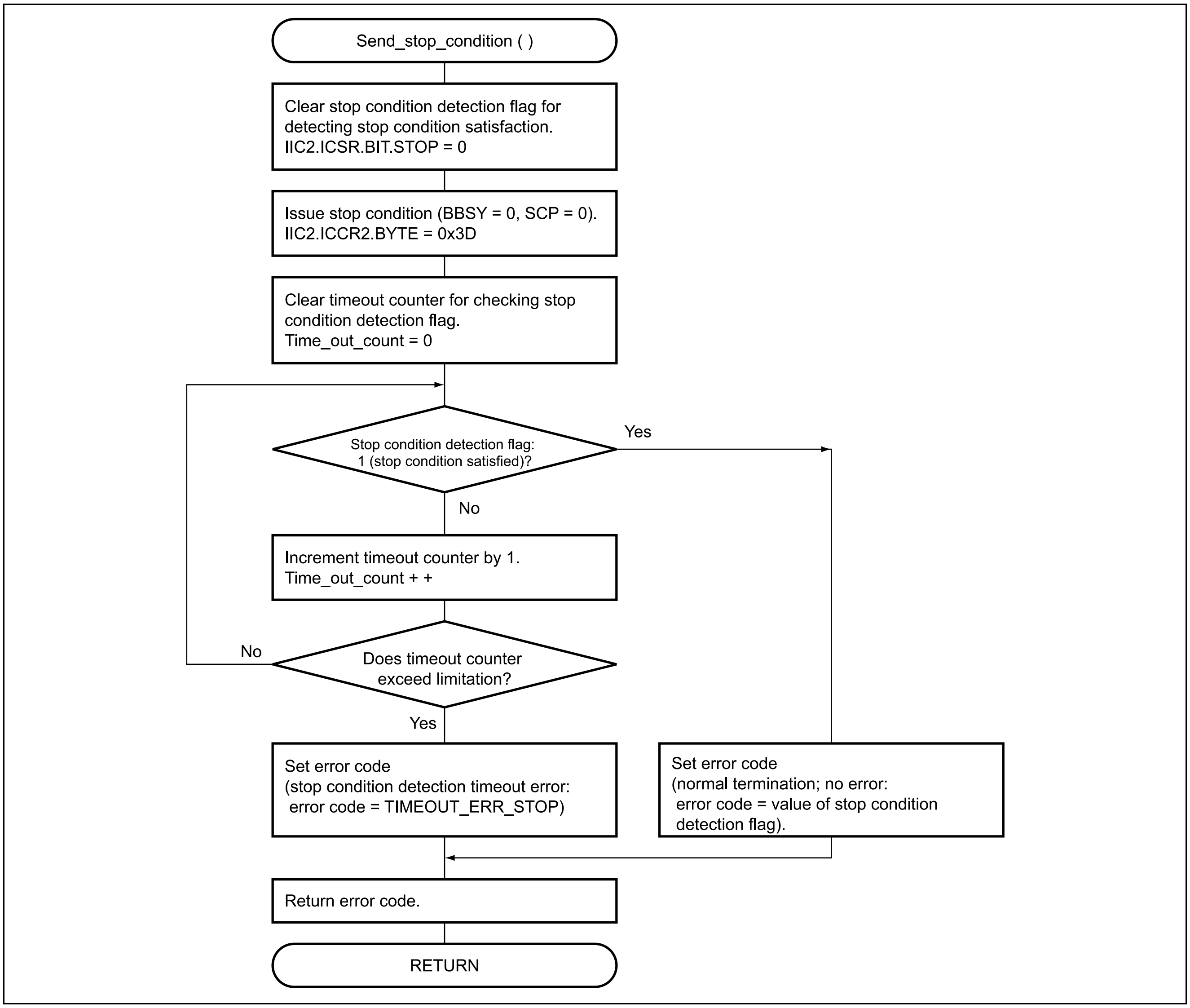
[6] Check_bus_condition () Function: Check IIC bus condition and wait until bus becomes available


Check_bus_condition ( )
Clear timeout counter for checking stop condition detection flag.
Time_out_count = 0
Read bus busy flag.
Bbsy_flag = IIC2.ICCR2.BIT.BBSY
Bus busy flag: 0 (bus free?)
Yes
No
Increment timeout counter by 1.
Time_out_count + +
Set error code (normal termination; no error: error code = 0)
No
Does timeout counter exceed limitation?
Yes
Set error code (stop condition detection timeout error: error code = TIMEOUT_ERR_BUS_BUSY)
Return error code.
RETURN

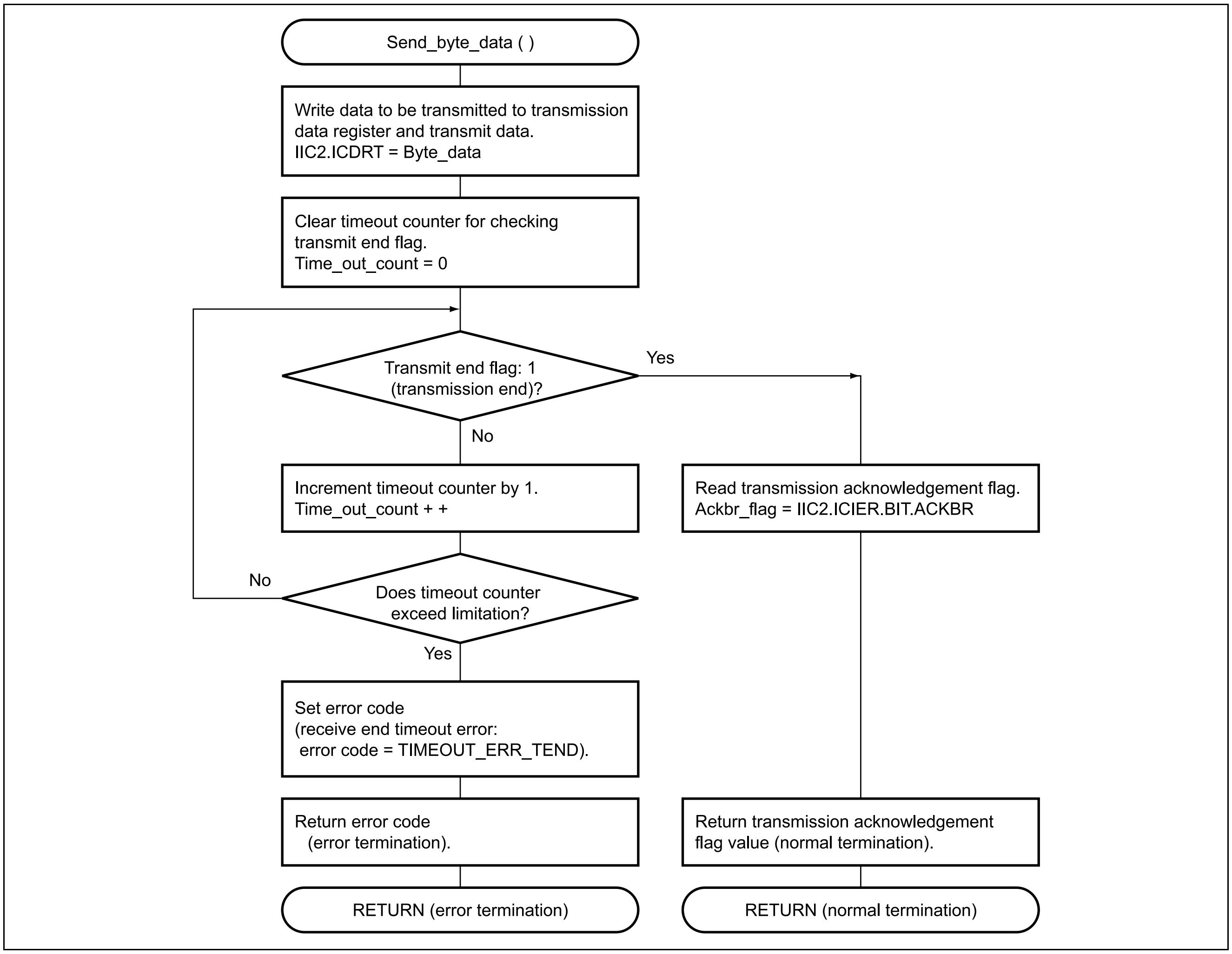
[7] Send_start_condition () Function: Issue start condition



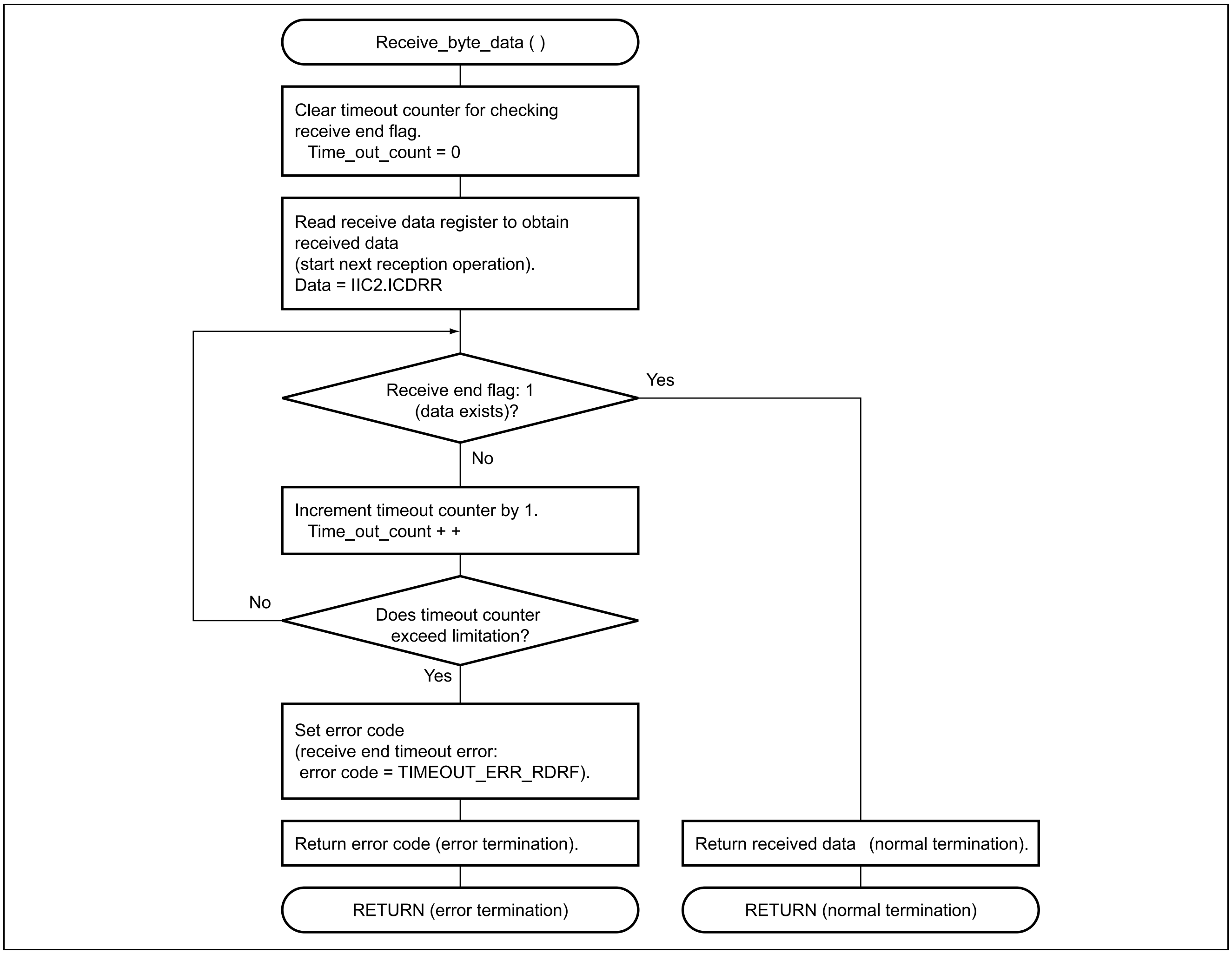
[8] Send_stop_condition () Function: Issue stop condition


Send_stop_condition ( )
Clear stop condition detection flag for detecting stop condition satisfaction.
IIC2.ICSR.BIT.STOP = 0
Issue stop condition (BBSY = 0, SCP = 0).
IIC2.ICCR2.BYTE = 0x3D
Clear timeout counter for checking stop condition detection flag.
Time_out_count = 0
Stop condition detection flag: 1 (stop condition satisfied)?
Yes
No
Increment timeout counter by 1.
Time_out_count + +
No
Does timeout counter exceed limitation?
Yes
Set error code (stop condition detection timeout error: error code = TIMEOUT_ERR_STOP)
Set error code (normal termination; no error: error code = value of stop condition detection flag).
Return error code.
RETURN

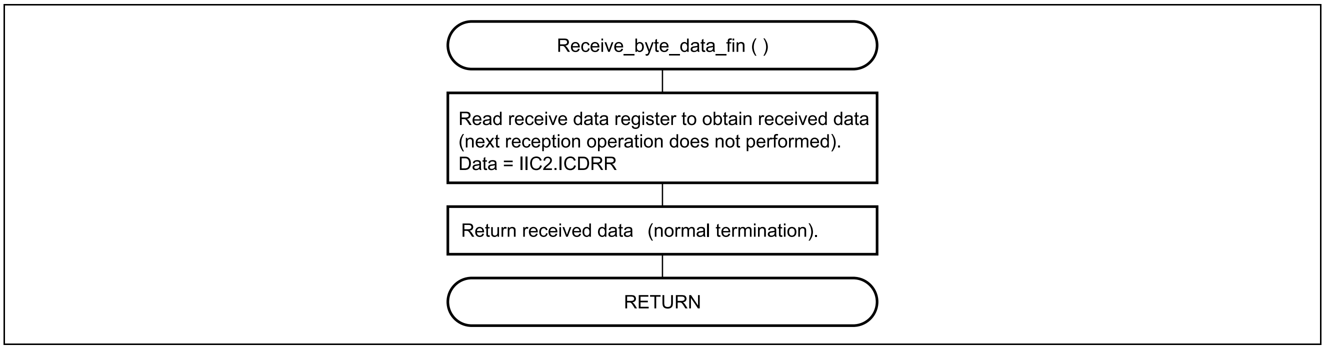
[9] Send_byte_data () Function: Perform 1-byte data transmission


Send_byte_data ( )
Write data to be transmitted to transmission data register and transmit data.
IIC2.ICDRT = Byte_data
Clear timeout counter for checking transmit end flag.
Time_out_count = 0
Transmit end flag: 1 (transmission end)?
Yes
No
Increment timeout counter by 1.
Time_out_count + +
Read transmission acknowledgement flag.
Ackbr_flag = IIC2.ICIER.BIT.ACKBR
No
Does timeout counter exceed limitation?
Yes
Set error code (receive end timeout error: error code = TIMEOUT_ERR_TEND).
Return error code (error termination).
Return transmission acknowledgement flag value (normal termination).
RETURN (error termination)
RETURN (normal termination)

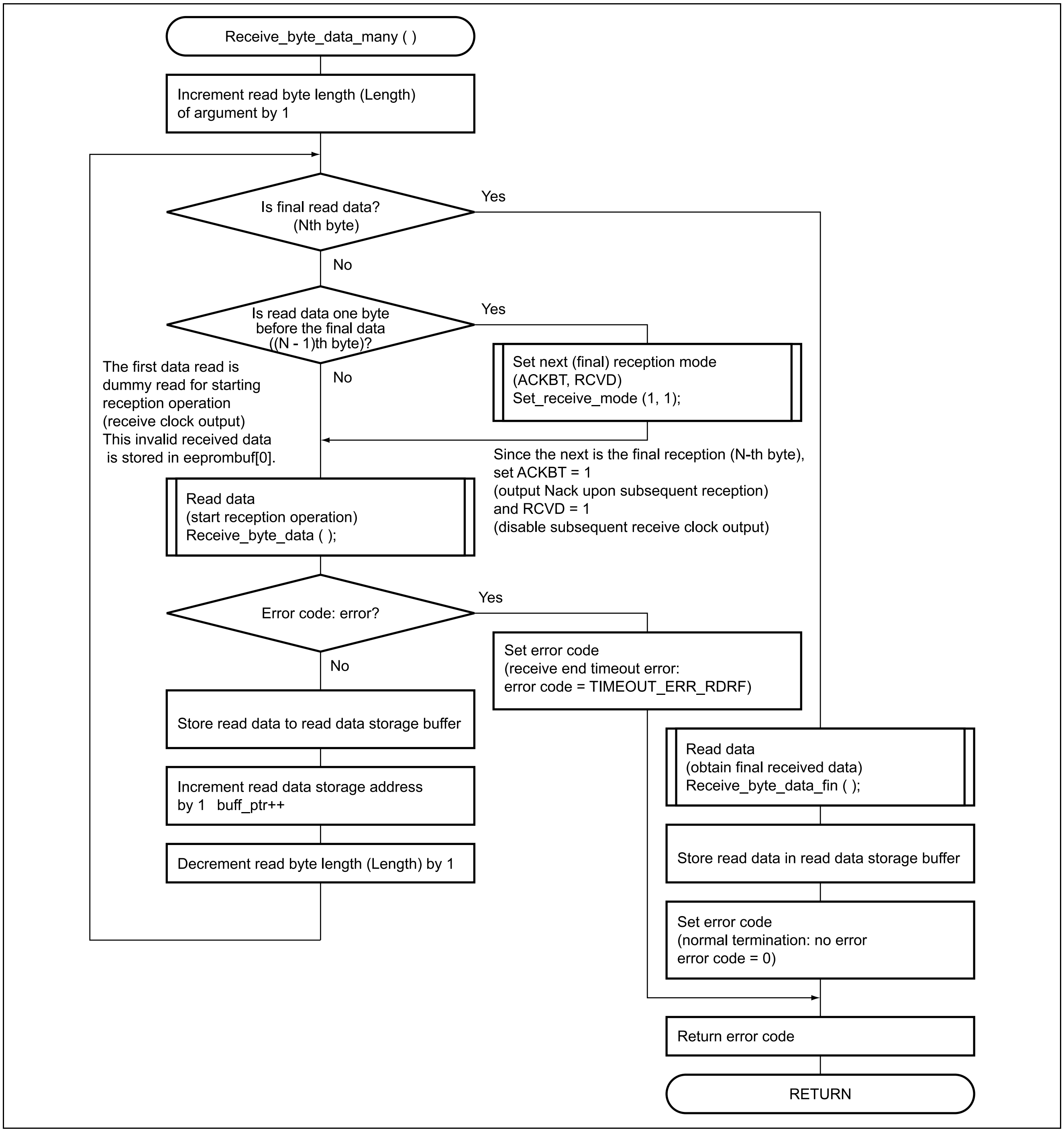
[10] Receive_byte_data () Function: 1-byte data reception


Receive_byte_data ( )
Clear timeout counter for checking receive end flag.
Time_out_count = 0
Read receive data register to obtain received data (start next reception operation).
Data = IIC2.ICDRR
Receive end flag: 1 (data exists)?
Yes
No
Increment timeout counter by 1.
Time_out_count + +
No
Does timeout counter exceed limitation?
Yes
Set error code (receive end timeout error: error code = TIMEOUT_ERR_RDRF).
Return error code (error termination).
Return received data (normal termination).
RETURN (error termination)
RETURN (normal termination)

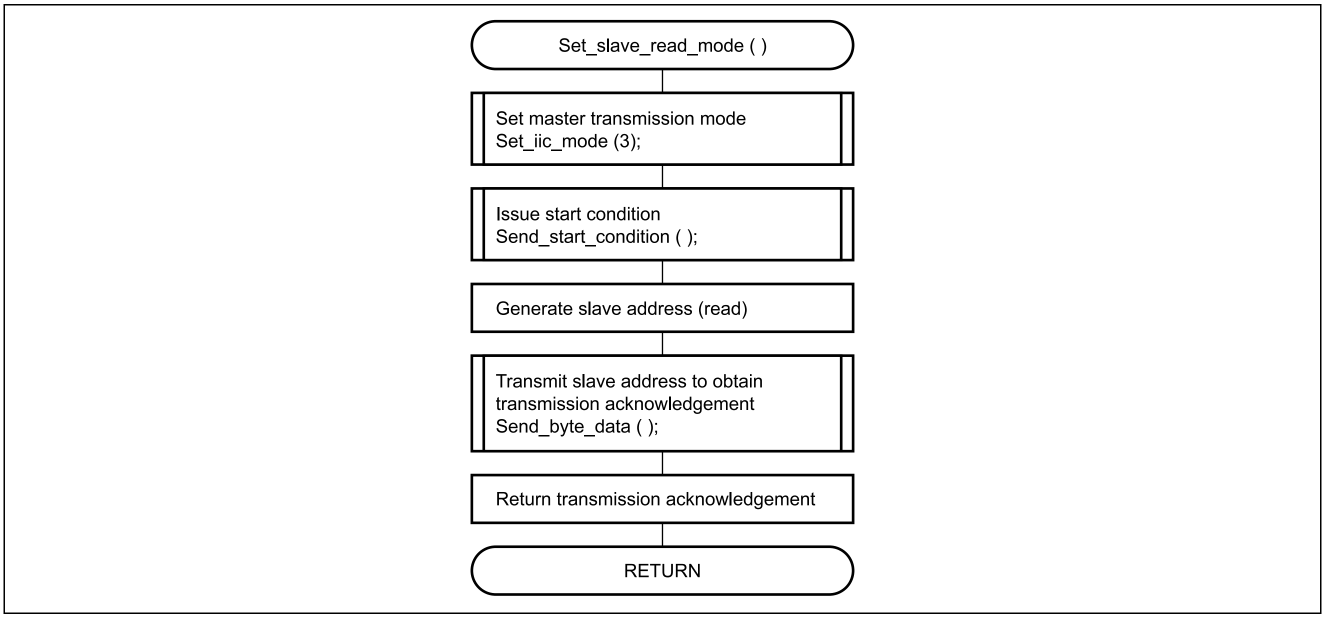
[11] Receive_byte_data_fin () Function: Perform final 1-byte data reception


Receive_byte_data_fin ( )
Read receive data register to obtain received data (next reception operation does not performed).
Data = IIC2.ICDRR
Return received data (normal termination).
RETURN

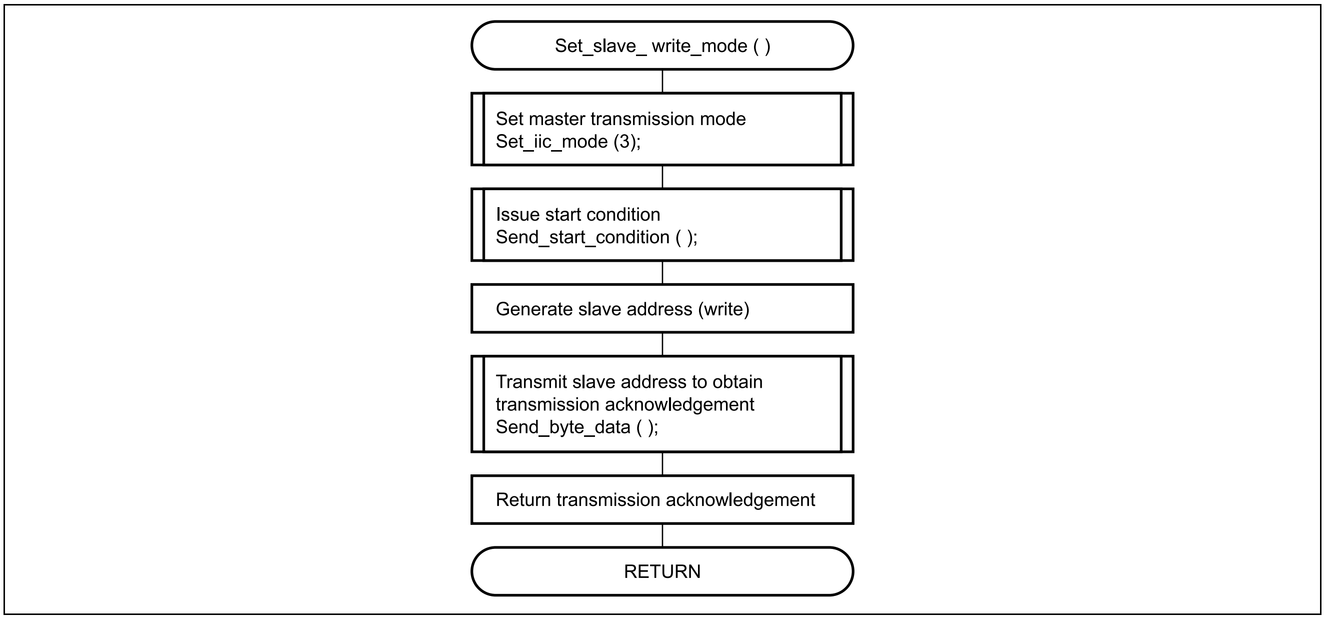
[12] Receive_byte_data_many () (1) Function: Receive byte data sequentially


Receive_byte_data_many ( )
Increment read byte length (Length) of argument by 1
Is final read data? (Nth byte)
Yes
No
Is read data one byte before the final data ((N - 1)th byte)?
Yes
Set next (final) reception mode (ACKBT, RCVD) Set_receive_mode (1, 1);
No
The first data read is dummy read for starting reception operation (receive clock output) This invalid received data is stored in eeprombuf[0].
Since the next is the final reception (N-th byte), set ACKBT = 1 (output Nack upon subsequent reception) and RCVD = 1 (disable subsequent receive clock output)
Read data (start reception operation) Receive_byte_data ( );
Error code: error?
Yes
Set error code (receive end timeout error: error code = TIMEOUT_ERR_RDRF)
No
Store read data to read data storage buffer
Read data (obtain final received data) Receive_byte_data_fin ( );
Increment read data storage address by 1 buff_ptr++
Store read data in read data storage buffer
Decrement read byte length (Length) by 1
Set error code (normal termination: no error error code = 0)
Return error code
RETURN

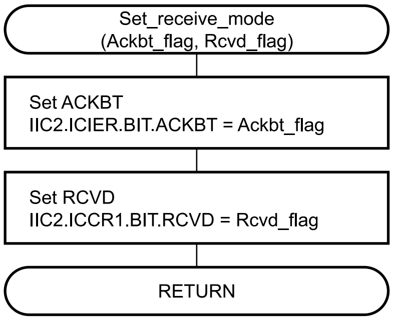
[13] Set_slave_read_mode () Function: Transmit slave address (read mode)


Set_slave_read_mode ( )
Set master transmission mode
Set_iic_mode (3);
Issue start condition
Send_start_condition ( );
Generate slave address (read)
Transmit slave address to obtain
transmission acknowledgement
Send_byte_data ( );
Return transmission acknowledgement
RETURN

[14] Set_slave_write_mode() Function: Transmit slave address (write mode)

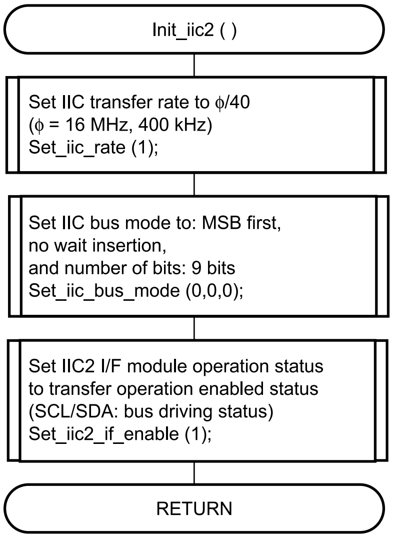

Set_slave_ write_mode ( )
Set master transmission mode
Set_iic_mode (3);
Issue start condition
Send_start_condition ( );
Generate slave address (write)
Transmit slave address to obtain
transmission acknowledgement
Send_byte_data ( );
Return transmission acknowledgement
RETURN

[15] Set_receive_mode () Function: Set ACKBT and RCVD flags for receiving data

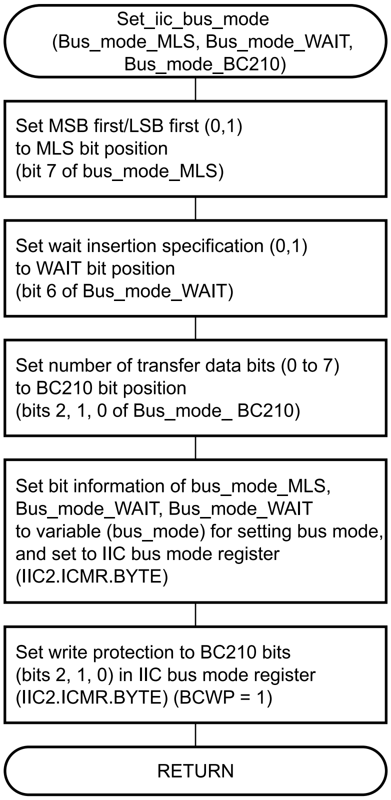

Set_receive_mode
(Ackbt_flag, Rcvd_flag)
Set ACKBT
IIC2.ICIER.BIT.ACKBT = Ackbt_flag
Set RCVD
IIC2.ICCR1.BIT.RCVD = Rcvd_flag
RETURN

Note (1) Settings prior to sequential reception:
Set ACKBT = 0 (output Ack upon subsequent reception)
and RCVD = 0 (enable subsequent receive clock output),
and read data.
(2) Settings before receiving data one byte before the final data ((N - 1)th byte):
Set ACKBT = 1 (output Nack upon subsequent reception)
and RCVD = 1 (disable subsequent receive clock output),
and read data.

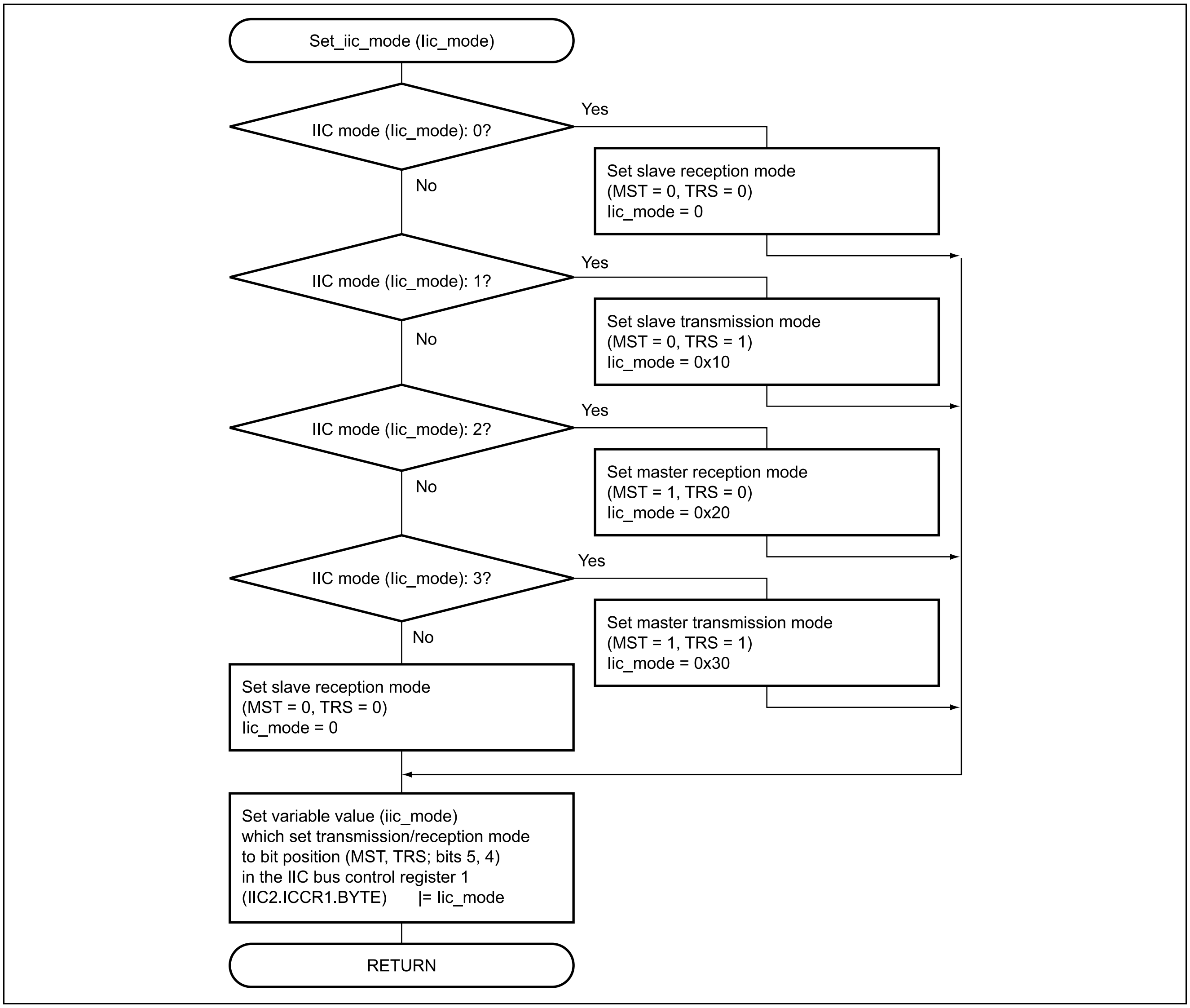
[16] Init_iic2 () Function: Initialize IIC2 I/F


Init_iic2 ( )
Set IIC transfer rate to φ/40
(φ = 16 MHz, 400 kHz)
Set_iic_rate (1);
Set IIC bus mode to: MSB first,
no wait insertion,
and number of bits: 9 bits
Set_iic_bus_mode (0,0,0);
Set IIC2 I/F module operation status
to transfer operation enabled status
(SCL/SDA: bus driving status)
Set_iic2_if_enable (1);
RETURN

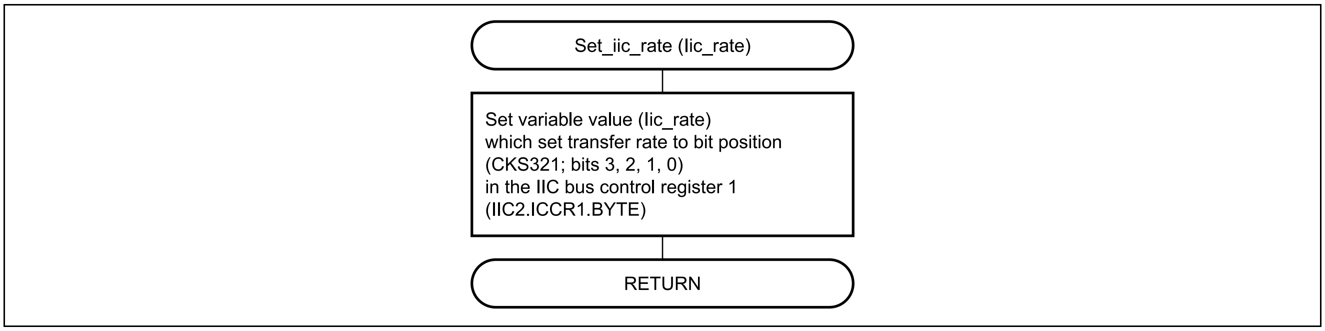
[17] Set_iic_bus_mode () Function: Set master device bus mode (MLS, WAIT, BC210)


Set_iic_bus_mode
(Bus_mode_MLS, Bus_mode_WAIT,
Bus_mode_BC210)
Set MSB first/LSB first (0,1)
to MLS bit position
(bit 7 of bus_mode_MLS)
Set wait insertion specification (0,1)
to WAIT bit position
(bit 6 of Bus_mode_WAIT)
Set number of transfer data bits (0 to 7)
to BC210 bit position
(bits 2, 1, 0 of Bus_mode_ BC210)
Set bit information of bus_mode_MLS,
Bus_mode_WAIT, Bus_mode_WAIT
to variable (bus_mode) for setting bus mode,
and set to IIC bus mode register
(IIC2.ICMR.BYTE)
Set write protection to BC210 bits
(bits 2, 1, 0) in IIC bus mode register
(IIC2.ICMR.BYTE) (BCWP = 1)
RETURN

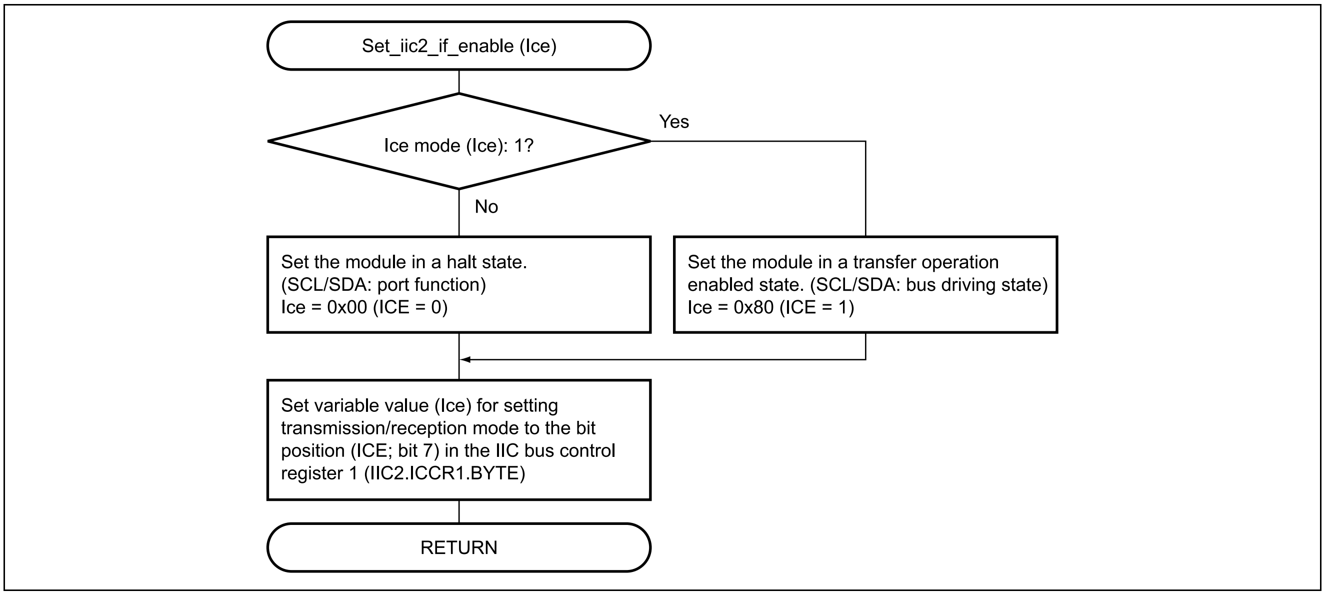
[18] Set_iic_mode () Function: Set master device transmission/reception mode (MST, TRS)



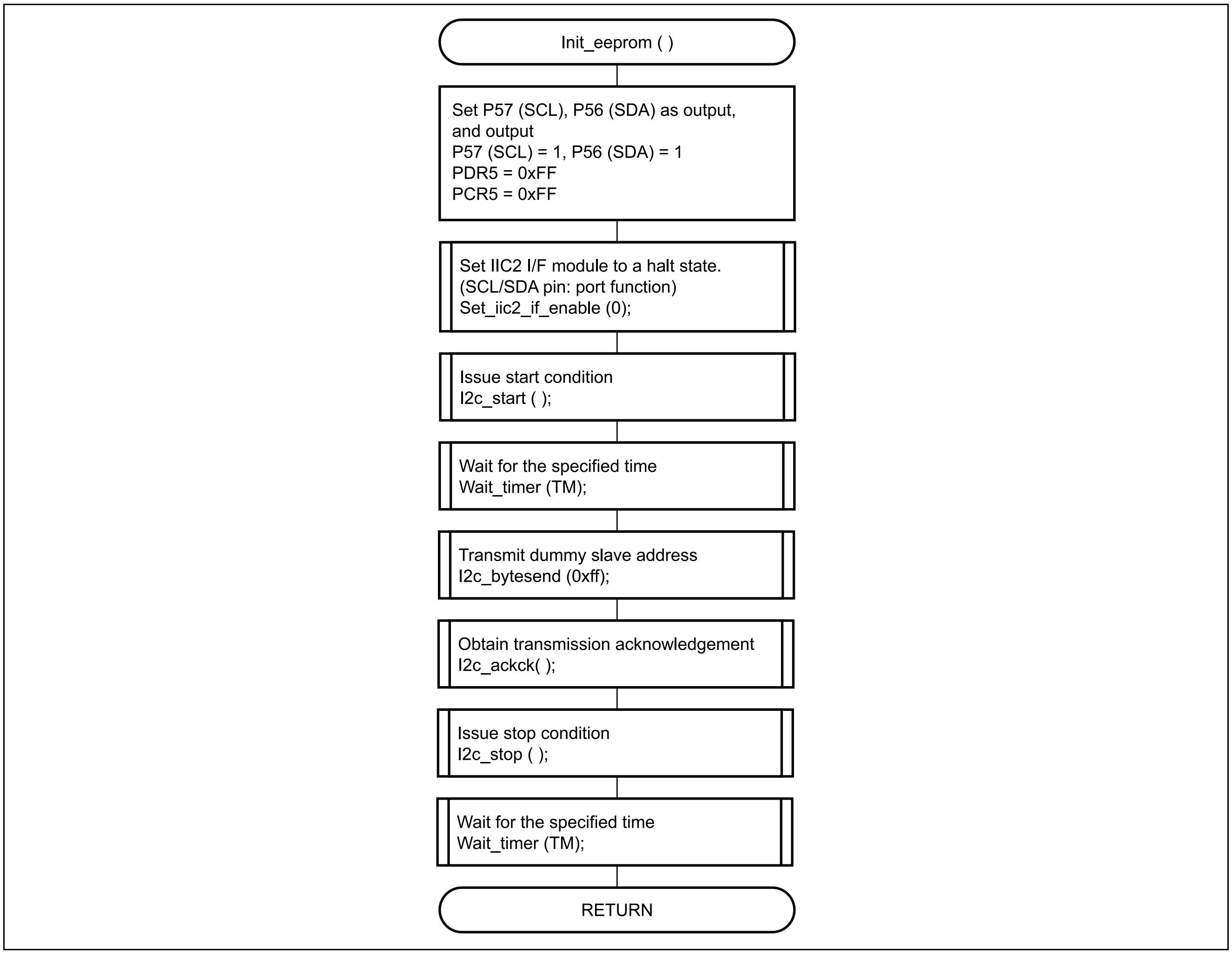
[19] Set_iic_rate () Function: Set master device transfer rate (CKS3, 2, 1, 0)


Set_iic_rate (Iic_rate)
Set variable value (Iic_rate)
which set transfer rate to bit position
(CKS321; bits 3, 2, 1, 0)
in the IIC bus control register 1
(IIC2.ICCR1.BYTE)
RETURN

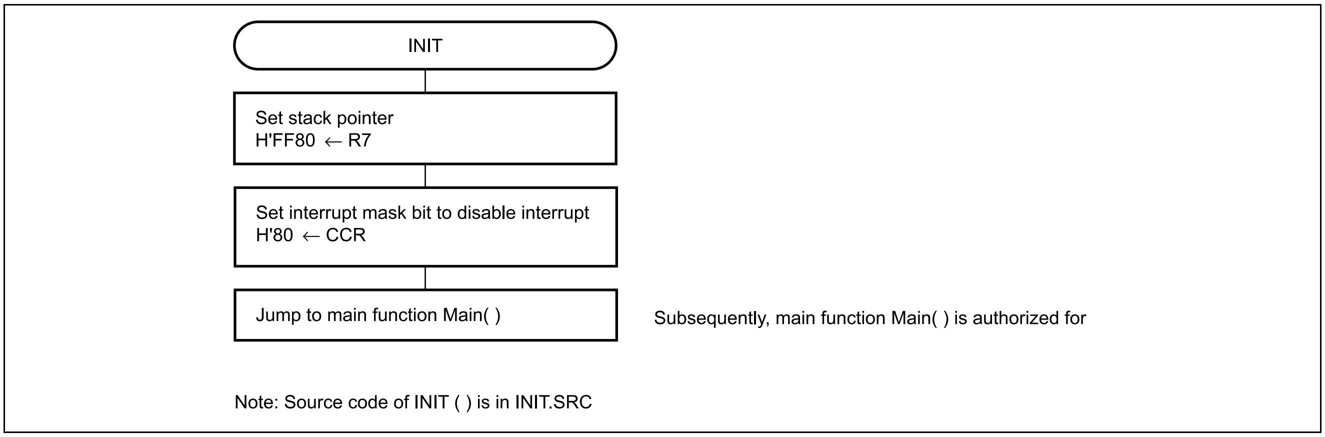
[20] Set_iic2_if_enable () Function: Set the IIC2 I/F module's operation status bit (ICE)


Set_iic2_if_enable (Ice)
Ice mode (Ice): 1?
Yes
No
Set the module in a halt state. (SCL/SDA: port function) Ice = 0x00 (ICE = 0)
Set the module in a transfer operation enabled state. (SCL/SDA: bus driving state) Ice = 0x80 (ICE = 1)
Set variable value (Ice) for setting transmission/reception mode to the bit position (ICE; bit 7) in the IIC bus control register 1 (IIC2.ICCR1.BYTE)
RETURN

[21] Init_eeprom () Function: Initialize EEPROM bus condition


Init_eeprom ( )
Set P57 (SCL), P56 (SDA) as output, and output P57 (SCL) = 1, P56 (SDA) = 1
PDR5 = 0xFF
PCR5 = 0xFF
Set IIC2 I/F module to a halt state. (SCL/SDA pin: port function)
Set_iic2_if_enable (0);
Issue start condition
I2c_start ( );
Wait for the specified time
Wait_timer (TM);
Transmit dummy slave address
I2c_bytesend (0xff);
Obtain transmission acknowledgement
I2c_ackck( );
Issue stop condition
I2c_stop ( );
Wait for the specified time
Wait_timer (TM);
RETURN

[32] INIT() Function: Initialization (start up)


INIT
Set stack pointer
H'FF80 ← R7
Set interrupt mask bit to disable interrupt
H'80 ← CCR
Jump to main function Main( )
Subsequently, main function Main( ) is authorized for

Note: Source code of INIT ( ) is in INIT.SRC

6. Program Listing

Program list of this application program is shown in section 6, [1] to [4].

```
/********************************************************************************************************************************/
/*   H8/3694, H8/3687 IIC2 Include File                 Ver 1.1                                                                 */
/********************************************************************************************************************************/
struct st_iic2 {                                                                  /* struct IIC                                 */
           union {                                                                /* ICCR1                                      */
                 unsigned char BYTE;                                              /* Byte Access                                */
                 struct {                                                         /* Bit Access                                 */
                       unsigned char ICE :1;                                      /* ICE                                        */
                       unsigned char RCVD:1;                                      /* RCVD                                       */
                       unsigned char MST :1;                                      /* MST                                        */
                       unsigned char TRS :1;                                      /* TRS                                        */
                       unsigned char CKS3:1;                                      /* CKS3                                       */
                       unsigned char CKS2:1;                                      /* CKS2                                       */
                       unsigned char CKS1:1;                                      /* CKS1                                       */
                       unsigned char CKS0:1;                                      /* CKS0                                       */
                       }      BIT;                                                /*                                            */
                 }           ICCR1;                                               /*                                            */
           union {                                                                /* ICCR2                                      */
                 unsigned char BYTE;                                              /* Byte Access                                */
                 struct {                                                         /* Bit Access                                 */
                       unsigned char BBSY  :1;                                    /* BBSY                                       */
                       unsigned char SCP   :1;                                    /* SCP                                        */
                       unsigned char SDAO  :1;                                    /* SDAO                                       */
                       unsigned char SDAOP :1;                                    /* SDAOP                                      */
                       unsigned char SCLO  :1;                                    /* SCLO                                       */
                       unsigned char       :1;                                    /*                                            */
                       unsigned char IICRST:1;                                    /* IICRST                                     */
                       unsigned char       :1;                                    /*                                            */
                       }      BIT;                                                /*                                            */
                 }           ICCR2;                                               /*                                            */
           union {                                                                /* ICMR                                       */
                 unsigned char BYTE;                                              /* Byte Access                                */
                 struct {                                                         /* Bit Access                                 */
                       unsigned char MLS   :1;                                    /* MLS                                        */
                       unsigned char WAIT  :1;                                    /* WAIT                                       */
                       unsigned char       :2;                                    /*                                            */
                       unsigned char BCWP  :1;                                    /* BCWP                                       */
                       unsigned char BC2   :1;                                    /* BC2                                        */
                       unsigned char BC1   :1;                                    /* BC1                                        */
                       unsigned char BC0   :1;                                    /* BC0                                        */
                       }      BIT;                                                /*                                            */
                 }           ICMR;                                                /*                                            */
```

```
            union {                                                                /* ICIER                                      */
                 unsigned char BYTE;                                               /* Byte Access                                */
                 struct {                                                          /* Bit Access                                 */
                       unsigned char TIE   :1;                                     /* TIE                                        */
                       unsigned char TEIE  :1;                                     /* TEIE                                       */
                       unsigned char RIE   :1;                                     /* RIE                                        */
                       unsigned char NAKIE :1;                                     /* NAKIE                                      */
                       unsigned char STIE  :1;                                     /* STIE                                       */
                       unsigned char ACKE  :1;                                     /* ACKE                                       */
                       unsigned char ACKBR :1;                                     /* ACKBR                                      */
                       unsigned char ACKBT :1;                                     /* ACKBT                                      */
                       }      BIT;                                                 /*                                            */
                 }           ICIER;                                                /*                                            */
            union {                                                                /* ICSR                                       */
                 unsigned char BYTE;                                               /* Byte Access                                */
                 struct {                                                          /* Bit Access                                 */
                       unsigned char TDRE :1;                                      /* TDRE                                       */
                       unsigned char TEND :1;                                      /* TEND                                       */
                       unsigned char RDRF :1;                                      /* RDRF                                       */
                       unsigned char NACKF:1;                                      /* NACKF                                      */
                       unsigned char STOP :1;                                      /* STOP                                       */
                       unsigned char AL_OVE :1;                                    /* AL/OVE                                     */
                       unsigned char AAS  :1;                                      /* AAS                                        */
                       unsigned char ADZ  :1;                                      /* ADZ                                        */
                       }      BIT;                                                 /*                                            */
                 }           ICSR;                                                 /*                                            */
            union {                                                                /* SAR                                        */
                 unsigned char BYTE;                                               /* Byte Access                                */
                 struct {                                                          /* Bit Access                                 */
                       unsigned char SVAX:7;                                       /* SVA6-0                                     */
                       unsigned char FS  :1;                                       /* FS                                         */
                       }      BIT;                                                 /*                                            */
                 }           SAR;                                                  /*                                            */
            unsigned char     ICDRT;                                               /* ICDRT                                      */
            unsigned char     ICDRR;                                               /* ICDRR                                      */
};                                                                                 /*                                            */

#define IIC2   (*(volatile struct st_iic2  *)0xF748)                               /* IIC2 Address                               */
```

[2] File name: INIT.SRC Function: Initialization (start up)

```
;*********************************************************************************************************************************
;*   H8/3694, H8/3687 IIC2 APPLICATION NOTE STARTUP ROUTINE  Ver 1.1                                                             *
;*********************************************************************************************************************************
;
     EXPORT _INIT
     IMPORT _main
;
     SECTION    P,CODE
_INIT:
     MOV.W     #H'FF80,R7
     LDC.B     #B'10000000,CCR
     JMP       @_main
;
     .END

/*------------------------------------------------------------------------------------------------------------------------------*/
/*   H8/3694, H8/3687 on-chip I2Cbus interface 2 (IIC2) Application note                                                        */
/*   Application examples for reading from/writing to serial EEPROM (main program)                                              */
/*                                                                                                                              */
/*   1. Target EEPROM:   EEPROM for which memory address is specified in two bytes                                              */
/*   2. Operation confirmed products:                                                                                           */
/*                   (1) H8/3687N,H8/3694N(512B) with on-chip EEPROM (manufactured by RENESAS TECHNOLOGY)                       */
/*                  (2) HN58X2432I(4kB),HN58X2464I(8kB) EEPROM (manufactured by RENESAS TECHNOLOGY)                             */
/*                                                                                                                              */
/*      Ver 1.1     2003.5.9                                                                                                    */
/*                                                                                                                              */
/*        Copyright (C) RENESAS TECHNOLOGY 2003                                                                                 */
/*------------------------------------------------------------------------------------------------------------------------------*/
/********************************************************************************************************************************/
/*   Header Definition                                                                                                          */
/********************************************************************************************************************************/

#include    <machine.h>

/********************************************************************************************************************************/
/*   Function Prototype declarations                                                                                            */
/********************************************************************************************************************************/

extern  void    INIT( void );

extern  unsigned short  Master_read_byte_random( unsigned char Device_id, unsigned short Addr );
extern  unsigned char   Master_read_sequential( unsigned char Device_id, unsigned short Addr, unsigned short Length, unsigned char
*buff_ptr );
extern  unsigned char   Master_byte_write( unsigned char Device_id, unsigned short Addr, unsigned char Data);
extern  unsigned char   Master_page_write( unsigned char Device_id, unsigned short Addr, unsigned char Length, unsigned char
*buff_ptr );

extern void Init_eeprom( void );

void main ( void );
```

[3] File name: eeprom.c (2) Function: Main processing

```
/********************************************************************************************************************************/
/*   Vector address definition                                                                                                  */
/********************************************************************************************************************************/
#pragma section                                                            /* VECTOR SECTOIN SET                                */
void (*const VEC_TBL1[])(void) = {                                         /* 0x00 - 0x0f                                       */
     INIT,                                                                 /* reset vector (address 0):                         */
                                                                           /* call initialization function                      */
};

#ifdef __cplusplus
extern "C" {
#endif
void abort(void);
#ifdef __cplusplus
}
#endif

/********************************************************************************************************************************/
/*   RAM allocated section definition                                                                                           */
/********************************************************************************************************************************/
#pragma section                                                            /* section:B                                         */
unsigned char   eeprom_buf[513];                                           /* Definition of EEPROM data storage buffer          */
                                                                           /* (512 bytes)                                       */
                                                                           /* Buffer size is reserved as number of data bytes   */
                                                                           /* to be stored + 1                                  */

/********************************************************************************************************************************/
/*   ROM allocated section definition                                                                                           */
/********************************************************************************************************************************/
#pragma section                                                            /* section:P                                         */

/********************************************************************************************************************************/
/*   Main program                                                                                                               */
/********************************************************************************************************************************/
#pragma option nooptimize

/*------------------------------------------------------------------------------------------------------------------------------*/
/*   Process result output port address definition                                                                              */
/*------------------------------------------------------------------------------------------------------------------------------*/
#define PCR1   (*(volatile unsigned char *)0xFFE4)                         /* Port control register 1                           */
#define PDR1   (*(volatile unsigned char *)0xFFD4)                         /* Port data register 1                              */

/*------------------------------------------------------------------------------------------------------------------------------*/
/*   IIC2 port address definition                                                                                               */
/*------------------------------------------------------------------------------------------------------------------------------*/
#define PCR5   (*(volatile unsigned char *)0xFFE8)                         /* Port control register 5                           */
#define PDR5   (*(volatile unsigned char *)0xFFD8)                         /* Port data register 5                              */
void main(void){

unsigned char  Device_id;
unsigned short Address;
unsigned short Address_start;
unsigned short Address_end;

unsigned short Rezult_code;
unsigned char  Err_code;
unsigned char  Data_w;
unsigned char  Data_r;
unsigned short Length;
```

```
PDR1 = 0x00;                                                               /* Port1 = 0                                         */
PCR1 = 0x07;                                                               /* Set P10, P11, P12 as output                       */
                                                                           /* (used for process result output)                  */


Init_eeprom();                                                             /* Initialize EEPROM bus condition                   */
Init_iic2();                                                               /* Initialize IIC2 I/F                               */

/*============================================================================================================================*/
/*   Usage examples of 1-byet write (Master_byte_write) and1-byte read (Master_read_byte_random)                              */
/*                                                                                                                            */
/*   Write lower 8-bit value of address to EEPROM address (0x000 to 0x1ff), read, and verify                                  */
/*   When an error occurs, output error code corresponding to process from process result output port (P10, P11, P12)         */
/*============================================================================================================================*/
     Device_id = 0x00;                                                     /* Set EEPROM slave address code: 0 (A2,A1,A0)       */
     Address_start = 0x000;                                                /* Set EEPROM memory start address                   */
     Address_end = 0x200;                                                  /* Set EEPROM memory end address                     */
     PDR1 = 0x00;                                                          /* Clear process result output port                  */

     for (Address = Address_start; Address < Address_end; Address++){
          Data_w = (unsigned char) (Address & 0x00FF);                     /* Obtain lower 8-bit value of address               */
          Err_code = Master_byte_write (Device_id, Address, Data_w);       /* Perform 1-byte write to EEPROM                    */
          if (Err_code != 0){                                              /* Check error code, and if an error occurs,         */
               PDR1 = 0x01;                                                /* set error code (write error = 1), and             */
               goto Exit_err;                                              /* exit with an error                                */
          }

          Rezult_code = Master_read_byte_random (Device_id, Address);      /* Perform 1-byte read from EEPROM                   */
          Err_code = Rezult_code >> 8;                                     /* Obtain error code (upper 8 bits)                  */
          Data_r = (unsigned char) (Rezult_code & 0x00FF);                 /* Obtain read data (lower 8 bits)                   */
          if (Err_code != 0){                                              /* Check error code, and if an error occurs,         */
               PDR1 = 0x02;                                                /* set error code (read error = 2), and              */
               goto Exit_err;                                              /* exit with an error                                */
               }

          if (Data_w != Data_r){                                           /* Compare write data to read data,                  */
                                                                           /* and if an error occurs,                           */
               PDR1 = 0x03;                                                /* set error code (verify error = 3), and            */
               goto Exit_err;                                              /* exit with an error                                */
          }                                                                /*                                                   */
     }
/*============================================================================================================================*/
/*   Usage examples of page write (Master_page_write) and sequential read (Master_read_sequential)                            */
/*                                                                                                                            */
/*   Perform page write to EEPROM address (0x000 to 0x1ff) lower 8-bit value of address, perform sequential read, and verify.  */
/*   If an error occurs, output error code corresponding to process from process result output ports (P10, P11, P12).         */
/*============================================================================================================================*/
     Device_id = 0x00;                                                     /* Set EEPROM slave address code: 0 (A2, A1, A0)     */
     Address_start = 0x000;                                                /* Set EEPROM memory start address                   */
     Address_end = 0x200;                                                  /* Set EEPROM memory end address                     */

/*----------------------------------------------------------------------------------------------------------------------------*/
/*   Set data to be performed page writing to EEPROM data buffer                                                              */
/*----------------------------------------------------------------------------------------------------------------------------*/
     for (Address = Address_start; Address < Address_end; Address++){
          Data_w = (unsigned char)((Address + 1) & 0x00FF);                /* Obtain lower 8-bit value of address               */
          eeprom_buf[Address+1] = Data_w;                                  /* Store lower 8 bits of address                     */
                                                                           /* to EEPROM data buffer                             */
     }
```

```
/*------------------------------------------------------------------------------------------------------------------------------*/
/*   Perform page write from data buffer to EEPROM                                                                              */
/*------------------------------------------------------------------------------------------------------------------------------*/
     Length = 0x08;                                                        /* Set 1-page size to be written                     */
                                                                           /* (3687N, 3694N; 8 bytes/page)                      */
   /* Length = 0x20; */                                                    /* Set 1-page size to be written                     */
                                                                           /* (HN58X2432I, HN58X2464I; 32 bytes/page)           */

     for (Address = Address_start; Address < Address_end; Address+= Length){
                                                                           /* Perform writing all pages                         */
          Err_code = Master_page_write(Device_id, Address, Length, (eeprom_buf+Address));
                                                                           /* Perform writing 1 page to EEPROM                  */
          if (Err_code != 0){                                              /* Check error code, and if an error occurs,         */
               PDR1 = 0x04;                                                /* set error code (page write error = 4), and        */
               goto Exit_err;                                              /* exit with an error.                               */
          }
     }

/*------------------------------------------------------------------------------------------------------------------------------*/
/*   Clear EEPROM data buffer to 0                                                                                              */
/*------------------------------------------------------------------------------------------------------------------------------*/
     for (Address = Address_start; Address < Address_end; Address++){
          eeprom_buf[Address+1] = 0;
     }

/*------------------------------------------------------------------------------------------------------------------------------*/
/*   Perform sequential read from EEPROM to data buffer                                                                         */
/*------------------------------------------------------------------------------------------------------------------------------*/
     Length = Address_end - Address_start;                                 /* Set data size to be performed sequential reading  */

     Err_code = Master_read_sequential(Device_id, Address_start, Length, eeprom_buf);
                                                                           /* Perform sequential read for data size             */
                                                                           /* from EEPROM to data buffer                        */
     if (Err_code != 0){                                                   /* Check error code, and if an error occurs,         */
          PDR1 = 0x05;                                                     /* set error code (sequential read error = 5), and   */
          goto Exit_err;                                                   /* exit with an error.                               */
     }

     for (Address = Address_start; Address < Address_end; Address++){
          Data_w = (unsigned char)((Address + 1) & 0x00FF);                /* Obtain written data (lower 8 bits of address)     */
          Data_r = eeprom_buf[Address+1];                                  /* Obtain sequentially read data from data buffer    */

          if (Data_w != Data_r){                                           /* Compare written data to read data,                */
                                                                           /* and if an error occurs, set error code            */
               PDR1 = 0x06;                                                /* (sequential verify error = 6), and                */
               goto Exit_err;                                              /* exit with an error.                               */
          }
     }

     PDR1 = 0x00;                                                          /* Normal termination: set normal termination code   */
                                                                           /* to process result output port                     */

/*------------------------------------------------------------------------------------------------------------------------------*/
/*   Exit test                                                                                                                  */
/*------------------------------------------------------------------------------------------------------------------------------*/
     Exit_err:                                                             /* Exit test                                         */
          while(1)                                                         /* Stop program                                      */
          ;                                                                /*                                                   */
}

#pragma option
void abort(void){

}
```

[4] File name: iic2_eeprom.c (1)   Function: Sub processing (function group)

```
/*------------------------------------------------------------------------------------------------------------------------------*/
/*   H8/3694, H8/3687 on-chip I2C bus interface 2 (IIC2) Application note                                                       */
/*   Application examples for reading from/writing to serial EEPROM (sub program)                                               */
/*                                                                                                                              */
/*   1. Target EEPROM:   EEPROM which specifies memory address in two bytes                                                     */
/*   2. Operation confirmed product types:                                                                                      */
/*                 (1) H8/3687N,H8/3694N(512B) on-chip EEPROM (manufactured by RENESAS TECHNOLOGY)                              */
/*                 (2) HN58X2432I(4kB), HN58X2464I(8kB) EEPROM (manufactured by RENESAS TECHNOLOGY)                             */
/*      Ver 1.1     2003.5.9                                                                                                    */
/*                                                                                                                              */
/*        Copyright (C) RENESAS TECHNOLOGY 2003                                                                                 */
/*------------------------------------------------------------------------------------------------------------------------------*/
/******************************************************************************************************************************/
/*   Header definition                                                                                                          */
/******************************************************************************************************************************/
#include  <machine.h>
#include  "H8_3694_IIC2.h"                                   /* Definition file of IIC2 module register map     */

/******************************************************************************************************************************/
/*   EEPROM device code definition                                                                                              */
/******************************************************************************************************************************/
#define   DEVICE_CODE    0xa0                                            /* EEPROM device code: 1010(D3,D2,D1,D0)           */
#define   SLAVE_ADRS     0x00                                            /* Slave address code: 000(A2,A1,A0)               */
#define   RW_CODE_W      0x00                                            /* R/W code: 0 (data write)                        */
#define   RW_CODE_R      0x01                                            /* R/W code: 1 (data read)                         */
#define   DEVICE_ADDRESS_WORD_W□(DEVICE_CODE | SLAVE_ADRS | RW_CODE_W)       /* Slave address (for writing)                     */
#define   DEVICE_ADDRESS_WORD_R□(DEVICE_CODE | SLAVE_ADRS | RW_CODE_R)       /* Slave address (for reading)                     */

/******************************************************************************************************************************/
/*   Timeout limit count value definition                                                                                       */
/******************************************************************************************************************************/
#define   TIMEOUT_LIMIT_BBSY  1000                                       /* Timeout limit for bus busy check                */
#define   TIMEOUT_LIMIT_ACK   1000                                       /* Timeout limit for ACK check                     */
#define   TIMEOUT_LIMIT_TEND  1000                                       /* Timeout limit for TEND check                    */
#define   TIMEOUT_LIMIT_RDRF  1000                                       /* Timeout limit for RDRF check                    */
#define   TIMEOUT_LIMIT_STOP  1000                                       /* Timeout limit for STOP check                    */

/******************************************************************************************************************************/
/*   Error code definition                                                                                                      */
/******************************************************************************************************************************/
#define   TIMEOUT_ERR_BUS_BUSY     200                                   /* Error code: Bus busy timeout                    */
#define   TIMEOUT_ERR_ACK          201                                   /* Error code: Acknowledgement polling timeout     */
#define   TIMEOUT_ERR_TEND         202                                   /* Error code: Transmit end timeout                */
#define   TIMEOUT_ERR_RDRF         203                                   /* Error code: Receive end timeout                 */
#define   TIMEOUT_ERR_STOP         204                                   /* Error code: Stop condition issue timeout        */
#define   ERR_ACK                  205                                   /* Error code: Acknowledgement error               */
```

```
/*******************************************************************************************************************************/
/*   Port address definition                                                                                                   */
/*******************************************************************************************************************************/
#define PCR5   (*(volatile unsigned char *)0xFFE8)                          /* Port control register 5                         */
#define PDR5   (*(volatile unsigned char *)0xFFD8)                          /* Port data register 5                            */

/*******************************************************************************************************************************/
/*   Other constant definition                                                                                                 */
/*******************************************************************************************************************************/
#define   TM             100                                                /* Wait for the specified time wait counts         */

/*******************************************************************************************************************************/
/*   Prototype declaration for functions to be used                                                                            */
/*******************************************************************************************************************************/
unsigned char Check_bus_condition ( void );
unsigned char Master_address_set (unsigned char Device_id, unsigned short Addr );
unsigned short Master_read_byte_random (unsigned char Device_id, unsigned short Addr );
unsigned char Master_read_sequential ( unsigned char Device_id, unsigned short Addr, unsigned short Length, unsigned char
*buff_ptr );
unsigned char Master_byte_write (unsigned char Device_id, unsigned short Addr, unsigned char Data);
unsigned char Master_page_write (unsigned char Device_id, unsigned short Addr, unsigned char Length, unsigned char *buff_ptr);
unsigned char Send_byte_data ( unsigned char Byte_data );
void Send_start_condition ( void );
void Set_iic_bus_mode ( unsigned char Bus_mode_MLS, unsigned char Bus_mode_WAIT, unsigned char Bus_mode_BC210);
void Set_iic_mode ( unsigned char Iic_mode );
void Set_iic_rate ( unsigned char Iic_rate );
unsigned char Send_stop_condition ( void );
void Set_iic2_if_enable ( unsigned char Ice );
unsigned char Set_slave_read_mode ( unsigned char Device_id );
unsigned char Set_slave_write_mode ( unsigned char Device_id );
unsigned short Receive_byte_data ( void );
void Set_receive_mode ( unsigned char Ackbt_flag, unsigned char Rcvd_flag );
unsigned char Receive_byte_data_many ( unsigned short Length, unsigned char *buff_ptr );
unsigned short Receive_byte_data_fin ( void );

void Init_eeprom ( void );
void I2c_start ( void );
void I2c_stop ( void );
void I2c_set ( unsigned char Scl , unsigned char Sda );
void I2c_bytesend ( unsigned char Tx_data );
void I2c_bitsend ( unsigned char Tx_data , unsigned char Ckbit );
void I2c_send ( unsigned char Bit_data );
unsigned char I2c_ackck ( void );
void I2c_sda_in ( void );
void I2c_sda_out ( unsigned char Data );
void Wait_timer ( unsigned short Wait_cnt );
```

```
/*==============================================================================================================================*/
/*   Check IIC bus condition, and wait until bus becomes available                                                              */
/*                                                                                                                              */
/*   Entry:    None ; None                                                                                                      */
/*   Return:   Bbsy_flag (unsigned char); Bus condition flag                                                                    */
/*             0: Bus free                                                                                                      */
/*             1: Bus busy                                                                                                      */
/*             TIMEOUT_ERR_BUS_BUSY: Timeout                                                                                    */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned char  Check_bus_condition ( void ){

     unsigned char     Bbsy_flag;
     unsigned short  Time_out_count;

     Time_out_count = 0;                                                    /* Clear timeout counter                            */

     while (1) {                                                            /*                                                  */
          Bbsy_flag = IIC2.ICCR2.BIT.BBSY;                                  /* Read BBSY to obtain bus condition                */
          if (Bbsy_flag== 0){                                               /* Wait until bus condition becomes available       */
               break;
          }

          Time_out_count++;                                                 /* Check timeout                                    */
          if (Time_out_count > TIMEOUT_LIMIT_BBSY){                         /* If timeout occurs,                               */
               Bbsy_flag = (unsigned char) TIMEOUT_ERR_BUS_BUSY;            /* set bus condition code = TIMEOUT_ERR_BUS_BUSY    */
               break;                                                       /* and exit                                         */
          }                                                                 /*                                                  */
     }                                                                      /* Wait until bus condition becomes available       */

     return( Bbsy_flag );                                                   /* Return bus condition code                        */
}
```

```
/*==============================================================================================================================*/
/*   Transmit slave address and memory address to IIC device                                                                    */
/*                                                                                                                              */
/*   Entry:    Device_id (unsigned char) ; Slave address code: 0 to 7 (A2, A1, A0)                                              */
/*             Addr (unsigned short)      ; Memory address                                                                      */
/*   Return:   Err_code (unsigned char)  ; ; Error code                                                                         */
/*             0: Normal termination                                                                                            */
/*             2: Second byte transmission acknowledgement error (upper 8 bits of memory address)                               */
/*             3: Third byte transmission acknowledgement error (lower 8 bits of memory address)                                */
/*             TIMEOUT_ERR_BUS_BUSY: Bus busy timeout                                                                           */
/*             TIMEOUT_ERR_ACK: Acknowledgement polling timeout                                                                 */
/*        Note: Transmit end timeout error is returned as 2 to 3 of Err_code.                                                   */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned char  Master_address_set ( unsigned char Device_id, unsigned short Addr ){

     unsigned char  Bbsy_flag;
     unsigned char  Ack_flag;

     unsigned char  Addr_msb;
     unsigned char  Addr_lsb;
     unsigned char  Err_code;
     unsigned short Time_out_count;
     Bbsy_flag = Check_bus_condition();                                     /* Check bus condition                              */
                                                                            /* and wait until bus becomes available             */
     if( Bbsy_flag != 0 ){                                                  /* If bus busy timeout occurs,                      */
          Err_code = (unsigned char)TIMEOUT_ERR_BUS_BUSY;                   /* set error code = TIMEOUT_ERR_BUS_BUSY            */
          goto Exit_err;                                                    /* and exit                                         */
     }                                                                      /*                                                  */

     Time_out_count = 0;                                                    /* Clear timeout counter                            */

     while (1) {                                                            /*                                                  */
          Ack_flag = Set_slave_write_mode ( Device_id );                    /* Transmit slave address (write)                   */
                                                                            /* (Transmit first byte)                            */
          if ( Ack_flag == 0 ) {
               break;
          }

          Time_out_count++;                                                 /* Check acknowledgement timeout                    */
          if (Time_out_count > TIMEOUT_LIMIT_ACK){                          /* If timeout occurs,                               */
               Err_code = (unsigned char)TIMEOUT_ERR_ACK;                   /* set error code = TIMEOUT_ERR_ACK                 */
               goto Exit_err;                                               /* and exit                                         */
          }                                                                 /*                                                  */
     }                                                                      /* Perform acknowledgement polling                  */
                                                                            /* until acknowledgement is obtained                */

     while (1) {
          Addr_msb = (unsigned char)( Addr >> 8 );                          /* Obtain upper (MSB) address                       */
          Ack_flag = Send_byte_data( Addr_msb );                            /* Transmit upper (MSB) memory address              */
                                                                            /* (second byte) and check acknowledgements         */
          if (Ack_flag != 0) {                                              /* If the acknowledgement value is 1,               */
                                                                            /* acknowledgement error occurs                     */
               Err_code = 2;                                                /* Set error code (second byte: error)              */
               break;                                                       /* and exit                                         */
          }
          Addr_lsb = (unsigned char)( Addr & 0x00ff );                       /* Obtain lower (LSB) address                       */
          Ack_flag = Send_byte_data( Addr_lsb );                             /* Transmit lower (LSB) memory address (third byte) */
                                                                             /* and check acknowledgement                        */
          if (Ack_flag != 0) {                                               /* If the acknowledgement value is 1,               */
                                                                             /* acknowledgement error occurs                     */
               Err_code = 3;                                                 /* Set error code (third byte: error)               */
               break;                                                        /* and exit                                         */
          }

          Err_code = 0;                                                      /* Since this is normal termination,                */
                                                                             /* clear error code                                 */
          break;                                                             /* Exit                                             */
     }

Exit_err:
     return( Err_code );                                                     /* Return error code                                */
}
```

```
/*==============================================================================================================================*/
/*   Random address read (byte read) from IIC device                                                                            */
/*                                                                                                                              */
/*   Entry:    Device_id (unsigned char); Slave address code: 0 to 7 (A2, A1, A0)                                               */
/*             Addr (unsigned short)  ; Memory address                                                                          */
/*   Return:   Normal termination : Data_s (unsigned short)     ; Received data (lower 8 bits)□                                 */
/*             Error termination : Err_code_s (unsigned short)   ; Error code (upper 8 bits)                                    */
/*                                                                                                                              */
/*             Error code;                                                                                                      */
/*                  0: Normal termination                                                                                       */
/*                  2: Second byte transmission acknowledgement error (upper 8 bits of memory address)                          */
/*                  3: Third byte transmission acknowledgement error (lower 8 bits of memory address)                           */
/*                  ERR_ACK: Slave address (read) acknowledgement error                                                         */
/*                  TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error                                                                */
/*                  TIMEOUT_ERR_ACK: Acknowledgement polling timeout error                                                      */
/*                  TIMEOUT_ERR_RDRF: Receive end timeout error                                                                 */
/*                  TIMEOUT_ERR_STOP: Stop condition detection timeout error                                                    */
/*        Note: Transmit end timeout error is returned as 2 to 3 of Err_code.                                                   */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned short  Master_read_byte_random ( unsigned char Device_id, unsigned short Addr ){

     unsigned char  Ack_flag;
     unsigned char  Stop_flag;
     unsigned char  Data;
     unsigned short Data_s;
     unsigned char  Err_code;
     unsigned short Err_code_s;

     Err_code = Master_address_set ( Device_id, Addr );                      /* Transmit slave address and memroy address        */
                                                                            /* in write mode                                    */
     if (Err_code != 0){                                                    /* If an error occurs, abort processing,            */
                                                                            /* and issue stop condition                         */
          goto Exit_err;                                                    /* Perform error processing                         */
}                                                                           /*                                                  */

     Ack_flag = Set_slave_read_mode ( Device_id );                           /* Retransmit slave address in read mode            */
     if (Ack_flag != 0){                                                    /* If there is no acknowledgement,                  */
                                                                            /* abort processing, and issue stop condition       */
          Err_code = ERR_ACK  ;                                             /* Set acknowledgement error code                   */
          goto Exit_err;                                                    /* Perform error processing                         */
     }

     Set_iic_mode(2);                                                       /* Set master reception mode (MST = 1, TRS = 0)     */

     IIC2.ICSR.BIT.TEND = 0;                                                /* Clear TEND                                       */
     IIC2.ICSR.BIT.TDRE = 0;                                                /* Clear TDRE                                       */

     Set_receive_mode ( 1, 1 );                                             /* Set subsequent reception mode (ACKBT, RCVD).     */
                                                                            /* ACKBT=1 (because of only 1-byte reception,       */
                                                                            /* set the acknowledgement value to No (ACK = 1)    */
                                                                            /* as final data.)                                  */
                                                                            /* RCVD =1 (disable subsequent reception operation) */
     Data_s = Receive_byte_data ();                                          /* Read data (start reception operation)           */
     if ((Data_s & 0xff00) != 0){                                            /* If an error occurs                              */
                                                                             /* (when upper 8 bits are not 0),                  */
                                                                             /* abort processing and issue stop condition       */
          Err_code = (unsigned char)TIMEOUT_ERR_RDRF;                        /* Set receive end timeout error code              */
          goto Exit_err;                                                     /* Perform error processing                        */
     }                                                                       /*                                                 */

     Data_s = Receive_byte_data_fin (); □□                                   /* Read data (obtain final received data)          */

     Stop_flag = Send_stop_condition();                                      /* Issue stop condition (BBSY = 0, SCP = 0)        */
     if (Stop_flag != 1){                                                    /* If an error occurs, abort processing            */
                                                                             /* and issue stop condition                        */
          Err_code = (unsigned char)TIMEOUT_ERR_STOP;                        /* Set stop condition detection timeout error code */
          goto Exit_err_2;                                                   /* Perform error processing                        */
     }                                                                       /*                                                 */

     IIC2.ICCR1.BIT.RCVD = 0;                                                /* Clear RCVD(set subsequent reception operation as*/
                                                                             /* to be continuous)                               */
     Set_iic_mode(0);                                                        /* Set slave reception mode (MST = 0, TRS = 0)     */

     return ( Data_s );                                                      /* Normal termination; return received data        */
                                                                             /* (lower 8 bits: data,                            */
                                                                             /* upper 8 bits: error code = 0)                   */

Exit_err:
     Stop_flag = Send_stop_condition();                                      /* Issue stop condition (BBSY = 0, SCP = 0)        */
     if (Stop_flag != 1){                                                    /* If an error occurs, set error code              */
          Err_code = (unsigned char)TIMEOUT_ERR_STOP;                        /* Set stop condition detection timeout error code */
     }

Exit_err_2:
     IIC2.ICCR1.BIT.RCVD = 0;                                                /* Clear RCVD(set subsequent reception operation   */
                                                                             /* as to be continuous)                            */
     Set_iic_mode(0);                                                        /* Set slave reception mode (MST = 0, TRS = 0)     */

     Err_code_s = ((unsigned short) Err_code << 8);                          /* Shift error code to upper 8 bits                */
     return ( Err_code_s );                                                  /* Error termination; return error code            */
                                                                             /* (upper 8 bits)                                  */

}
```

```
/*=============================================================================================================================*/
/*   Sequential address read from IIC device                                                                                   */
/*                                                                                                                             */
/*   Entry:    Device_id      (unsigned char)    ; Slave address code: 0 to 7 (A2, A1, A0)                                     */
/*             Addr           (unsigned short)   ; Read start memory address                                                   */
/*             Length         (unsigned short)   ; Length of read bytes (more than 2 bytes)                                    */
/*             *buff_ptr      (unsigned char)    ; Read data storage buffer start address                                      */
/*   Return:   Err_code       (unsigned char)    ; Error code                                                                  */
/*             0: Normal termination                                                                                           */
/*             2: Second byte acknowledgement error (upper 8 bits of memory address)                                           */
/*             3: Third byte acknowledgement error (lower 8 bits of memory address)                                            */
/*             ERR_ACK: Slave address (read) acknowledgement error                                                             */
/*             TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error                                                                    */
/*             TIMEOUT_ERR_ACK: Acknowledgement polling timeout error                                                          */
/*             TIMEOUT_ERR_RDRF: Data receive end timeout error                                                                */
/*             TIMEOUT_ERR_STOP: Stop condition detection timeout error                                                        */
/*        Note: Transmit end timeout error is returned as 2 to 3 of Err_code.                                                  */
/*                                                                                                                             */
/*=============================================================================================================================*/
unsigned char  Master_read_sequential ( unsigned char Device_id, unsigned short Addr, unsigned short Length, unsigned char
*buff_ptr ){

     unsigned char  Ack_flag;
     unsigned char  Stop_flag;
     unsigned char  Err_code;

     Err_code = Master_address_set ( Device_id, Addr );                     /* Transmit slave address and memory address       */
                                                                            /* in write mode                                   */
     if (Err_code != 0){                                                    /* If an error occurs, abort processing,           */
                                                                            /* and issue stop condition                        */
          goto Exit_err;                                                    /* Perform error processing                        */
     }                                                                      /*                                                 */

     Ack_flag = Set_slave_read_mode ( Device_id );                          /* Retransmit slave address in read mode           */
     if (Ack_flag != 0){                                                    /* If there is no acknowledgement, abort processing*/
                                                                            /* and issue stop condition                        */
          Err_code = ERR_ACK  ;                                             /* Set acknowledgement error code                  */
          goto Exit_err;                                                    /* Perform error processing                        */
     }                                                                      /*                                                 */

     Set_iic_mode(2);                                                       /* Set master reception mode (MST = 1, TRS = 0)    */

     IIC2.ICSR.BIT.TEND = 0;                                                /* Clear TEND                                      */
     IIC2.ICSR.BIT.TDRE = 0;                                                /* Clear TDRE                                      */

     Set_receive_mode ( 0, 0 );                                             /* Set subsequent reception mode                   */
                                                                            /* (ACKBT = 0, RCVD = 0)                           */
                                                                            /* ACKBT = 0 (set acknowledgement (ACK = 0))       */
                                                                            /* RCVD = 0 (set subsequent reception operation    */
                                                                            /* as to be continuous)                            */
     Err_code = Receive_byte_data_many( Length, buff_ptr );                  /* Receive data with a size to be received         */
                                                                             /* and store it in buffer                          */
     if (Err_code != 0){                                                     /* If an error occurs, abort processing            */
                                                                             /* and issue stop condition                        */
          Err_code = (unsigned char)TIMEOUT_ERR_RDRF;                        /* Set receive end timeout error code              */
     }                                                                       /*                                                 */
Exit_err:
     Stop_flag = Send_stop_condition();                                      /* Issue stop condition (BBSY = 0, SCP = 0)        */
     if (Stop_flag != 1){                                                    /* If an error occurs,                             */
          Err_code = (unsigned char)TIMEOUT_ERR_STOP;                        /* set stop condition detection timeout error code */
     }                                                                       /*                                                 */

     IIC2.ICCR1.BIT.RCVD = 0;                                                /* Clear RCVD (set subsequent reception operation  */
                                                                             /* as to be continuous)                            */
     Set_iic_mode(0);                                                        /* MST = 0, TRS = 0 (set slave reception mode)     */

     return ( Err_code );                                                    /* Return error code                               */
                                                                             /* (in normal operation, error code = 0)           */
}
```

```
/*==============================================================================================================================*/
/*   Byte data write to IIC device (byte write)                                                                                 */
/*                                                                                                                              */
/*   Entry:    Device_id (unsigned char)   ; Slave address code: 0 to 7 (A2,A1,A0)                                              */
/*             Addr      (unsigned short)  ; Memory address                                                                     */
/*             Data      (unsigned char)   ; Write data                                                                         */
/*   Return:   Err_code  (unsigned char)   ; Error code                                                                         */
/*             0: Normal termination                                                                                            */
/*             2: Second byte acknowledgement error (upper 8 bits of memory address)                                            */
/*             3: Third byte acknowledgement error (lower 8 bits of memory address)                                             */
/*             4: Fourth byte acknowledgement error (data)                                                                      */
/*             TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error                                                                     */
/*             TIMEOUT_ERR_ACK: Acknowledgement polling timeout error                                                           */
/*             TIMEOUT_ERR_STOP: Stop condition detection timeout error                                                         */
/*        Note: Transmit end timeout error is returned as 2 to 3 of Err_code.                                                   */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned char  Master_byte_write ( unsigned char Device_id, unsigned short Addr, unsigned char Data ){

     unsigned char  Ack_flag;
     unsigned char  Err_code;
     unsigned char  Stop_flag;

     while (1) {
          Err_code = Master_address_set( Device_id, Addr );                  /* Transmit slave address                          */
                                                                             /* and memory address in write mode                */
          if (Err_code != 0){                                                /* If an error occurs, abort processing            */
                                                                             /* and issue stop condition                        */
               break;                                                        /*                                                 */
          }                                                                  /*                                                 */

          Ack_flag = Send_byte_data( Data );                                 /* Transmit data                                   */
          if (Ack_flag != 0) {                                               /* If the acknowledgement value is 1,              */
                                                                             /* an acknowledgement error occurs                 */
               Err_code = 4;                                                 /* Set error code (fourth byte: error) and         */
               break;                                                        /* issue stop condition                            */
          }

          Err_code = 0;                                                      /* Because of normal termination, clear error code, */
          break;                                                             /* and issue stop condition                        */
     }

     Stop_flag = Send_stop_condition();                                      /* Issue stop condition (BBSY=0, SCP=0)            */
     if (Stop_flag != 1){                                                    /* If an error occurs,                             */
          Err_code = (unsigned char)TIMEOUT_ERR_STOP;                        /* set stop condition detection timeout error code */
     }                                                                       /*                                                 */

     Set_iic_mode( 0 );                                                      /* MST=0, TRS=0 (set slave reception mode)         */

     IIC2.ICSR.BIT.TEND = 0;                                                 /* Clear TEND                                      */
     IIC2.ICSR.BIT.TDRE = 0;                                                 /* Clear TDRE                                      */

     return( Err_code );                                                     /* Return error code                               */

}
```

```
/*==============================================================================================================================*/
/*   Page data write to IIC device (page write)                                                                                 */
/*                                                                                                                              */
/*   Entry:    Device_id (unsigned char)   ; Slave address code: 0 to 7 (A2,A1,A0)                                              */
/*             Addr      (unsigned short)  ; write start memory address                                                         */
/*             Length    (unsigned char)   ; Write byte length (within 1 page)                                                  */
/*             buff_ptr  (unsigned char)   ; Written data storage buffer start address                                          */
/*   Return:   Err_code  (unsigned char)   ; Error code                                                                         */
/*             0: Normal termination                                                                                            */
/*             2: Second byte acknowledgement error (upper 8 bits of address)                                                   */
/*             2: Second byte acknowledgement error (upper 8 bits of address)                                                   */
/*             4 to 131 (transmitted byte length + 3):                                                                          */
/*                  Transmission acknowledgement error in the bytes following fourth byte (data)                                */
/*             TIMEOUT_ERR_BUS_BUSY: Bus busy timeout error                                                                     */
/*             TIMEOUT_ERR_ACK: Acknowledgement polling timeout error                                                           */
/*             TIMEOUT_ERR_STOP: Stop condition detection timeout error                                                         */
/*        Note: Transmit end timeout error is retuned as 2 to 131 of Err_code.                                                  */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned char  Master_page_write ( unsigned char Device_id, unsigned short Addr, unsigned char Length, unsigned char *buff_ptr ){

     unsigned char  Ack_flag;
     unsigned char  Err_code;
     unsigned char  Data;
     unsigned char  Cnt;
     unsigned char  Stop_flag;

     Err_code = Master_address_set ( Device_id, Addr );                      /* Transmit slave address and memory address       */
                                                                             /* in write mode                                   */
     if (Err_code != 0){                                                     /* If an error occurs, abort processing            */
                                                                             /* and issue stop condition                        */
          goto Exit_err;                                                     /* Error termination                               */
     }

     buff_ptr++;                                                             /* Increment written data storage address          */
                                                                             /* by 1 to set head of data by 1                   */

     for ( Cnt = 1; Cnt <= Length; Cnt++ ){                                  /* Transmit data equaling the size                 */
                                                                             /* of the written byte length value                */
          Data = *buff_ptr;                                                  /* Obtain written data                             */
                                                                             /* from written data storage address               */
          buff_ptr++;                                                        /* Increment written data storage address          */

          Ack_flag = Send_byte_data( Data );                                 /* Transmit data                                   */
          if (Ack_flag != 0) {                                               /* If the acknowledgement value is not 0,          */
                                                                             /* acknowledgement error occurs                    */
               Err_code = 3 + Cnt;                                           /* Set error code (following fourth byte: error)   */
               goto Exit_err;                                                /* Error termination;                              */
                                                                             /*abort processing and issue stop condition        */
          }                                                                  /* □Error code is 3 + transmitted data length      */
     }

     Err_code = 0;                                                           /* Because of normal termination, clear error code */

Exit_err:
     Stop_flag = Send_stop_condition();                                      /* Issue stop condition (BBSY=0, SCP=0)            */
     if (Stop_flag != 1){                                                    /* If an error occurs,                             */
          Err_code = (unsigned char)TIMEOUT_ERR_STOP;                        /* set stop condition detection timeout error code */
     }                                                                       /*                                                 */

     Set_iic_mode( 0 );                                                      /* MST=0, TRS=0 (set slave reception mode)         */

     IIC2.ICSR.BIT.TEND = 0;                                                 /* Clear TEND                                      */
     IIC2.ICSR.BIT.TDRE = 0;                                                 /* Clear TDRE                                      */

     return( Err_code );                                                     /* Return error code                               */
}
```

```
/*==============================================================================================================================*/
/*   Transmit data (1 byte) to IIC device                                                                                       */
/*                                                                                                                              */
/*   Entry:    Byte_data      (unsigned char)  ; Transmitted data                                                               */
/*   Return:   Ackbr_flag     (unsigned char)  ; Acknowledgement flag                                                           */
/*             0: Normal termination                                                                                            */
/*             1: Abnormal termination                                                                                          */
/*             TIMEOUT_ERR_TEND: Transmit end timeout                                                                           */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned char  Send_byte_data ( unsigned char Byte_data ){

     unsigned char  Tend_flag;
     unsigned char  Ackbr_flag;
     unsigned short Time_out_count;

     IIC2.ICDRT = Byte_data;                                                  /* Transmit data                                  */

     Time_out_count = 0;                                                      /* Clear timeout count                            */

     while(1) {                                                               /*                                                */
          Tend_flag = IIC2.ICSR.BIT.TEND;                                     /* Check TEND and                                 */
          if ( Tend_flag == 1 ){                                              /* wait for transmission completion               */
               break;                                                         /*                                                */
          }                                                                   /*                                                */

          Time_out_count++;                                                   /* Check transmit end timeout                     */
          if (Time_out_count > TIMEOUT_LIMIT_TEND){□                          /* If timeout occurs,                             */
               Ackbr_flag = (unsigned char)TIMEOUT_ERR_TEND;                  /* set Ackbr_flag  = TIMEOUT_ERR_TEND and         */
               goto Exit_err;                                                 /* exit                                           */
          }                                                                   /*                                                */
     }

     Ackbr_flag = IIC2.ICIER.BIT.ACKBR;                                       /* When transmission completed,                   */
                                                                              /* obtain acknowledgement                         */

Exit_err:
     return( Ackbr_flag );                                                    /* Return acknowledgement                         */
}


/*==============================================================================================================================*/
/*   Issue start condition to IIC device                                                                                        */
/*                                                                                                                              */
/*   Entry:    None ; None                                                                                                      */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  Send_start_condition ( void ){

     IIC2.ICCR2.BYTE = 0xBD;                                                  /* Issue start condition (BBSY=1, SCP=0)          */
}
```

```
/*=============================================================================================================================*/
/*   Issue stop condition to IIC device                                                                                        */
/*                                                                                                                             */
/*   Entry:    None ; None                                                                                                     */
/*   Return:   Stop_flag (unsigned char); Stop condition detection flag                                                        */
/*             1: Stop condition detected                                                                                      */
/*             TIMEOUT_ERR_STOP: Stop condition detection timeout (stop condition not detected)                                */
/*                                                                                                                             */
/*=============================================================================================================================*/
unsigned char  Send_stop_condition ( void ){

     unsigned char   Stop_flag;
     unsigned short  Time_out_count;

     IIC2.ICSR.BIT.STOP = 0;                                                /* Clear STOP                                    */
     IIC2.ICCR2.BYTE = 0x3D;                                                /* Issue stop condition (BBSY=0, SCP=0)          */

     Time_out_count = 0;                                                    /* Clear timeout counter                         */

     while (1) {                                                            /*                                               */
          Stop_flag = IIC2.ICSR.BIT.STOP;                                   /* Check stop condition detection flag, and      */
          if ( Stop_flag == 1 ){                                            /* wait until stop condition is met              */
               break;                                                       /*                                               */
          }                                                                 /*                                               */

          Time_out_count++;                                                 /* Check stop condition detection timeout        */
          if (Time_out_count > TIMEOUT_LIMIT_STOP){                         /* If timeout occurs,                            */
               Stop_flag = (unsigned char)TIMEOUT_ERR_STOP;                 /* set Stop_flag = TIMEOUT_ERR_STOP, and         */
               goto Exit_err;                                               /* exit                                          */
          }                                                                 /*                                               */
     }
Exit_err:
     return ( Stop_flag );                                                  /* Return stop condition detection flag          */
}

/*=============================================================================================================================*/
/*   Retransmit slave address (read) to IIC device and set read mode                                                           */
/*                                                                                                                             */
/*   Entry:    Device_id  (unsigned char)  ; Slave address code: 0 to 7 (A2,A1,A0)                                             */
/*   Return:   Ack_flag     (unsigned char)  ; Reception acknowledgement flag                                                  */
/*             0: Normal, 1: Error                                                                                             */
/*                                                                                                                             */
/*=============================================================================================================================*/
unsigned char  Set_slave_read_mode ( unsigned char Device_id ){

     unsigned char   Slave_address;
     unsigned char   Ack_flag;

     Set_iic_mode(3);                                                       /* Set master transmission mode (MST=1, TRS=1)   */
     Send_start_condition();                                                /* Issue start condition (BBSY=1, SCP=0)         */

     Slave_address = (unsigned char)DEVICE_ADDRESS_WORD_R | (Device_id << 1);
                                                                            /* Generate slave address (read)                 */
     Ack_flag = Send_byte_data( Slave_address );                            /* Transmit slave address                        */
                                                                            /* and obtain transmission acknowledgement       */

     return( Ack_flag );                                                    /* Return reception acknowledgement              */
}
```

```
/*==============================================================================================================================*/
/*   Retransmit slave address (read) to IIC device, and set write mode                                                          */
/*                                                                                                                              */
/*   Entry:    Device_id    (unsigned char)  ; Slave address code: 0 to 7 (A2,A1,A0)                                            */
/*   Return:   Ack_flag     (unsigned char)  ; Receive acknowledgement flag                                                     */
/*             0: Normal, 1: Error                                                                                              */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned char  Set_slave_write_mode ( unsigned char Device_id ){

     unsigned char  Slave_address;
     unsigned char  Ack_flag;

     Set_iic_mode(3);                                                       /* Set master transmission mode (MST=1, TRS=1)      */
     Send_start_condition();                                                /* Issue start condition (BBSY=1, SCP=0)            */

     Slave_address = (unsigned char)DEVICE_ADDRESS_WORD_W | (Device_id << 1);
                                                                            /* Generate slave address (write)                   */
     Ack_flag = Send_byte_data( Slave_address );                            /* Transmit slave address                           */

     return( Ack_flag );                                                    /* Return reception acknowledgement                 */
}

/*==============================================================================================================================*/
/*   Receive data from IIC device (start reception operation simultaneously with data read)                                     */
/*   Start reception operation and perform RDRF check                                                                           */
/*                                                                                                                              */
/*   Entry:    None ; None                                                                                                      */
/*   Return:   Err_code_s (unsigned short) ; Error code (upper 8 bits)                                                          */
/*             Data_s     (unsigned short) ; Received data (lower 8 bits)                                                       */
/*                                                                                                                              */
/*             Error code;                                                                                                      */
/*                  0: Normal termination                                                                                       */
/*                  TIMEOUT_ERR_RDRF:Receive end timeout error                                                                  */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned short  Receive_byte_data ( void ){

     unsigned char   Rdrf_flag;
     unsigned char   Data;
     unsigned short  Data_s;
     unsigned short  Err_code_s;
     unsigned short  Time_out_count;

     Time_out_count = 0;                                                    /* Clear timeout counter                            */

     Data = IIC2.ICDRR;                                                     /* Read data                                        */
                                                                            /* (start subsequent reception when ICDRR is read)  */

     while(1) {                                                             /*                                                  */
          Rdrf_flag = IIC2.ICSR.BIT.RDRF;                                   /*                                                  */
          if ( Rdrf_flag == 1 ){                                            /* Wait for all data                                */
               break;                                                       /*                                                  */
          }                                                                 /*                                                  */

     Time_out_count++;                                                      /* Check receive end timeout                        */
     if (Time_out_count > TIMEOUT_LIMIT_RDRF){                              /* If timeout occurs,                               */
               Err_code_s = (unsigned short) TIMEOUT_ERR_RDRF;              /* set Err_code = TIMEOUT_ERR_RDRF and              */
               goto Exit_err;                                               /* exit                                             */
          }                                                                 /*                                                  */
     }                                                                      /* Normal termination                               */
     Data_s = (unsigned short) Data;                                          /* Extend data to 2 bytes                          */
     return ( Data_s );                                                       /* Return data                                     */

Exit_err:                                                                     /* Error termination                               */
     Err_code_s = (Err_code_s << 8);                                          /* Shift error code to upper 8 bits                */
     return ( Err_code_s );                                                   /* Return error code                               */
                                                                              /* (set upper 8 bits as error code)                */}
/*==============================================================================================================================*/
/*   Receive data from IIC device (read final data) Since subsequent reception operation is not performed                       */
/*   because of it is final data, RDRF check is not performed                                                                   */
/*                                                                                                                              */
/*   Entry:    None ; None                                                                                                      */
/*   Return:   Data_s (unsigned short); Received data (lower 8 bits)                                                            */
/*                                                         (upper 8 bits are 0)
     */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned short  Receive_byte_data_fin ( void ){

     unsigned char   Data;
     unsigned short  Data_s;

     Data = IIC2.ICDRR;                                                       /* Read final data                                 */
     Data_s = (unsigned short) Data;                                          /* Extend data to 2 bytes                          */

     return( Data_s );                                                        /* Return read data                                */
}

/*==============================================================================================================================*/
/*   Set ACKBT and RCVD flags for receiving data                                                                                */
/*                                                                                                                              */
/*   Entry:  Ackbt_flag (unsigned char) ; Flag to be set as ACKBT (0,1)                                                         */
/*           Rcvd_flag  (unsigned char) ; Flag to be set as RCVD (0,1)                                                          */
/*   Entry:    None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  Set_receive_mode ( unsigned char Ackbt_flag, unsigned char Rcvd_flag ){

     IIC2.ICIER.BIT.ACKBT = Ackbt_flag;                                       /* Set ACKBT                                       */
     IIC2.ICCR1.BIT.RCVD = Rcvd_flag;                                         /* Set RCVD                                        */

}
```

```
/*============================================================================================================================*/
/*   Receive byte data from IIC device sequentially(for read byte length)                                                     */
/*                                                                                                                            */
/*   Entry:    Length    (unsigned short) ; Read byte length (more than 2 bytes)                                              */
/*             *buff_ptr (unsigned char) ; Read data storage buffer start address                                             */
/*             Since dummy data upon starting reception is stored in the start address (buff_ptr),                           */
/*             valid data is stored with a size specified by Length from (buff_ptr + 1)                                       */
/*   Return:   Err_code (unsigned char) ; Error code                                                                          */
/*                  0: Normal termination                                                                                     */
/*                  TIMEOUT_ERR_RDRF:Receive end timeout error                                                                */
/*                                                                                                                            */
/*============================================================================================================================*/
unsigned char  Receive_byte_data_many ( unsigned short Length, unsigned char *buff_ptr ){

     unsigned char  Err_code;
     unsigned short Data_s;
     unsigned char  Data;
     Length++;                                                              /* Since the data first received is dummy data      */
                                                                            /* (invalid), actual received size must be          */
                                                                            /* incremented by 1.                                */

     while( Length > 1 ) {                                                  /* Receive data equalling the size                  */
                                                                            /* of data byte length value to be received         */
          if ( Length == 2 ){                                               /* Check whether the received data is one byte      */
                                                                            /* before the final data                            */
               Set_receive_mode ( 1, 1 );                                   /* Set subsequent reception mode (ACBT=1,RCVD=1)    */
                                                                            /* Set ACKBT (for receiving final data,             */
                                                                            /* set the acknowledgement value to No (ACK=1))     */
                                                                            /* Set RCVD                                         */
                                                                            /* (disable subsequent reception operation)         */
          }

          Data_s = Receive_byte_data ();                                    /* Read received data (start reception operation)   */
          if ((Data_s & 0xff00) != 0){                                      /* If an error occurs, abort processing,            */
                                                                            /* and issue stop condition                         */
               Err_code = (unsigned char) TIMEOUT_ERR_RDRF;                 /* Set receive end timeout error code               */
               goto Exit_err;                                               /* Perform error processing                         */
          }

          Data = (unsigned char)(Data_s & 0x00ff);                          /* Obtain lower 8 bits of received data             */
          *buff_ptr = Data;                                                 /* Store received data in buffer                    */

          buff_ptr++;                                                       /* Increment storage buffer address                 */
          Length--;                                                         /* Decrement received data length                   */
     }

     Data_s = Receive_byte_data_fin ();                                     /* Read received data (obtain final received data)  */

     Data = (unsigned char)(Data_s & 0x00ff);                               /* Obtain lower 8 bits of received data             */
     *buff_ptr = Data;                                                      /* Store received data in buffer                    */

     Err_code = 0;                                                          /* Normal termination; set error code as 0          */

Exit_err:
     return ( Err_code );                                                   /* Return error code                                */
}

/*============================================================================================================================*/
/*   Initialize IIC2 I/F                                                                                                      */
/*                                                                                                                            */
/*   Entry:  None   ; None                                                                                                    */
/*   Return: None   ; None                                                                                                    */
/*                                                                                                                            */
/*============================================================================================================================*/
void  Init_iic2 ( void ){

     Set_iic_rate (1);                                                      /* Set IIC transfer rate as Φ/40 (Φ = 16MHz, 400 kHz),  */
     Set_iic_bus_mode (0,0,0);                                              /* IIC bus mode as MSB first, no wait insertion,    */
                                                                            /* and,bit length:9 bits                            */
     Set_iic2_if_enable (1);                                                /* Set IIC2 I/F module to a transfer operation      */
                                                                            /* enabled state (SCL/SDA: bus driving status)      */
}
```

```
/*==============================================================================================================================*/
/*   Set master device bus mode (MLS, WAIT, BC210)                                                                              */
/*                                                                                                                              */
/*   Entry:    Bus_mode_MLS (unsigned char) ; MSB first/LSB first                                                               */
/*                  0(MLS=0): MSB first (When IIC is used, set to 0)                                                            */
/*                  1(MLS=1): LSB first                                                                                         */
/*             Bus_mode_WAIT (unsigned char) ; Wait insertion specification                                                     */
/*                  0(WAIT=0): Not insert wait                                                                                  */
/*                  1(WAIT=1): Insert wait                                                                                      */
/*             Bus_mode_BC210 (unsigned char) ; Transfer data bit length specification                                          */
/*                  0(BC210=000): 9-bit mode                                                                                    */
/*                  1(BC210=001): 2-bit mode                                                                                    */
/*                  2(BC210=010): 3-bit mode                                                                                    */
/*                  3(BC210=011): 4-bit mode                                                                                    */
/*                  4(BC210=100): 5-bit mode                                                                                    */
/*                  5(BC210=101): 6-bit mode                                                                                    */
/*                  6(BC210=110): 7-bit mode                                                                                    */
/*                  7(BC210=111): 8-bit mode                                                                                    */
/*   Return:   None ; None                                                                                                      */
/*        Note: Call as SCL = 1, except for setting the transfer data bit length to 0.                                          */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  Set_iic_bus_mode ( unsigned char Bus_mode_MLS, unsigned char Bus_mode_WAIT, unsigned char Bus_mode_BC210){

     unsigned char bus_mode;

     Bus_mode_MLS &= 0x01;                                                  /* t MSB first/LSB first to MLS                     */
     if (Bus_mode_MLS == 1){                                                /*                                                  */
          Bus_mode_MLS = 0x80;                                              /* MLS = bit7                                       */
     }                                                                      /*                                                  */

     Bus_mode_WAIT &= 0x01;                                                 /* Set wait insertion specification as WAIT         */
          if (Bus_mode_WAIT == 1){                                          /*                                                  */
          Bus_mode_WAIT = 0x40;                                             /* WAIT = bit6                                      */
          }                                                                 /*                                                  */

     Bus_mode_BC210 &= 0x03;                                                /* Set transfer data bit length as BC210            */

     bus_mode = 0;                                                          /* Generate bus mode setting value                  */
     bus_mode = (Bus_mode_MLS | Bus_mode_WAIT | Bus_mode_BC210);            /*                                                  */

     IIC2.ICMR.BYTE =  bus_mode;                                            /* Set bus mode to register                         */
     IIC2.ICMR.BYTE = (bus_mode | 0x08);                                    /* Set write protect for BC210 (BCWP=1)             */
}
```

```
/*==============================================================================================================================*/
/*   Set master device transmission/reception mode (MST, TRS)                                                                   */
/*                                                                                                                              */
/*   Entry: Iic_mode     (unsigned char) ; Transmission/reception mode                                                          */
/*                  0(MST=0, TRS=0): Slave reception mode                                                                       */
/*                  1(MST=0, TRS=1): Slave transmission mode                                                                    */
/*                  2(MST=1, TRS=0): Master reception mode                                                                      */
/*                  3(MST=1, TRS=1): Master transmission mode                                                                   */
/*   Return: None   ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  Set_iic_mode ( unsigned char Iic_mode ){

     Iic_mode &= 0x03;                                                      /* Mask argument range between 0 and 3              */

     switch ( Iic_mode ){                                                   /* Generate data to be set                          */
          case 0:   Iic_mode = 0;                                           /* MST=0, TRS=0                                     */
                    break;
          case 1:   Iic_mode = 0x10;                                        /* MST=0, TRS=1                                     */
                    break;
          case 2:   Iic_mode = 0x20;                                        /* MST=1, TRS=0                                     */
                    break;
          case 3:   Iic_mode = 0x30;                                        /* MST=1, TRS=1                                     */
                    break;
          default: Iic_mode = 0;                                            /* MST=0, TRS=0                                     */
     }

     IIC2.ICCR1.BYTE = ((IIC2.ICCR1.BYTE & 0xcf) | Iic_mode);               /* Set transmission/reception mode to register      */
}
/*==============================================================================================================================*/
/*   Set master device transfer rate (CKS3, 2, 1, 0)                                                                            */
/*                                                                                                                              */
/*   Entry: Iic_rate (unsigned char) ; Transfer rate                                                                            */
/*             0(CKS3210=0000): clock;Φ/28 mode                                                                                 */
/*             1(CKS3210=0001): clock;Φ/40 mode                                                                                 */
/*             2(CKS3210=0010): clock;Φ/48 mode                                                                                 */
/*             3(CKS3210=0011): clock;Φ/64 mode                                                                                 */
/*             4(CKS3210=0100): clock;Φ/80 mode                                                                                 */
/*             5(CKS3210=0101): clock;Φ/100 mode                                                                                */
/*             6(CKS3210=0110): clock;Φ/112 mode                                                                                */
/*             7(CKS3210=0111): clock;Φ/128 mode                                                                                */
/*             8(CKS3210=1000): clock;Φ/56 mode                                                                                 */
/*             9(CKS3210=1001): clock;Φ/80 mode                                                                                 */
/*             10(CKS3210=1010): clock;Φ/96 mode                                                                                */
/*             11(CKS3210=1011): clock;Φ/128 mode                                                                               */
/*             12(CKS3210=1100): clock;Φ/160 mode                                                                               */
/*             13(CKS3210=1101): clock;Φ/200 mode                                                                               */
/*             14(CKS3210=1110): clock;Φ/224 mode                                                                               */
/*             15(CKS3210=1111): clock;Φ/256 mode                                                                               */
/*   Return: None   ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  Set_iic_rate ( unsigned char Iic_rate ){

     Iic_rate &= 0x0f;                                                      /* et master device transfer rate as CKS3210        */
     IIC2.ICCR1.BYTE = ((IIC2.ICCR1.BYTE & 0xf0)| Iic_rate);                /* Set transfer rate to register                    */
}
```

```
/*=============================================================================================================================*/
/*   Set IIC2 I/F module operation status (ICE)                                                                                */
/*                                                                                                                             */
/*   Entry:  Ice (unsigned char) ; Operation status                                                                            */
/*                  0 (ICE=0): Halt state (SCL/SDA: port function)                                                             */
/*                  1 (ICE=1): Transfer-operation-enabled state (SCL/SDA: bus driving status)                                  */
/*   Return:   None ; None                                                                                                     */
/*                                                                                                                             */
/*=============================================================================================================================*/
void  Set_iic2_if_enable ( unsigned char Ice ){

     Ice &= 0x01;

     if ( Ice == 1 ){                                                       /* Check Ice, and                                  */
          Ice = 0x80;                                                       /* 1: Set ICE (7th bit)                            */
     }else{
          Ice = 0x00;                                                       /* 0: Reset ICE (7th bit                           */
     }

     IIC2.ICCR1.BYTE = ((IIC2.ICCR1.BYTE & 0x7f) | Ice);                    /* Set operation status to register                */
}

/*=============================================================================================================================*/
/*   Initialize EEPROM bus condition                                                                                           */
/*                                                                                                                             */
/*   Issue start condition using port, transmit dummy slave address (0xff), and issue a stop condition                         */
/*   to forcibly initialize the EEPROM bus condition                                                                           */
/*   (This initialization process is used for forcibly making EEPROM SDA bus input state                                       */
/*   when reception processing from the master device cannot be performed since EEPROM                                         */
/*   SDA bus still remains in output status due to communication termination during receiving data from EEPROM)                */
/*   Entry:     None     ; None                                                                                                */
/*   Return:  None  ; None                                                                                                     */
/*                                                                                                                             */
/*=============================================================================================================================*/
void  Init_eeprom ( void ){

     unsigned char Ack_flg;

     PDR5 = 0xFF;                                                           /* Set P57 (SCL), P56 (SDA) as output and          */
     PCR5 = 0xFF;                                                           /* output P57 (SCL) = 1, P56 (SDA) = 1             */

     Set_iic2_if_enable ( 0 );                                              /* Set IIC2 I/F module in a halt state             */
                                                                            /* (SCL/SDApin: port function)                     */

     I2c_start ();                                                          /* Issue start condition                           */
     Wait_timer (TM);                                                       /* Wait tiemr                                      */

     I2c_bytesend ( 0xff );                                                 /* Transmit dummy slave address, and               */
     Ack_flg = I2c_ackck();                                                 /* then obtain acknowledgement                     */

     I2c_stop ();                                                           /* Issue stop condition                            */
     Wait_timer (TM);                                                       /* Wait for the specified time                     */
}
```

```
/*==============================================================================================================================*/
/*   Issue start condition to IIC device (through port processing)                                                              */
/*                                                                                                                              */
/*   Entry:    None ; None                                                                                                      */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_start ( void ){

     I2c_sda_out( 0xC0 );                                                    /* Set P57 (SCL), P56 (SDA) as output,             */
                                                                             /* and output P57 (SCL) = 1, P56 (SDA) = 1         */

     I2c_set (1,1);                                                          /* Set SCL = 1, SDA = 1 and output from port       */
     I2c_set (1,0);                                                          /* Set SCL = 1, SDA = 0 and output from port       */
     I2c_set (0,0);                                                          /* Set SCL = 0, SDA = 0 and output from port       */
}

/*==============================================================================================================================*/
/*   Issue stop condition to IIC device(through port processing)                                                                */
/*                                                                                                                              */
/*   Entry:    None ; None                                                                                                      */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_stop ( void ){

     I2c_sda_out( 0x00 );                                                    /* Set P57 (SCL), P56 (SDA) as output,             */
                                                                             /* and output P57 (SCL) = 0, P56 (SDA) = 0         */

     I2c_set (0,0);                                                          /* Set SCL = 0, SDA = 0 and output from port       */
     I2c_set (1,0);                                                          /* Set SCL = 1, SDA = 0 and output from port       */
     I2c_set (1,1);                                                          /* Set SCL = 1, SDA = 1 and output from port       */
}

/*==============================================================================================================================*/
/*   Output SCL, SDA to IIC device (through port processing)                                                                    */
/*                                                                                                                              */
/*   Entry:    Scl (unsigned char) ; Clock (SCL) output value                                                                   */
/*                  0: SCL(P57) = 0, 1: SCL(P57) = 1                                                                            */
/*             Sda  (unsigned char) ; Data (SDA) output value                                                                   */
/*                  0: SDA(P56) = 0, 1: SDA(P56) = 1                                                                            */
/*   Return: None   ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_set ( unsigned char Scl , unsigned char Sda ){

     unsigned char   Data;

     Data = 0;                                                               /* Initialize data to output                       */
                                                                             /* to P57 (SCL), P56 (SDA)                         */
     if ( Scl == 1 ){                                                        /* If SCL = 1, set P57 = 1                         */
          Data = 0x80;                                                       /*                                                 */
     }                                                                       /*                                                 */

     if ( Sda == 1 ){                                                        /* If SDA = 1, set P56 = 1                         */
          Data |= 0x40;                                                      /*                                                 */
     }                                                                       /*                                                 */
     PDR5 = ((PDR5 & 0x3F) | Data);                                          /* Output data to P57 (SCL), P56 (SDA)             */
}
```

```
/*==============================================================================================================================*/
/*   Transmit byte data to IIC device (through port processing)                                                                 */
/*                                                                                                                              */
/*   Entry: Tx_data (unsigned char) ; Byte data to be transmitted                                                               */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_bytesend ( unsigned char Tx_data ){

     unsigned char   Ckbit;

     I2c_sda_out( 0x00 );                                                   /* Set P57 (SCL), P56 (SDA) as output,              */
                                                                            /* and output P57 (SCL) = 0, P56 (SDA) = 0          */

     for (Ckbit = 0x80; Ckbit > 0; Ckbit >>= 1){                            /* Transmit 8-bit data                              */
          I2c_bitsend ( Tx_data , Ckbit );                                  /* from MSB sequentially                            */
     }                                                                      /*                                                  */
}

/*==============================================================================================================================*/
/*   Transmit bit data to IIC device (through port processing)                                                                  */
/*                                                                                                                              */
/*   Entry:    Tx_data (unsigned char) ; Byte data to be transmitted                                                            */
/*             Ckbit   (unsigned char) ; Bit position of bit data to be transmitted (value setting 1 to bit position)           */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_bitsend ( unsigned char Tx_data , unsigned char Ckbit ){

     unsigned char    Bit_data;

     Bit_data = ( Tx_data & Ckbit );                                        /* Obtain bit information                           */
                                                                            /* from a bit position to be transmitted            */
     if ( Bit_data != 0 ){                                                  /* Check bit information, and generate bit data     */
          Bit_data = 1;                                                     /* Bit data = 1,                                    */
     }                                                                      /*                                                  */
     else {                                                                 /*                                                  */
          Bit_data = 0;                                                     /* Bit data  = 0,                                   */
     }                                                                      /*                                                  */

     I2c_send ( Bit_data );                                                 /* Transmit generated bit data                      */
}

/*==============================================================================================================================*/
/*   Transmit bit data from IIC port synchronized with a clock (through port processing)                                        */
/*                                                                                                                              */
/*   Entry:    Bit_data (unsigned char) ; Bit data to be transmitted                                                            */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_send ( unsigned char Bit_data ){

          I2c_set (0,Bit_data);                                             /* Output SCL = 0, SDA = bit_dat from port          */
          Wait_timer (TM);                                                  /* Wait for the specified time                      */

          I2c_set (1,Bit_data);                                             /* Output SCL = 1, SDA = bit_dat from port          */
          Wait_timer (TM);                                                  /* Wait for the specified time                      */

          I2c_set (0,Bit_data);                                             /* Output SCL = 0, SDA = bit_dat from port          */
          Wait_timer (TM);                                                  /* Wait for the specified time                      */
}
```

```
/*==============================================================================================================================*/
/*   Obtain acknowledgement from IIC device (through port processing)                                                           */
/*   (read data at ninth clock)                                                                                                 */
/*                                                                                                                              */
/*   Entry:    Bit_data (unsigned char) ; Bit data to be transmitted                                                            */
/*   Return:   Ack_flag (unsigned char) ; Acknowledgement                                                                       */
/*                  0: Normal, 1: Error                                                                                         */
/*                                                                                                                              */
/*==============================================================================================================================*/
unsigned char  I2c_ackck ( void ){

     unsigned char Ack_flag;

     I2c_sda_in();                                                           /* Set P57 (SCL) as output and P56 (SDA) as input   */

     I2c_set (0,0);                                                          /* Output SCL = 0, SDA = 1 from port                */
     Wait_timer (TM);                                                        /* Wait tiemr                                       */

     I2c_set (1,0);                                                          /* Output SCL = 1, SDA = from port                  */
     Wait_timer (TM);                                                        /* Wait for the specified time                      */

     Ack_flag = PDR5 & 0x40;                                                 /* Obtain acknowledgement (SDA: P56)                */
     Wait_timer (TM);                                                        /* Wait for the specified time                      */

     I2c_set (0,0);                                                          /* Output SCL = 0, SDA = 1 from port                */
     Wait_timer (TM);                                                        /* Wait for the specified time                      */

     if (Ack_flag != 0){                                                     /* If the acknowledgement value                     */
                                                                             /* (SDA: P56) is 1, set 1                           */
          Ack_flag = 1;                                                      /*                                                  */
     }                                                                       /*                                                  */

     return (Ack_flag);                                                      /* Return acknowledgement                           */
}

/*==============================================================================================================================*/
/*   Set SDA (P56) as input, and SCL (P57) as output of IIC port                                                                */
/*                                                                                                                              */
/*   Entry:    None ; None                                                                                                      */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_sda_in ( void ){

     PCR5 = 0x80;                                                            /* Set SDA (P56)as input, andSCL (P57) as output    */
}

/*==============================================================================================================================*/
/*   Set SDA (P56), SCL (P57) of IIC port as output, and output data                                                            */
/*                                                                                                                              */
/*   Entry:    Data (unsigned char) ; Output data (bit 7 = SCL, bit 6 = SDA)                                                    */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  I2c_sda_out ( unsigned char Data ){

     PDR5 = Data;                                                            /* Set Data to PDR5                                 */
     PCR5 = 0xC0;                                                            /* Set P57 (SCL), P56 (SDA) as output,              */
                                                                             /* and output Data                                  */
}
```

```
/*==============================================================================================================================*/
/*   Wait for the specified time (decrement wait counts until the count becomes 0)                                              */
/*                                                                                                                              */
/*   Entry:    Wait_cnt (unsigned char) ; Wait counts                                                                           */
/*   Return:   None ; None                                                                                                      */
/*                                                                                                                              */
/*==============================================================================================================================*/
void  Wait_timer ( unsigned short Wait_cnt ){

     unsigned short    cnt;

     for (cnt = 0; cnt < Wait_cnt; cnt ++){
     }
}
```

Revision Record

		Description	
Rev.	Date	Page	Summary
1.00	Sep.29.03	—	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors. Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.