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.

To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Cautions

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.

APPLICATION NOTE

Read/Write Accesses to the On-Chip EEPROM

Introduction

Data is written to and read from the on-chip EEPROM through the H8/3664 I^2C bus interface.

Target Device

H8/300H Tiny Series H8/3664N

Contents

1. Specifications 3
2. Function Used 4
3. Description of Operation 5
3.1 EEPROM Interface 5
3.2 Bus Format and Timing 5
3.3 Start Condition 5
3.4 Stop Condition 5
3.5 Acknowledgement 6
3.6 Slave Address 6
3.7 Writing Operation 7
3.8 Acknowledgement Polling 8
3.9 Reading Operation 9
4. Description of the Software 11
5. Flowchart 15
6. Header Listing 28
7. Program Listing 45

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi's or any third party's patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party's rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi's sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi's sales office for any questions regarding this document or Hitachi semiconductor products.

1. Specifications

1. Data is written to and read from the on-chip EEPROM through the H8/3664 I²C bus interface.
   For writing to and reading from the EEPROM:
   — 1 byte writing (Write_byte_EEPROM)
   — 1 byte reading (Read_byte_EEPROM)
   — Page (8 bytes) writing (Write_page_EEPROM)
   — Page (8 bytes) reading (Read_page_EEPROM)
   — n (1 to 512) bytes continuous reading (Read_n_EEPROM)

   The above functions are used to write one of the three patterns below to a 512-byte area of memory in the EEPROM (address range H'000 – H'1FF).
   — Writing pattern: H'00, H'01, H'02 – H'FE, H'FF, H'00, H'01, H'02 – H'FE, H'FF
   — Writing pattern: H'00, H'00, H'01, H'01, H'02, H'02 – H'FE, H'FE, H'FF, H'FF
   — Writing pattern: H'FF, H'FF – H'FF, H'FF (ALL H'FF)

2. The slave address for EEPROM connection is [1010000], and writing of the data starts at EEPROM memory address H'00.
3. One master device (H8/3664) and one slave device (EEPROM) are connected to the I²C bus of this system. Figure 1.1 shows an example of the connection between the H8/3664 and EEPROM.
4. The frequency of the clock for the transfer is assumed to be 400 kHz.

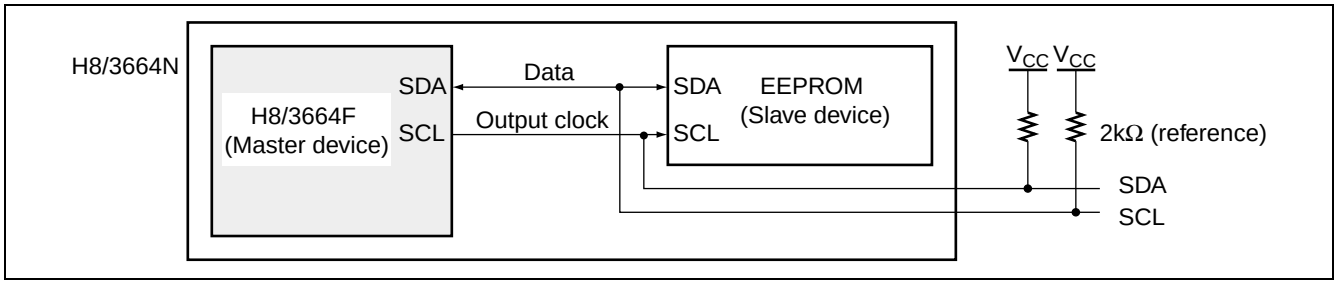


Figure 1.1 EEPROM Control through Connection with the I²C Bus Interface

2. Function Used

Figure 2.1 shows the basic format for data transfer across the I²C bus interface (writing a byte to the EEPROM).

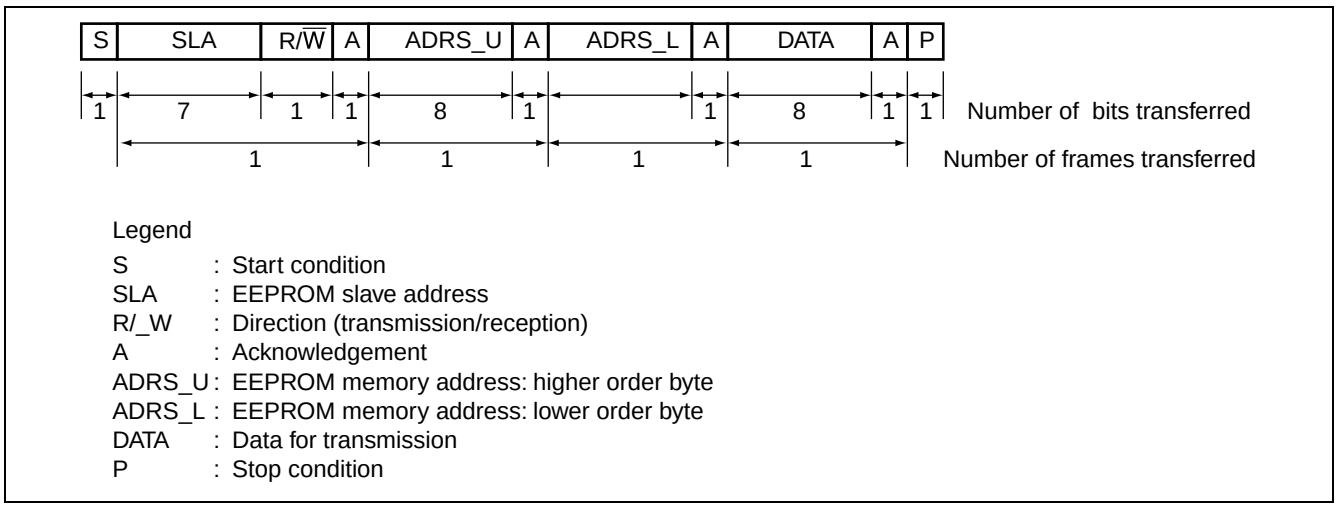


Figure 2.1 I²C Bus Interface Format

3. Description of Operation

3.1 EEPROM Interface

The H8/3664N is an LSI with a multi-chip structure, having both a H8/3664F and 4-kbit EEPROM mounted within the same package. The EEPROM is accessed through an I^2C bus interface. Since the I^2C bus is externally accessible, it is possible to communicate with other devices connected to the I^2C bus.

3.2 Bus Format and Timing

The format and timing of data transfer over the I^2C bus conform to the **I^2C bus format**. The following points only apply to the interface with the EEPROM.

1. EEPROM addresses are composed of two bytes; the order of transfer is higher-order byte then lower-order byte, with the most-significant bit (MSB) leading in each case.
2. The data to be written is also transmitted with the MSB.

The bus format and timing of EEPROM-data transfer over the I^2C bus are shown in Figure 3.1.

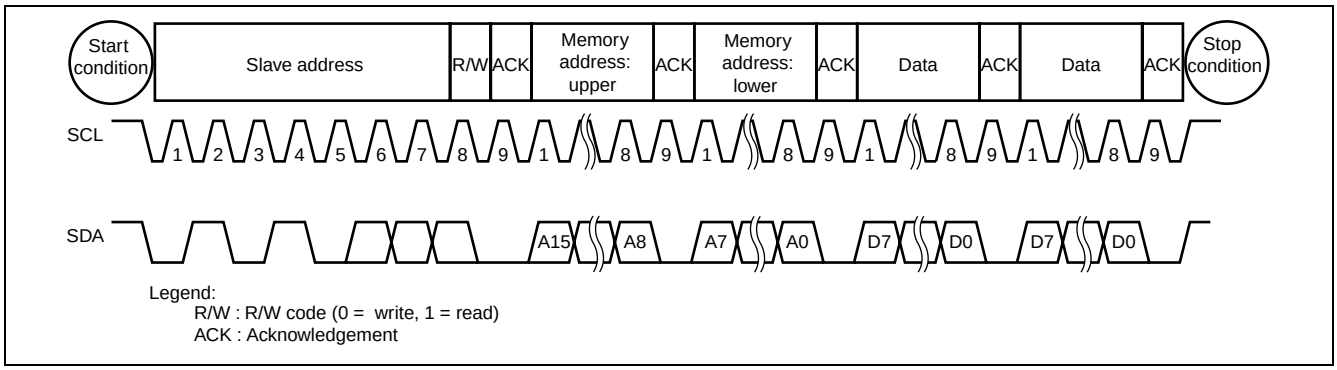


Figure 3.1 Bus Format and Timing for EEPROM Data Transfer

3.3 Start Condition

To initiate reading or writing, a start condition has to be generated by switching the SDA input from high to low level while the SCL input is high level.

3.4 Stop Condition

To stop reading or writing, a stop condition has to be generated by switching the SDA input from low level to high level while the SCL input is high level.

A stop condition has to be generated on completion of a read operation and places the bus in the access-standby mode.

The stop condition is generated for a write operation when input of the new data has been completed, and initiates program of the memory, which takes the period of one write cycle (t_{wc}). The bus is then placed in the access-standby mode.

3.5 Acknowledgement

The serial data, including address information and data read or to be written are transmitted and received in 8-bit units. The acknowledgement signal indicates the normal transmission or reception of an 8-bit unit.

In writing, EEPROM outputs the acknowledgement signal, "0", in each 9th clock cycle of data reception.

In reading, EEPROM transmits the reading data after the acknowledgement which follows the reception of data. The bus state becomes bus released after the transmission of each byte. The EEPROM transmits the next byte of data on detecting the acknowledgement signal. If the stop condition is received but the acknowledgement signal has not been detected, the reading operation is terminated and the bus is placed in the access standby mode. When neither the acknowledgement signal nor the stop condition is detected, no data is transmitted and the bus remains in the bus-released state.

3.6 Slave Address

In the sample task, this address is left in its initial state.

After generating the start condition, the processor sends a 7-bit slave address and 1-bit R/W code over the bus, selecting the EEPROM and its mode of operation. This input starts reading or writing of the EEPROM.
As shown in table 3.1, the slave address is composed of seven bits, a 4-bit device code in the higher-order bits and a 3-bit slave-address code in the lower order bits. The device code identifies the device type; in this LSI, the EEPROM's device code is fixed at the 1010 value that indicates general-purpose EEPROM.

The slave-address code identifies one of up to eight units with the 1010 device code that may be connected to the I^2C bus. The bits of the slave-address code are input in the order A2, A1 and A0; if the code coincides with that in the EEPROM's Slave-Address Inquiry Register (ESAR), the EEPROM is selected.

The slave-address code is stored in address H'FF09 of the EEPROM. Within the 10-ms period after the reset line is released, the code is transferred from the Slave Address Register location in the memory array to the Slave-Address Inquiry Register (ESAR). Note that the EEPROM is not accessible during this transfer.

The initial value of the slave-address code in the ESAR is H'00. However, any value in the range H'00 – H'07 can be written to this register. Be sure to use byte-writing to overwrite this value (in this sample task, the address is used in its initial state).

The next bit after the slave address is the R/W code. Writing is selected by '0' and reading is selected by '1'.

When the device code is not 1010 or the slave-address code does not match, the EEPROM stays in the access-standby mode.

Table 3.1 Slave Address

Bit	Bit Name	Initial Value	Setting	Remarks
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	A2	The initial value is not fixed.
2	Slave address code A1	0	A1	The initial value is not fixed.
1	Slave address code A0	0	A0	The initial value is not fixed.

3.7 Writing Operation

Two writing operations have been defined: byte writing (one byte at a time) and page writing (eight bytes at a time). To start writing operation, input 0 in the R/W code following the slave address.

1. Byte writing

   After the correct seven slave-address bits and R/W code "0" (for writing) have been input to the EEPROM, it outputs the acknowledgement signal, "0", in the 9th bit and enters the write mode. The EEPROM then receives the successive two bytes of the memory address, with the higher-order byte first, and responds to each with an acknowledgement. The processor then sends the data to be written, and the EEPROM again responds with an acknowledgement. All of the transfers are MSB-first.

   Generation of the stop condition after this initiates the control sequence for overwriting within the EEPROM, which does not receive further input over the I^2C bus (SCL and SDA lines) until this operation has been completed. On completion of the overwriting operation, the EEPROM automatically returns to the access-standby mode.

   Figure 3.2 shows the sequence of the byte-writing operation.

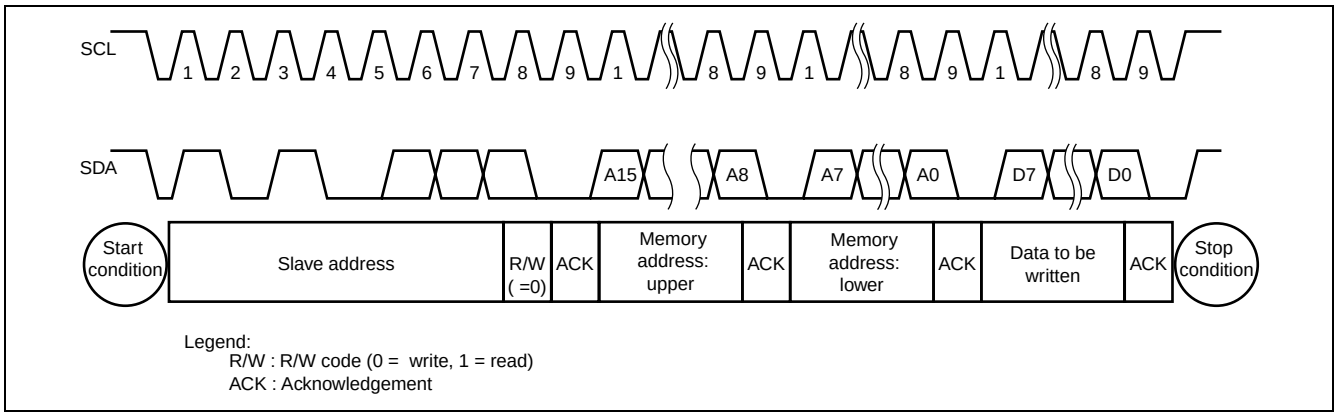


Figure 3.2 Byte Writing

2. Page writing

This LSI provides a page-writing function, which allows the rewriting of up to eight bytes at a time. As with byte writing, the EEPROM receives data in this order: slave address + R/W code → memory address (n) → write data (Dn), acknowledging reception in every 9th bit. However, when the stop condition is not generated after the input of a given byte of data for writing (Dn), with a further byte of data for writing (Dn+1) received instead, the page-writing mode is entered. When this data for writing (Dn+1) is received, the three least significant bits (A2 to A0) of the EEPROM address are automatically incremented to the address (n+1). This allows the input of up to eight consecutive bytes of data for writing.

As each byte of data for writing is received, the address within the page is incremented. When the three least significant bits (A2 to A0) of the EEPROM address reach the final address of a page, the address counter rolls over to return to the first address on the page. In this case, data is written to the same address two or more times, but only the final value is valid. Generation of the stop condition ends the input of data for writing and the actual overwriting operation commences. Figure 3.3 shows the sequence of the page-writing operation.

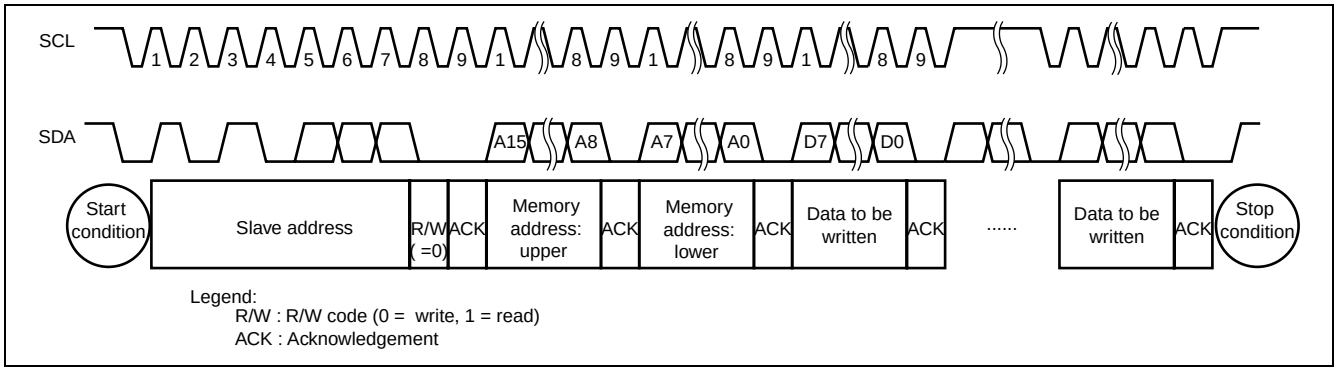


Figure 3.3 Page Writing

3.8 Acknowledgement Polling

The processor uses an acknowledgement-polling function to judge whether or not overwriting of the EEPROM is in progress. In the polling function, the slave address + R/W code are sent to the EEPROM after generation of the start condition. For acknowledgement polling, adjust the R/W code to 0. The value of the acknowledgement signal in the 9th bit is tested; a "1" indicates that overwriting is still in progress state while a "0" indicates completion. The function of acknowledgement polling becomes active at the moment the stop condition is output, after the input of data for writing.

3.9 Reading Operation

There are three kinds of reading operation, current-address reading, random-address reading and sequential reading. A read operation is started in the same way as a write operation, except that the R/W code which follows the slave address is set to "1" to indicate reading.

1. Current-address reading

   After reading from or writing to location (n) of the EEPROM, its internal address counter holds (n+1). A read operation in the current-address reading mode reads the address (n+1).

   In the same way as a write operation, reading starts with input to the EEPROM in this order: start condition → slave address + R/W code (R/W=1). The EEPROM outputs an acknowledgement "0" in next bit and then outputs the byte of data at address (n+1), with the MSB first. Input to the EEPROM in the order acknowledgement = "1" (indicating that bus release is possible without the input of an acknowledgement) → stop condition ends the read operation and returns the EEPROM to the access-standby mode.

   When read access ends and the last address accessed was H' 01FF, the current address counter rolls over to indicate the 0th address. That is, when the final address on a page is accessed, the current address counter returns to the first address on the page.

   The current address remains in effect as long as the power supply is not turned off. The initial value of the current address after the power supply has been turned off is not specified. Please use random-address reading (explained below) to specify the address after the power supply has been turned on. Figure 3.4 shows the sequence of current-address reading.

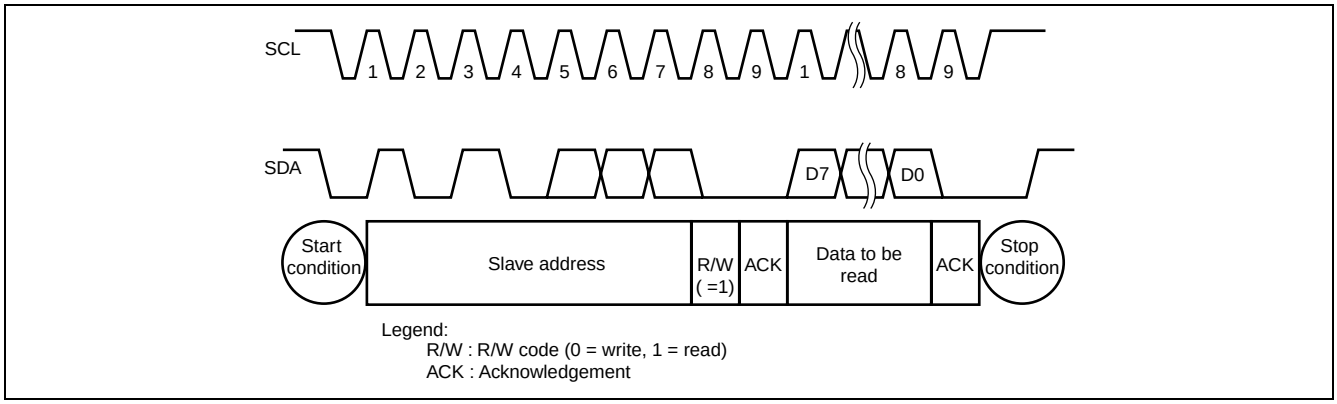


Figure 3.4 Current-Address Reading

2. Random-address reading

This mode is used to read data from a specified address. A dummy write operation is used to set the address to be read. This involves input in this order: start condition → slave address + R/W code (R/W=0) → memory address (upper) → memory address (lower). After the EEPROM has confirmed reception with an acknowledgement "0" output following input of the second memory-address byte, the processor again generates the start condition and executes a current-address read, which now reads data from the address specified by the dummy write operation. After output of the read data, input to the EEPROM in the order '1' (indicating that bus release is possible without the input of an acknowledgement) → the stop condition ends the read operation and returns the EEPROM to the access-standby mode. Figure 3.5 shows the sequence of random-address reading.

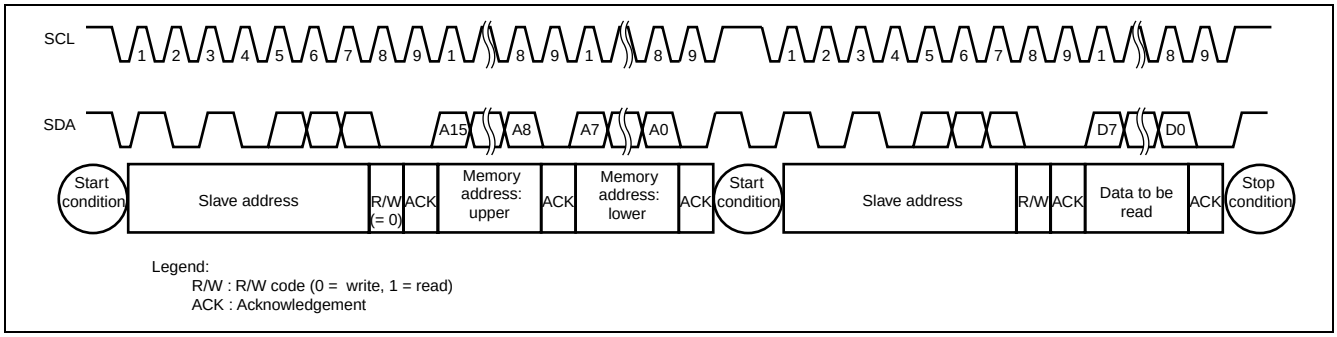


Figure 3.5 Random Address Reading

3. Sequential reading

This mode is used for the continuous reading of data, which can be started through either current-address reading or random-address reading. When the EEPROM receives the acknowledgement signal "0" after the output of one byte of data, its current address is incremented and it outputs the next byte of data. Consecutive bytes continue to be output as long as the output is followed by acknowledgement, i.e., "0". After the address counter reaches the last address, H'01FF, it rolls over to indicate the 0^{th} address. Sequential reading can then continue from this location.

Sequential reading is ended by the input, in order, of acknowledgement "1" (indicating that bus release is possible without the input of an acknowledgement) → stop condition, in the same way as current-address and random-address reading.

Figure 3.6 shows the sequence of data in sequential reading initiated by current-address reading.

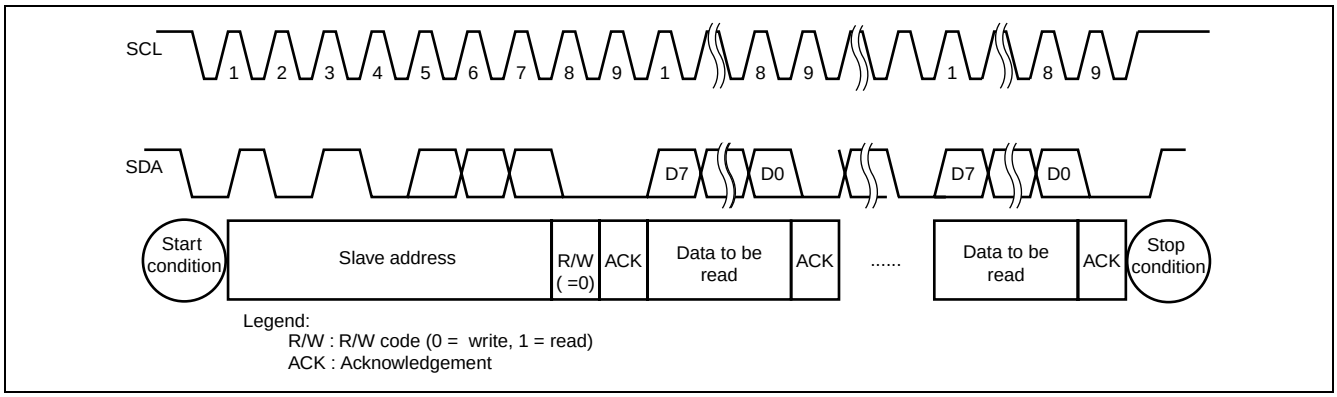


Figure 3.6 Sequential Reading (Initiated by Current-Address Reading)

4. Description of the Software

Module description:

The modules of this sample task are listed in table 4.1.

Table 4.1 Module Description

Module (Function) Name	Arguments	Return Value	Functions
INIT (assembly language)	None	None	Sets stack pointer (H'FF80 is set in R7) Sets CCR (disable interrupts) Jump to main.
main	None	None	Main module
Test_EEPROM	test_code (Test item: 0-5)	None	Tests items of EEPROM operation 0: Default 1: 1-byte writing, 1-byte reading Data written: H'00 - H'FF (repeated once) 2: 8-byte writing Data written: H'00, H'00 - H'FF, H'FF 3: 8-byte reading 4: 8-byte writing Data written: All H'FF 5: 512-byte reading
wait	Number of wait loops	None	Wait (100: About 100µsec)
Write_byte_EEPROM	adrs (target address), data (data to be written)	ack (0: noack/ 1: ack)	Writes 1 byte to the EEPROM Checks the acknowledgement. When writing is disabled (ACK = 1), waits 1 msec and re-tries 10 times.
Read_byte_EEPROM	adrs (target address)	Data (Reading data)	Reads 1 byte from the EEPROM
Write_page_EEPROM	adrs (target address), wr_ptr (address where the data to be written is stored)	ack (0: noack/ 1: ack)	Writes 8 bytes to the EEPROM Checks the acknowledgement. When writing is disabled (ACK = 1), waits 1 msec and re-tries 10 times.
Read_page_EEPROM	adrs (target address), rd_ptr (address for storage of read data)	ack (0: noack/ 1: ack)	Reads 8 bytes from the EEPROM

Table 4.1 Module Description (cont)

Module (function) name	Arguments	Return value	Functions
Read_n_EEPROM	adrs (Reading data), rd_ptr (address for storage of read data), reading number	ack (0: noack/ 1: ack)	Reads n bytes from the EEPROM.
Set_adrs_EEPROM	adrs (target address)	ack (0: noack/ 1: ack)	Sets the EEPROM address for writing. Sets the EEPROM address for reading (through a dummy write operation)
Write_data_EEPROM	wr_data (data to write)	ack (0: noack/ 1: ack)	Transmits data for writing over the IIC
Recv_data1_EEPROM	None	Data (Reading data)	Receives 1 byte from the IIC
Recv_datan_EEPROM	adrs (target address), data (address for storage of read data)	ack (0: noack/ 1: ack)	Receives n bytes from the IIC
wait_e	Number of wait loops	None	Wait (n = 100: about 100 µsec)

Table 4.2 gives the descriptions of the global variables in this sample task.

Table 4.2 Global Variable Used

Variable name	Type	Size	Usage
eeprom_buf	unsigned char	512	EEPROM buffer for writing/reading (same size as the built-in EEPROM)
test_code	unsigned char	1	EEPROM testing item selection command
dummy	unsigned char	1	Dummy variable used in reading from the EEPROM's internal registers

Table 4.3 gives the descriptions of the constants (definitions) in this sample task.

Table 4.3 Constants Used (definitions)

Definition name	Type	Value	Usage
DEVICE_CODE	unsigned char	0xA0	Device code that selects the EEPROM (slave) (fixation bits 7-0:1010 ----)
SLAVE_ADRS	unsigned char	0x00	Address code that selects the EEPROM (slave) (Default bits 7-0: ---- 000-)
IIC_DATA_W	unsigned char	0x00	R/W code that selects writing to the EEPROM (W[bits 7-0]:---- ---0)
IIC_DATA_R	unsigned char	0x01	R/W code that selects reading to the EEPROM (R[bits 7-0]:---- ---1)
WR_RETRY_CNT	unsigned char	10	Limit on the number of consecutive attempts at writing to the EEPROM After each write operation, repeat ten times each ms until ACK is received.
WR_OK	unsigned char	11	Value used to break out of loop processing when ACK is received after writing.

Table 4.4 gives the descriptions of the internal register used in this sample task.

Table 4.4 Usage of Internal Registers

Register	Bit	Function	Operation	Setting
ICDR		• Stores transmission/received data.	Storage/reference	—
SAR	FS	• Sets the transmission format, in conjunction with FSX in SARX and SW in DDCSWR.	No operation	0
SARX	FSX	• Sets the transmission format, in conjunction with FS of SAR and SW of DDCSWR.	No operation	1
ICMR	MLS	• Selects the order of bits as MSB first.	Setting	0
	WAIT	• Selects continuous transmission of data and acknowledgement.	Setting	0
	CKS2-0	• Sets the frequency of the transfer clock to 400kHz, in combination with the IICX bit of STCR.	Setting	CKS2 = 0 CKS1 = 0 CKS0 = 1
	BC2-0	• Sets the number of bits transferred in each frame of the I^2C format at nine.	Setting	BC2 = 0 BC1 = 0 BC0 = 0

Table 4.4 Usage of Internal Registers (cont)

Register Bit		Function	Operation	Setting
ICCR	ICE	• Controls access to the ICMR, ICDR / SAR and SARX registers. • Selects I²C bus interface operation (SCL / SDA pin is port function) / non operation (SCL / SDA pin is bus driving state).	Setting	0/1
	IEIC	• Disables I²C bus interface interrupt requests.	Setting	0
	MST	• Uses I²C bus interface in the master mode.	Setting	1
	TRS	• Uses I²C bus interface in the transmission mode.	Setting	0/1
	ACKE	• An acknowledgement bit value of '1' interrupts continuous transfer.	Setting	0/1
	BBSY	• Confirms whether or not the I²C bus is in released state. • In combination with SCP, generates start and stop conditions.	Storage/reference	0/1
	IRIC	• Detects the start condition • Judges the end of data transmission • Detects acknowledgement bit '1'.	Setting	0/1
	SCP	• In combination of BBSY, generates start and stop conditions.	Setting	0/1
ICSR	ESTP	• Error stop condition detection flag (Slave mode in effect)	No operation	—
	STOP	• Normal stop condition detection flag (Slave mode in effect)	No operation	—
	IRTR	• Continuous transmission / reception interrupt-request flag	No operation	—
	AASX	• Second slave address recognition flag	No operation	—
	AL	• Arbitration lost flag	No operation	—
	AAS	• Slave address recognition flag	No operation	—
	ADZ	• General call address recognition flag	No operation	—
	ACKB	• Stores the values of acknowledgement bits from the EEPROM	Reference	—
TSCR	IICRST	• Resets the I²C controller	No operation	—
	IICX	• Selects transmission rate	Setting	—

5. Flowchart

Figure 5.1 is a chart of the module hierarchy.

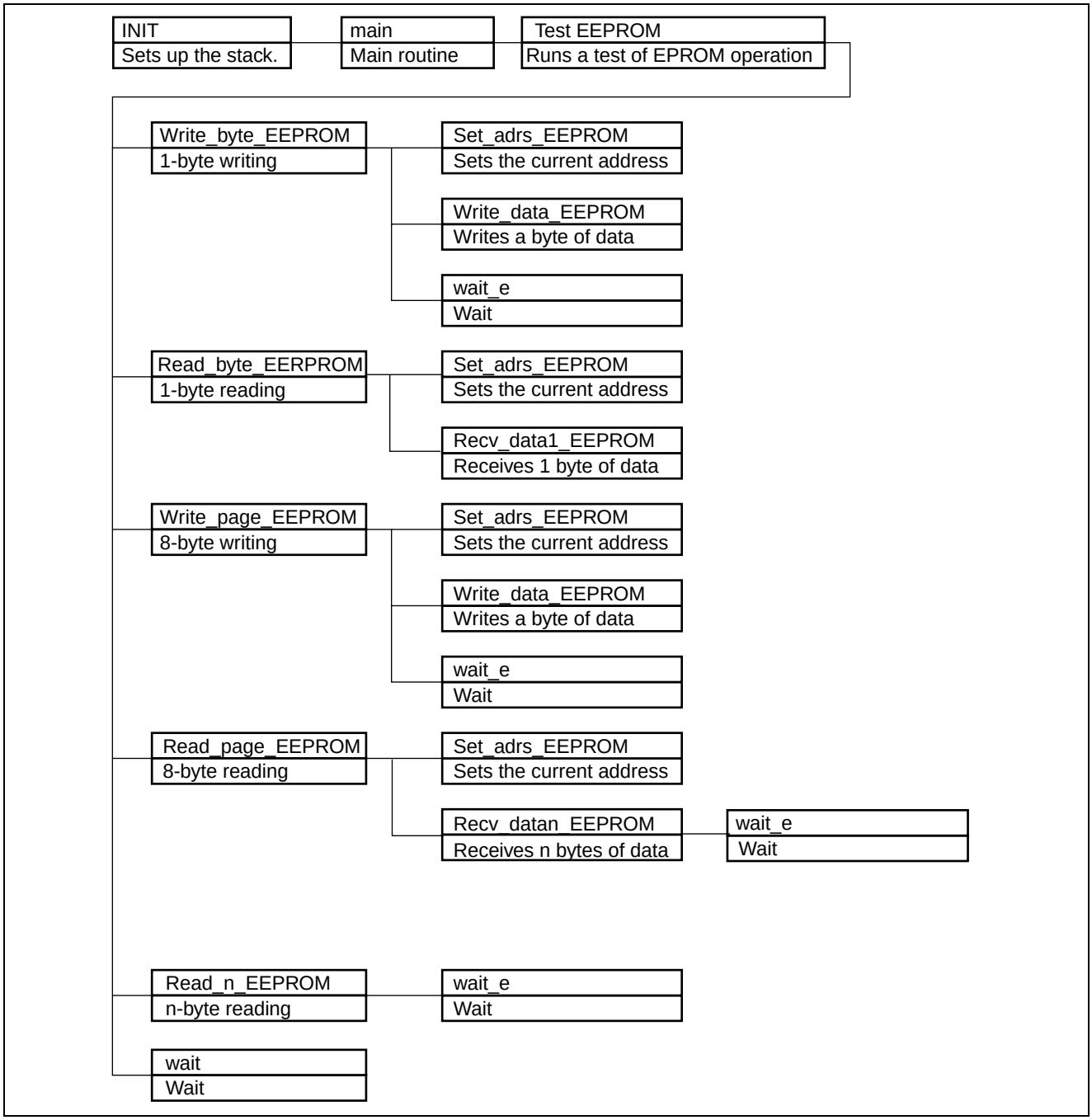
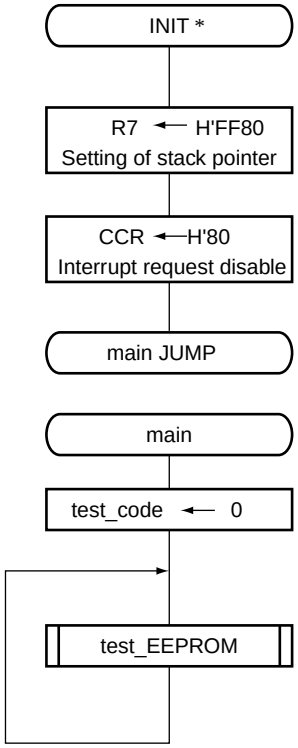


Figure 5.1 Module Hierarchical Chart

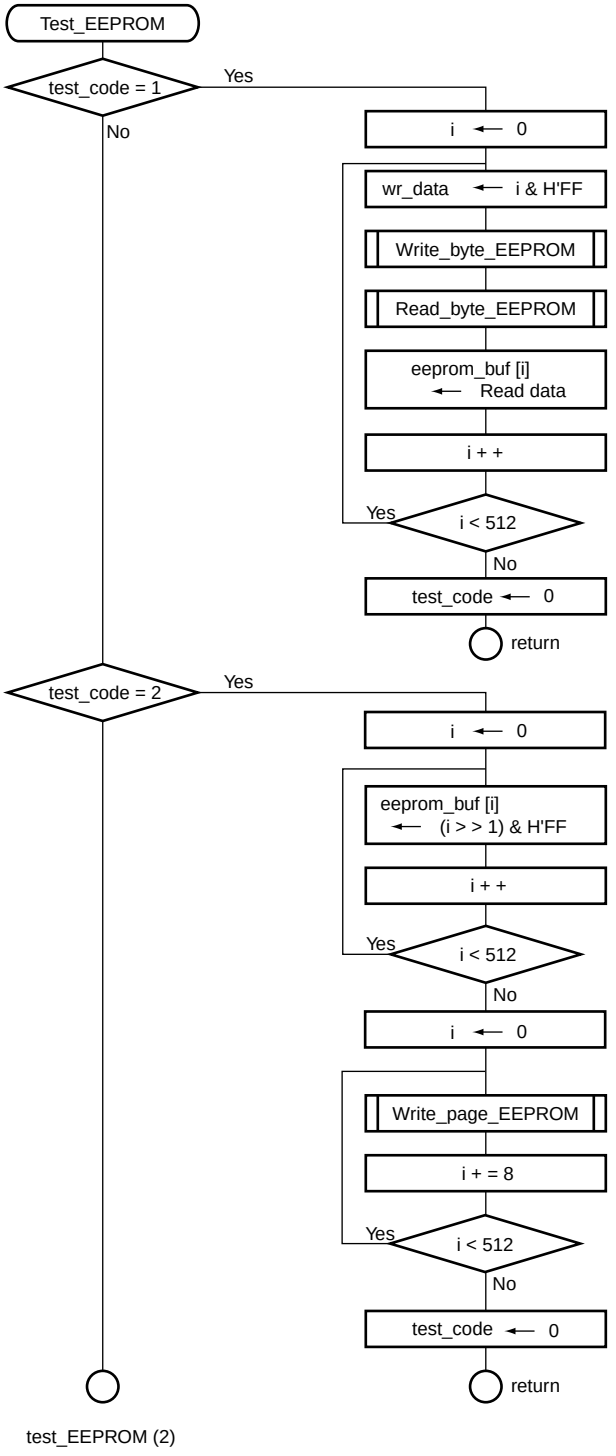


..... When test_code is 0, EEPROM control processing is not performed.
Therefore, test_code must be separately set from software or, with an E10T or similar emulator, through memory manipulation.
In this sample task, the emulator's dump function is used to check data read into the EEPROM. Operation changes as the value in memory at test_code is varied within the range from one to five.

Note: * In this sample task, the stack and CCR settings are covered by INIT.src (assembly language).

Link address specification:

Section Name	Address
CV1	H'0000
P	H'0100
B	H'FF80



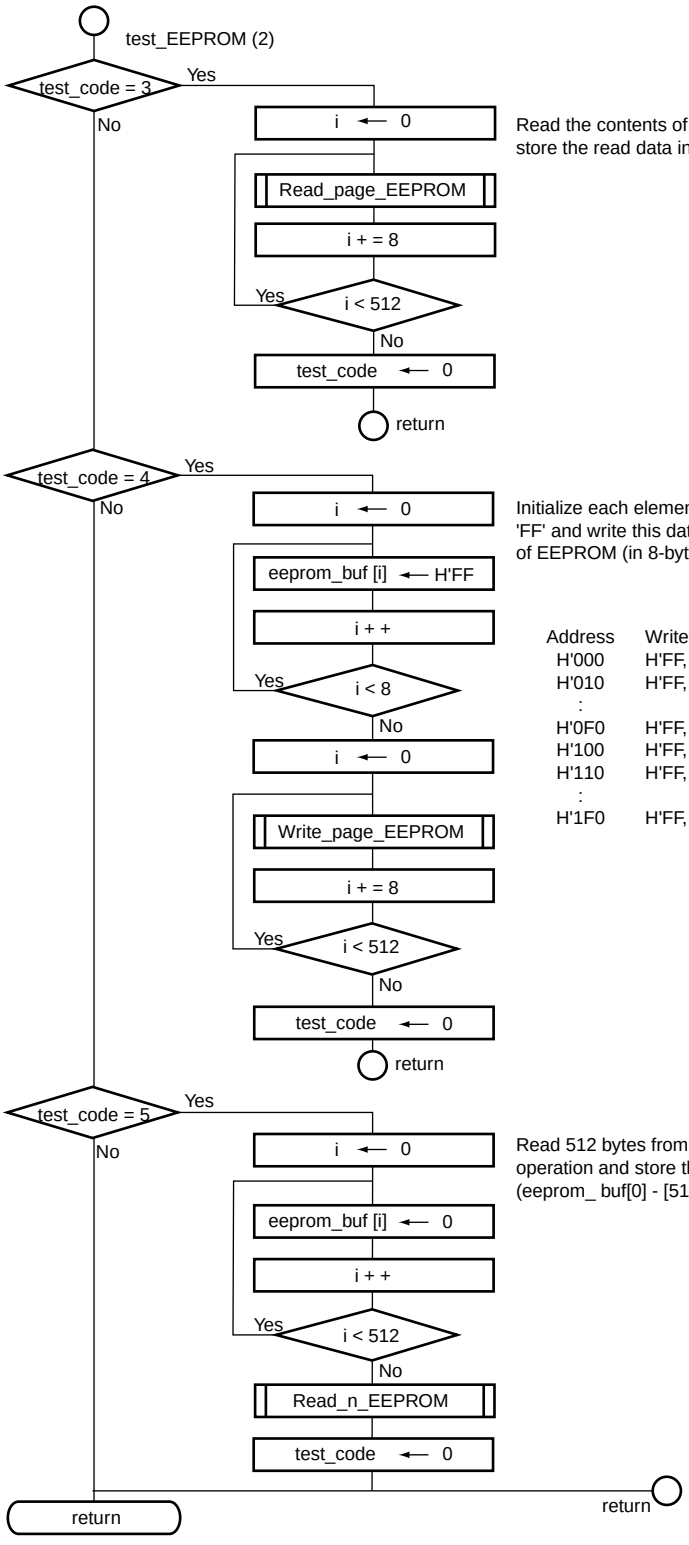
test_EEPROM (2)

Write data equal to the lower-order 8 bits of the address to the EEPROM, read 1 byte from the target address, and store this read data in an array (eeprom_buf[0] - [511]).

Address	Write data
H'000	H'00, H'01, H'02..... H'0F
H'010	H'10, H'11, H'12..... H'1F
:	
H'0F0	H'F0, H'F1, H'F2..... H'FF
H'100	H'00, H'01, H'02..... H'0F
H'110	H'10, H'11, H'12..... H'1F
:	
H'1F0	H'F0, H'F1, H'F2..... H'FF

Store the 8 higher-order bits of the 9-bit loop counter in an array (eeprom_buf[0] - [511]). Transfer data, in eight-byte units (write page), from the stored array to the EEPROM.

Address	Write data
H'000	H'00, H'00, H'01..... H'07
H'010	H'08, H'08, H'09..... H'0F
:	
H'0F0	H'78, H'78, H'79..... H'7F
H'100	H'80, H'80, H'81..... H'87
H'110	H'88, H'88, H'89..... H'8F
:	
H'1F0	H'F8, H'F8, H'F9..... H'FF

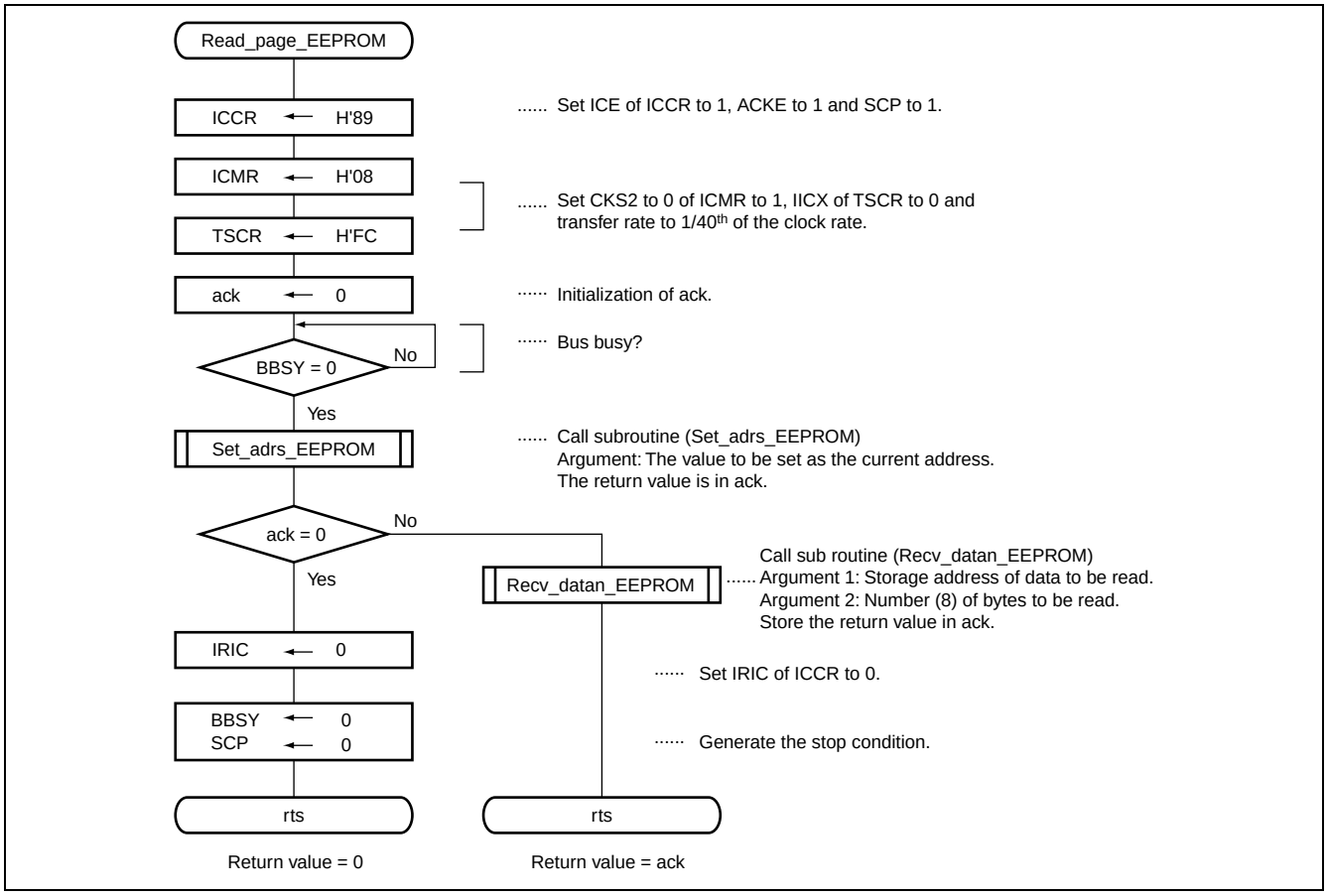


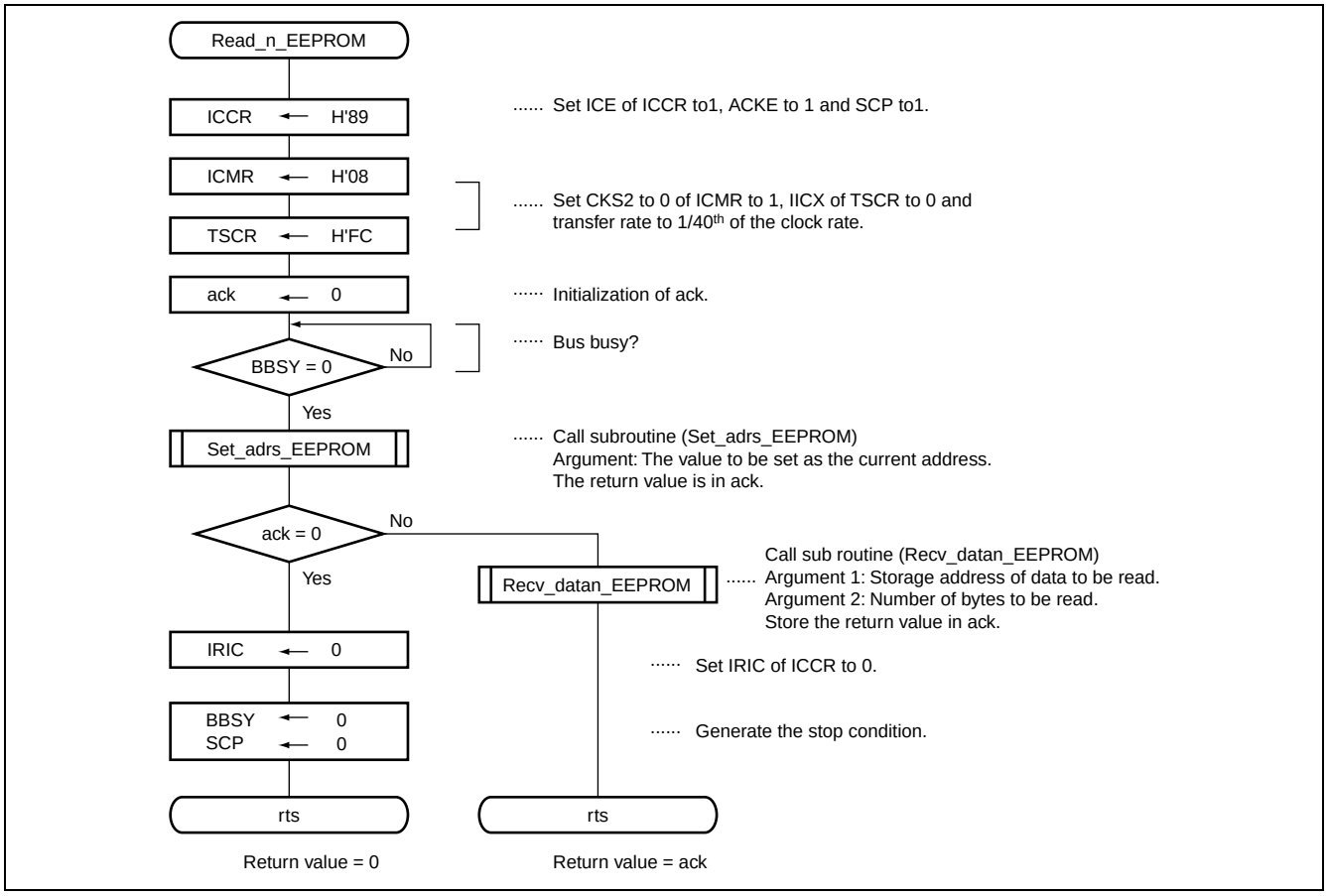
Read the contents of the EEPROM in 8-byte units and store the read data in an array (eeprom_ buf[0] - [511]).

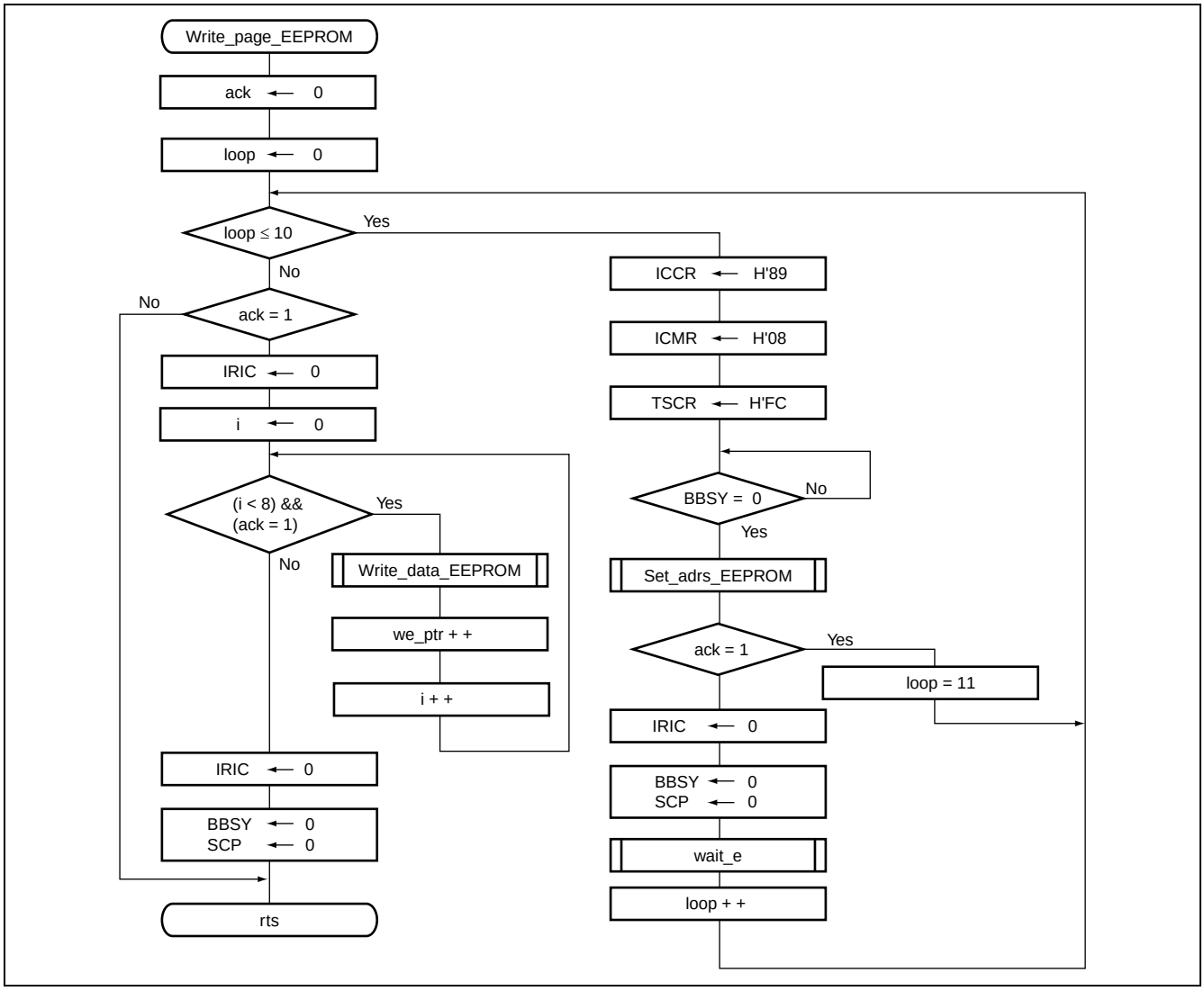
Initialize each element of an array (eeprom_ buf[0] - [7] to 'FF' and write this data to the location from H'000 to H'1FF of EEPROM (in 8-byte units).

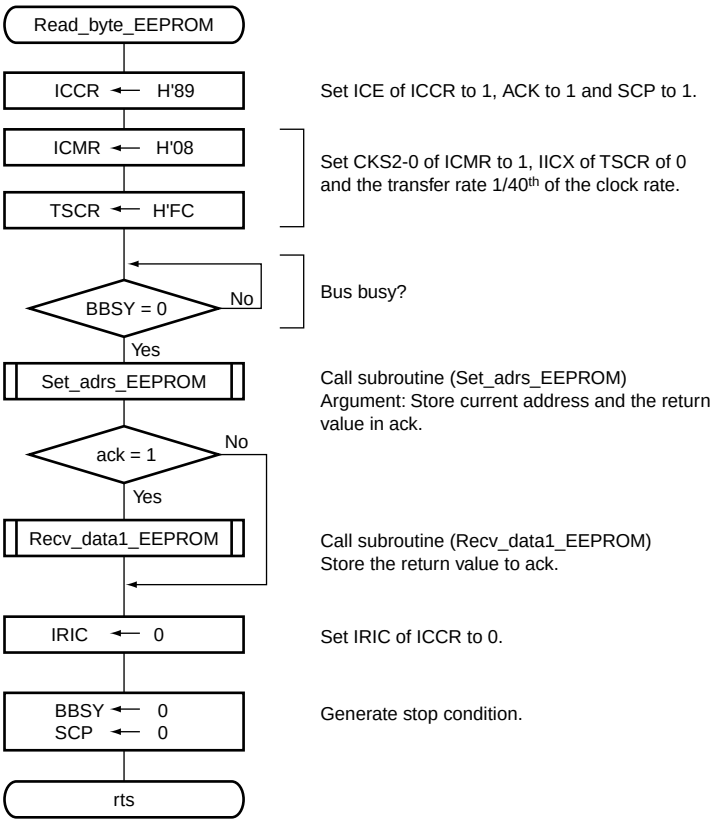
Address	Write data
H'000	H'FF, H'FF, H'FF..... H'FF
H'010	H'FF, H'FF, H'FF..... H'FF
:	
H'0F0	H'FF, H'FF, H'FF..... H'FF
H'100	H'FF, H'FF, H'FF..... H'FF
H'110	H'FF, H'FF, H'FF..... H'FF
:	
H'1F0	H'FF, H'FF, H'FF..... H'FF

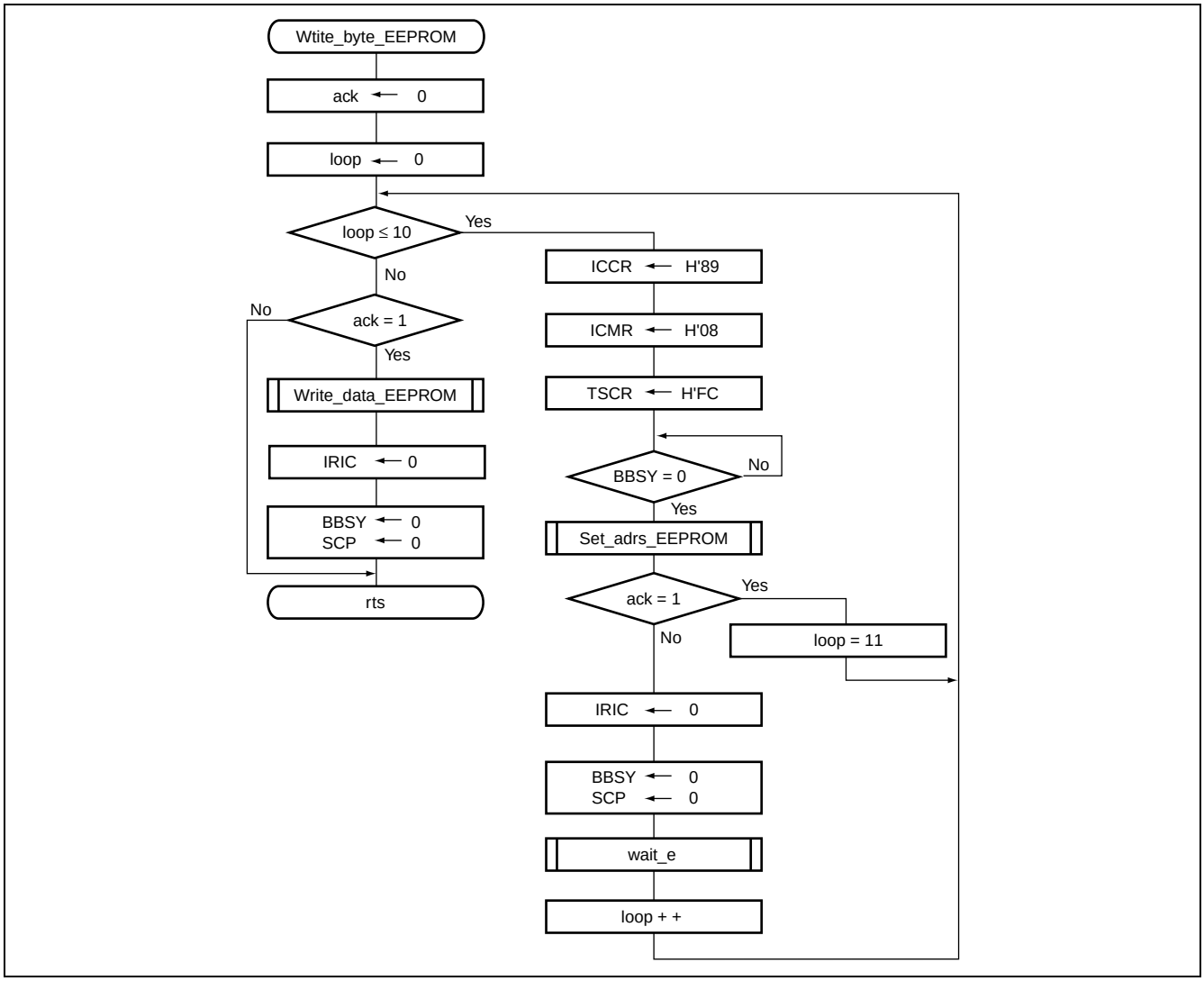
Read 512 bytes from the EEPROM in a continuous operation and store them in an array (eeprom_ buf[0] - [511]).

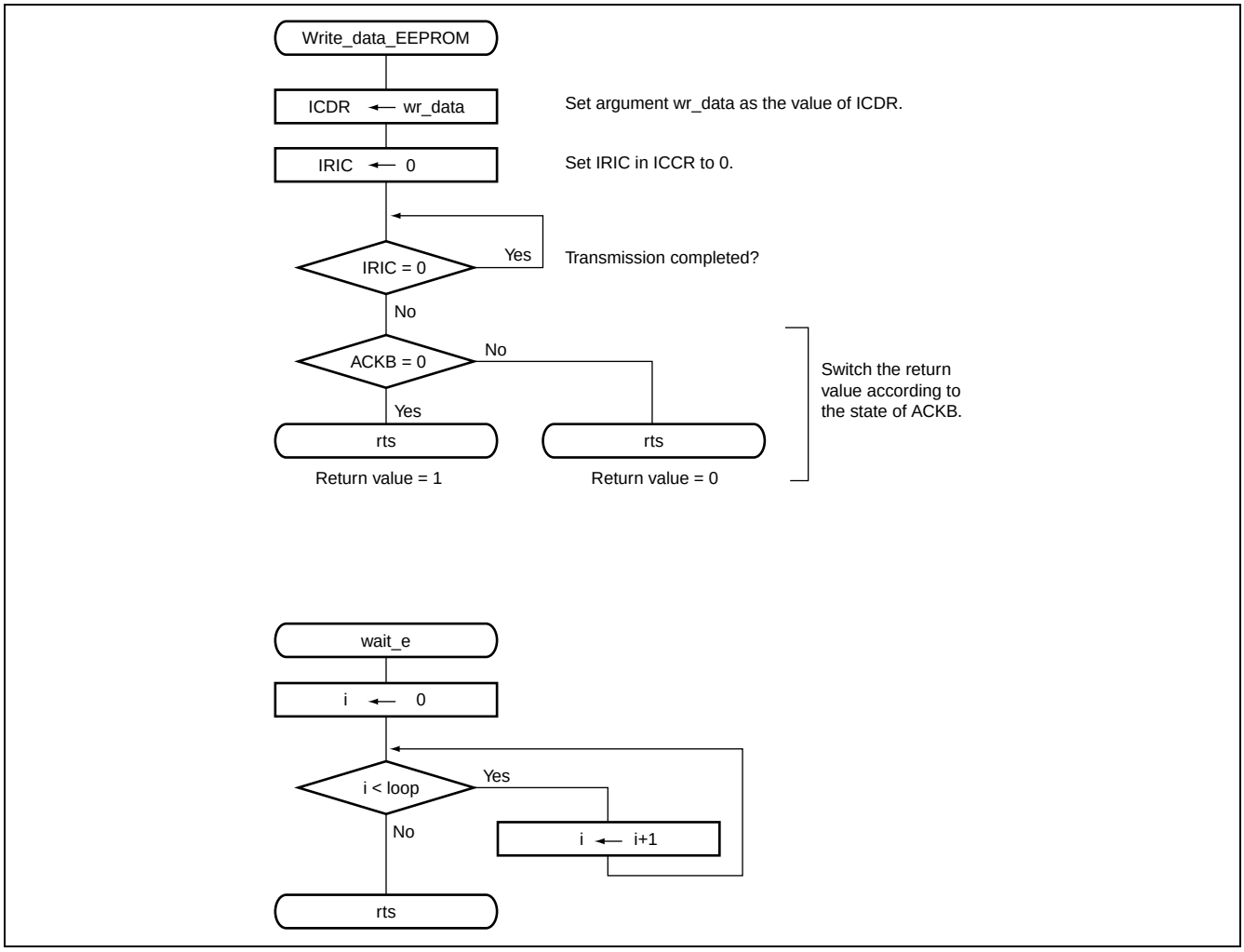

Read_page_EEPROM
ICCR ← H'89
ICMR ← H'08
TSCR ← H'FC
ack ← 0
BBSY = 0
No
Yes
Set_adrs_EEPROM
ack = 0
No
Yes
IRIC ← 0
BBSY ← 0
SCP ← 0
rts
Return value = 0
Recv_datan_EEPROM
rts
Return value = ack
..... Set ICE of ICCR to 1, ACKE to 1 and SCP to 1.
..... Set CKS2 to 0 of ICMR to 1, IICX of TSCR to 0 and transfer rate to 1/40th of the clock rate.
..... Initialization of ack.
..... Bus busy?
..... Call subroutine (Set_adrs_EEPROM)
Argument: The value to be set as the current address.
The return value is in ack.
Call sub routine (Recv_datan_EEPROM)
..... Argument 1: Storage address of data to be read.
Argument 2: Number (8) of bytes to be read.
Store the return value in ack.
..... Set IRIC of ICCR to 0.
..... Generate the stop condition.

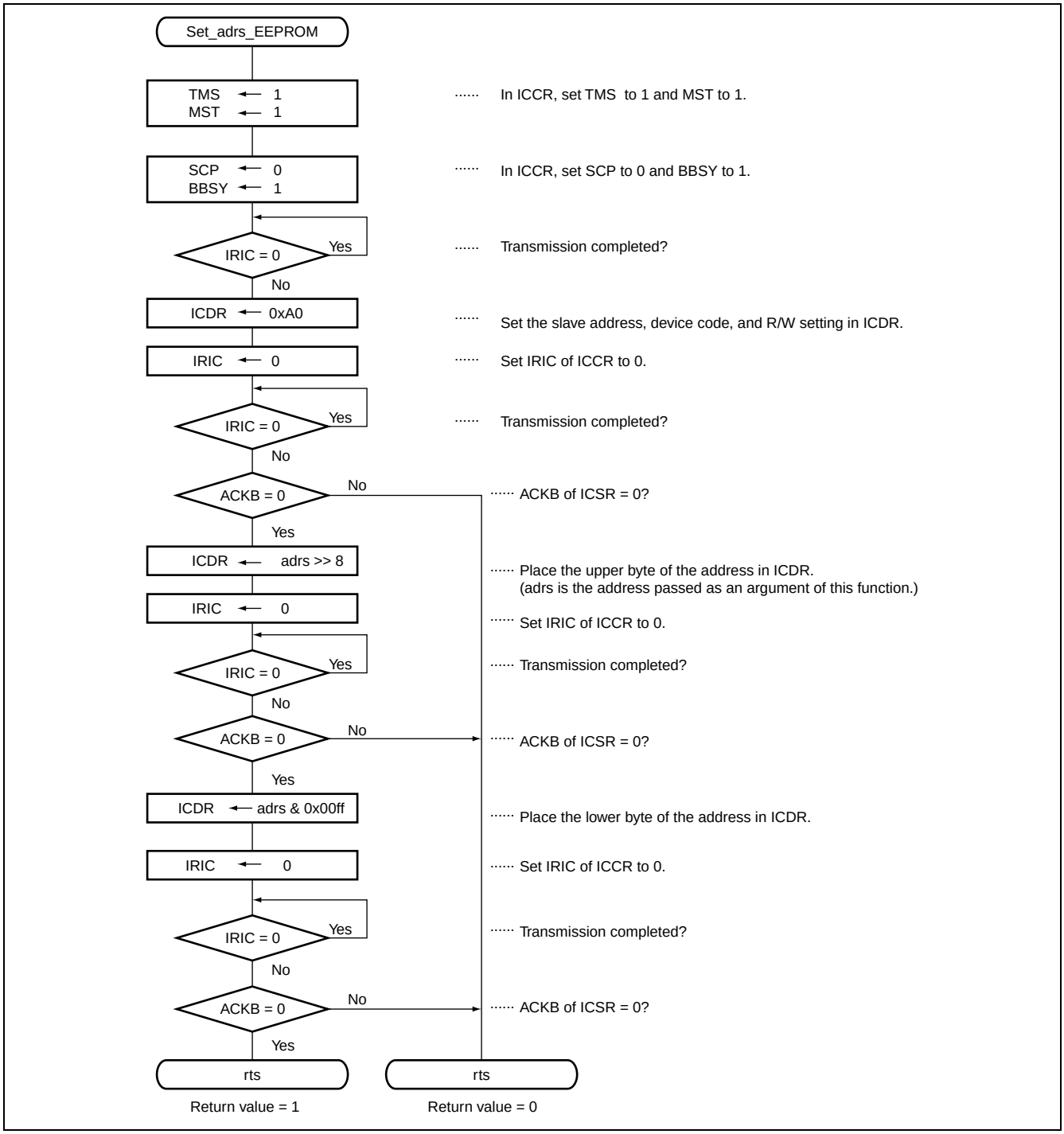

Read_n_EEPROM
ICCR ← H'89
ICMR ← H'08
TSCR ← H'FC
ack ← 0
BBSY = 0
No
Yes
Set_adrs_EEPROM
ack = 0
No
Yes
Recv_datan_EEPROM
IRIC ← 0
BBSY ← 0
SCP ← 0
rts
rts
Return value = 0
Return value = ack
..... Set ICE of ICCR to1, ACKE to 1 and SCP to1.
..... Set CKS2 to 0 of ICMR to 1, IICX of TSCR to 0 and transfer rate to 1/40th of the clock rate.
..... Initialization of ack.
..... Bus busy?
..... Call subroutine (Set_adrs_EEPROM)
Argument: The value to be set as the current address.
The return value is in ack.
Call sub routine (Recv_datan_EEPROM)
..... Argument 1: Storage address of data to be read.
Argument 2: Number of bytes to be read.
Store the return value in ack.
..... Set IRIC of ICCR to 0.
..... Generate the stop condition.

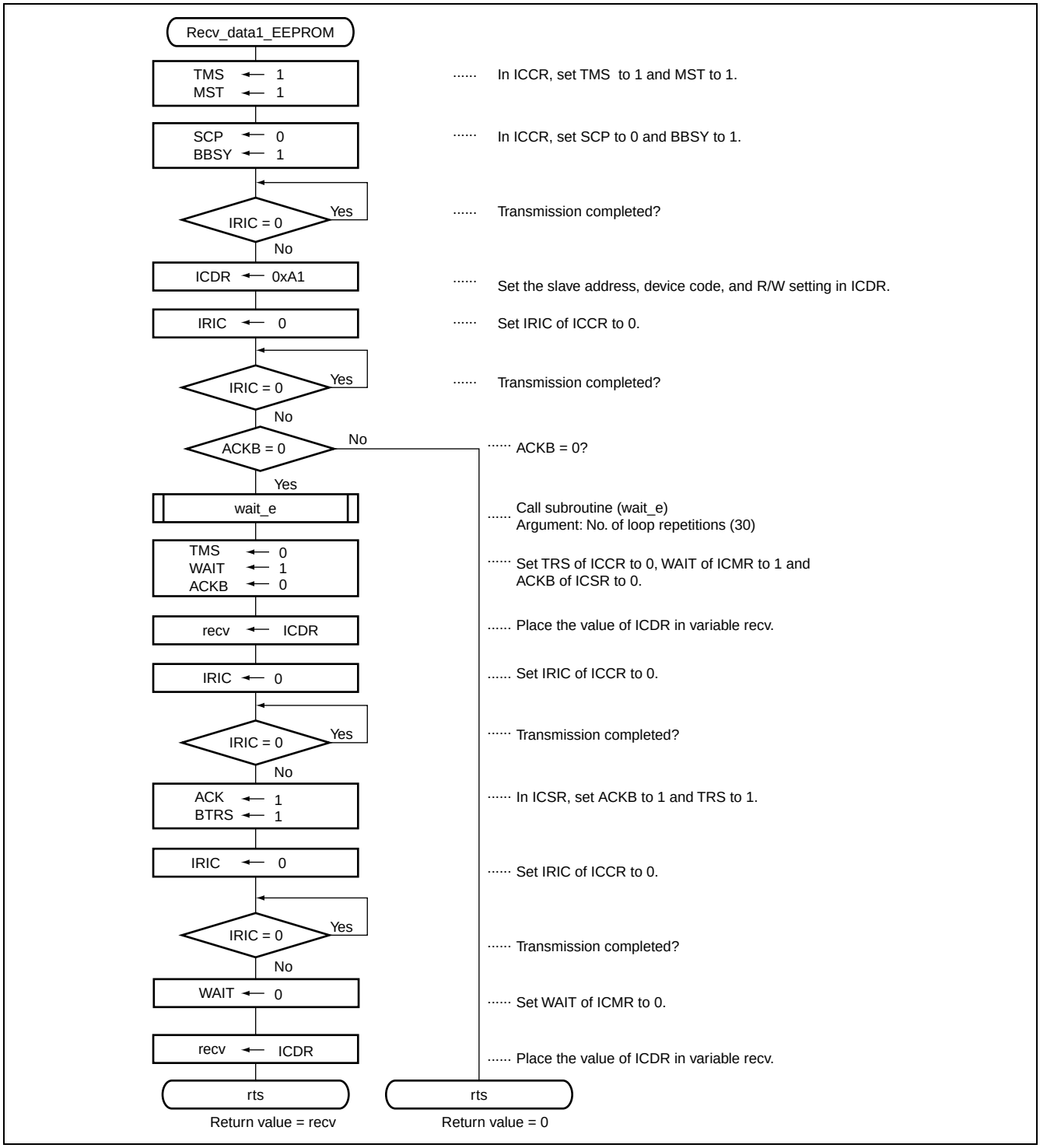

Write_page_EEPROM
ack ← 0
loop ← 0
loop ≤ 10
Yes
No
No
ack = 1
IRIC ← 0
i ← 0
(i < 8) &&
(ack = 1)
Yes
No
Write_data_EEPROM
we_ptr + +
i + +
IRIC ← 0
BBSY ← 0
SCP ← 0
rts
ICCR ← H'89
ICMR ← H'08
TSCR ← H'FC
BBSY = 0
No
Yes
Set_adrs_EEPROM
ack = 1
Yes
loop = 11
IRIC ← 0
BBSY ← 0
SCP ← 0
wait_e
loop + +

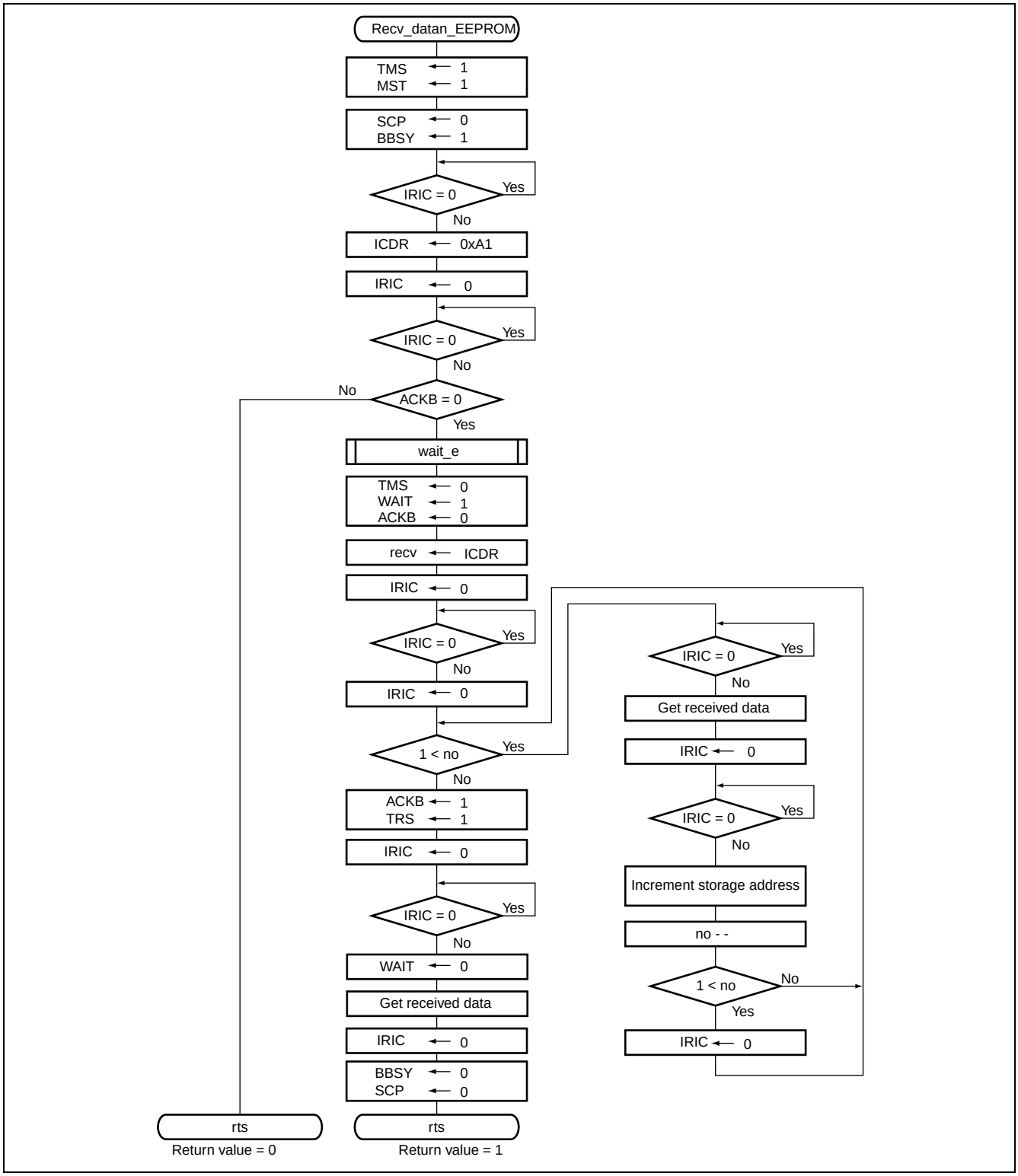

Read_byte_EEPROM
ICCR ← H'89
Set ICE of ICCR to 1, ACK to 1 and SCP to 1.
ICMR ← H'08
TSCR ← H'FC
Set CKS2-0 of ICMR to 1, IICX of TSCR of 0 and the transfer rate 1/40th of the clock rate.
BBSY = 0
No
Yes
Bus busy?
Set_adrs_EEPROM
Call subroutine (Set_adrs_EEPROM)
Argument: Store current address and the return value in ack.
ack = 1
No
Yes
Recv_data1_EEPROM
Call subroutine (Recv_data1_EEPROM)
Store the return value to ack.
IRIC ← 0
Set IRIC of ICCR to 0.
BBSY ← 0
SCP ← 0
Generate stop condition.
rts


Wtite_byte_EEPROM
ack ← 0
loop ← 0
loop ≤ 10
Yes
No
ack = 1
No
Yes
Write_data_EEPROM
IRIC ← 0
BBSY ← 0
SCP ← 0
rts
ICCR ← H'89
ICMR ← H'08
TSCR ← H'FC
BBSY = 0
No
Yes
Set_adrs_EEPROM
ack = 1
Yes
No
loop = 11
IRIC ← 0
BBSY ← 0
SCP ← 0
wait_e
loop + +


Write_data_EEPROM
ICDR ← wr_data
Set argument wr_data as the value of ICDR.
IRIC ← 0
Set IRIC in ICCR to 0.
IRIC = 0
Yes
Transmission completed?
No
ACKB = 0
No
Yes
Switch the return value according to the state of ACKB.
rts
Return value = 1
rts
Return value = 0
wait_e
i ← 0
i < loop
Yes
No
i ← i+1
rts


Set_adrs_EEPROM
TMS ← 1
MST ← 1
In ICCR, set TMS to 1 and MST to 1.
SCP ← 0
BBSY ← 1
In ICCR, set SCP to 0 and BBSY to 1.
IRIC = 0
Yes
No
Transmission completed?
ICDR ← 0xA0
Set the slave address, device code, and R/W setting in ICDR.
IRIC ← 0
Set IRIC of ICCR to 0.
IRIC = 0
Yes
No
Transmission completed?
ACKB = 0
No
Yes
ACKB of ICSR = 0?
ICDR ← adrs >> 8
Place the upper byte of the address in ICDR.
(adrs is the address passed as an argument of this function.)
IRIC ← 0
Set IRIC of ICCR to 0.
IRIC = 0
Yes
No
Transmission completed?
ACKB = 0
No
Yes
ACKB of ICSR = 0?
ICDR ← adrs & 0x00ff
Place the lower byte of the address in ICDR.
IRIC ← 0
Set IRIC of ICCR to 0.
IRIC = 0
Yes
No
Transmission completed?
ACKB = 0
No
Yes
ACKB of ICSR = 0?
rts
rts
Return value = 1
Return value = 0


Recv_data1_EEPROM
TMS ← 1
MST ← 1
In ICCR, set TMS to 1 and MST to 1.
SCP ← 0
BBSY ← 1
In ICCR, set SCP to 0 and BBSY to 1.
IRIC = 0
Yes
No
Transmission completed?
ICDR ← 0xA1
Set the slave address, device code, and R/W setting in ICDR.
IRIC ← 0
Set IRIC of ICCR to 0.
IRIC = 0
Yes
No
Transmission completed?
ACKB = 0
No
Yes
ACKB = 0?
wait_e
Call subroutine (wait_e)
Argument: No. of loop repetitions (30)
TMS ← 0
WAIT ← 1
ACKB ← 0
Set TRS of ICCR to 0, WAIT of ICMR to 1 and ACKB of ICSR to 0.
recv ← ICDR
Place the value of ICDR in variable recv.
IRIC ← 0
Set IRIC of ICCR to 0.
IRIC = 0
Yes
No
Transmission completed?
ACK ← 1
BTRS ← 1
In ICSR, set ACKB to 1 and TRS to 1.
IRIC ← 0
Set IRIC of ICCR to 0.
IRIC = 0
Yes
No
Transmission completed?
WAIT ← 0
Set WAIT of ICMR to 0.
recv ← ICDR
Place the value of ICDR in variable recv.
rts
Return value = recv
rts
Return value = 0


Recv_datan_EEPROM
TMS ← 1
MST ← 1
SCP ← 0
BBSY ← 1
IRIC = 0
Yes
No
ICDR ← 0xA1
IRIC ← 0
IRIC = 0
Yes
No
No
ACKB = 0
Yes
wait_e
TMS ← 0
WAIT ← 1
ACKB ← 0
recv ← ICDR
IRIC ← 0
IRIC = 0
Yes
No
IRIC ← 0
1 < no
Yes
No
ACKB ← 1
TRS ← 1
IRIC ← 0
IRIC = 0
Yes
No
WAIT ← 0
Get received data
IRIC ← 0
BBSY ← 0
SCP ← 0
rts
Return value = 0
rts
Return value = 1
IRIC = 0
Yes
No
Get received data
IRIC ← 0
IRIC = 0
Yes
No
Increment storage address
no - -
1 < no
No
Yes
IRIC ← 0

6. Header Listing

```
File : H8_3664_C.h
/*****************************************************************************************************************/
/*      H8/3664F Include File                          Ver 0.1                                                   */
/*****************************************************************************************************************/
struct st_flash {                                                   /* struct FLASH                  */
union {                                                             /* FLMCR1                        */
unsigned char BYTE;                                                 /* Byte Access                   */
struct {                                                            /* Bit  Access                   */
unsigned char wk :1;                                                /*                               */
unsigned char SWE:1;                                                /* SWE                           */
unsigned char ESU:1;                                                /* ESU                           */
unsigned char PSU:1;                                                /* PSU                           */
unsigned char EV :1;                                                /* EV                            */
unsigned char PV :1;                                                /* PV                            */
unsigned char E  :1;                                                /* E                             */
unsigned char P  :1;                                                /* P                             */
} BIT;                                                              /*                               */
} FLMCR1;                                                           /*                               */
union {                                                             /* FLMCR2                        */
unsigned char BYTE;                                                 /* Byte Access                   */
struct {                                                            /* Bit  Access                   */
unsigned char FLER:1;                                               /* FLER                          */
} BIT;                                                              /*                               */
} FLMCR2;                                                           /*                               */
union {                                                             /* FLPWCR                        */
unsigned char BYTE;                                                 /* Byte Access                   */
struct {                                                            /* Bit  Access                   */
unsigned char PDWND:1;                                              /* PDWND                         */
} BIT;                                                              /*                               */
} FLPWCR;                                                           /*                               */
union {                                                             /* EBR1                          */
unsigned char BYTE;                                                 /* Byte Access                   */
struct {                                                            /* Bit  Access                   */
unsigned char wk :3;                                                /*                               */
unsigned char EB4:1;                                                /* EB4                           */
unsigned char EB3:1;                                                /* EB3                           */
unsigned char EB2:1;                                                /* EB2                           */
unsigned char EB1:1;                                                /* EB1                           */
```

```
unsigned char EB0:1;                    /* EB0                  */
} BIT;                                  /*                      */
} EBR1;                                 /*                      */
char wk[7];                             /*                      */
union {                                 /* FENR                 */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char FLSHE:1;                  /* FLSHE                */
} BIT;                                  /*                      */
} FENR;                                 /*                      */
};                                      /*                      */

struct st_ta {                          /* struct TA            */
union {                                 /* TMA                  */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char CKSO:3;                   /* CKSO                 */
unsigned char     :1;                   /*                      */
unsigned char CKSI:4;                   /* CKSI                 */
} BIT;                                  /*                      */
} TMA;                                  /*                      */
unsigned char TCA;                      /* TCA                  */
};                                      /*                      */
struct st_tv {                          /* struct TV            */
union {                                 /* TCRV0                */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char CMIEB:1;                  /* CMIEB                */
unsigned char CMIEA:1;                  /* CMIEA                */
unsigned char OVIE :1;                  /* OVIE                 */
unsigned char CCLR :2;                  /* CCLR                 */
unsigned char CKS  :3;                  /* CKS                  */
} BIT;                                  /*                      */
} TCRV0;                                /*                      */
union {                                 /* TCSRV                */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char CMFB:1;                   /* CMFB                 */
unsigned char CMFA:1;                   /* CMFA                 */
unsigned char OVF :1;                   /* OVF                  */
```

```
unsigned char     :1;                    /*                          */
unsigned char OS  :4;                    /* OS                       */
} BIT;                                   /*                          */
} TCSRV;                                 /*                          */
unsigned char TCORA;                     /* TCORA                    */
unsigned char TCORB;                     /* TCORB                    */
unsigned char TCNTV;                     /* TCNT                     */
union {                                  /* TCRV1                    */
unsigned char BYTE;                      /* Byte Access              */
struct {                                 /*Bit  Access               */
unsigned char wk  :3;                    /*                          */
unsigned char TVEG:2;                    /* TVEG                     */
unsigned char TRGE:1;                    /* TRGE                     */
unsigned char     :1;                    /*                          */
unsigned char ICKS:1;                    /* ICKS                     */
} BIT;                                   /*                          */
} TCRV1;                                 /*                          */
};                                       /*                          */

struct st_tw {                           /* struct TW                */
union {                                  /* TMRW                     */
unsigned char BYTE;                      /* Byte Access              */
struct {                                 /* Bit  Access              */
unsigned char CTS  :1;                   /* CTS                      */
unsigned char      :1;                   /*                          */
unsigned char BUFEA:1;                   /* BUFEA                    */
unsigned char BUFEB:1;                   /* BUFEB                    */
unsigned char      :1;                   /*                          */
unsigned char PWMD :1;                   /* PWMD                     */
unsigned char PWMC :1;                   /* PWMC                     */
unsigned char PWMB :1;                   /* PWMB                     */
} BIT;                                   /*                          */
} TMRW;                                  /*                          */
union {                                  /* TCRW                     */
unsigned char BYTE;                      /* Byte Access              */
struct {                                 /* Bit  Access              */
unsigned char CCLR:1;                    /* CCLR                     */
unsigned char CKS :3;                    /* CKS                      */
unsigned char TOD :1;                    /* TOD                      */
unsigned char TOC :1;                    /* TOC                      */
```

```
unsigned char TOB :1;                    /* TOB                    */
unsigned char TOA :1;                    /* TOA                    */
} BIT;                                   /*                        */
} TCRW;                                  /*                        */
union {                                  /* TIERW                  */
unsigned char BYTE;                      /* Byte Access            */
struct {                                 /* Bit  Access            */
unsigned char OVIE :1;                   /* OVIE                   */
unsigned char      :3;                   /*                        */
unsigned char IMIED:1;                   /* IMIED                  */
unsigned char IMIEC:1;                   /* IMIEC                  */
unsigned char IMIEB:1;                   /* IMIEB                  */
unsigned char IMIEA:1;                   /* IMIEA                  */
} BIT;                                   /*                        */
} TIERW;                                 /*                        */
union {                                  /* TSRW                   */
unsigned char BYTE;                      /* Byte Access            */
struct {                                 /* Bit  Access            */
unsigned char OVF :1;                    /* OVF                    */
unsigned char     :3;                    /*                        */
unsigned char IMFD:1;                    /* IMFD                   */
unsigned char IMFC:1;                    /* IMFC                   */
unsigned char IMFB:1;                    /* IMFB                   */
unsigned char IMFA:1;                    /* IMFA                   */
} BIT;                                   /*                        */
} TSRW;                                  /*                        */

union {                                  /* TIOR0                  */
unsigned char BYTE;                      /* Byte Access            */
struct {                                 /* Bit  Access            */
unsigned char wk :1;                     /*                        */
unsigned char IOB:3;                     /* IOB                    */
unsigned char    :1;                     /*                        */
unsigned char IOA:3;                     /* IOA                    */
} BIT;                                   /*                        */
} TIOR0;                                 /*                        */
union {                                  /* TIOR1                  */
unsigned char BYTE;                      /* Byte Access            */
struct {                                 /* Bit  Access            */
unsigned char wk :1;                     /*                        */
```

```
unsigned char IOD:3;                    /* IOD                  */
unsigned char    :1;                    /*                      */
unsigned char IOC:3;                    /* IOC                  */
} BIT;                                  /*                      */
} TIOR1;                                /*                      */
unsigned int TCNT;                      /* TCNT                 */
unsigned int GRA;                       /* GRA                  */
unsigned int GRB;                       /* GRB                  */
unsigned int GRC;                       /* GRC                  */
unsigned int GRD;                       /* GRD                  */
};                                      /*                      */
struct st_sci3 {                        /* struct SCI3          */
union {                                 /* SMR                  */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char COM :1;                   /* COM                  */
unsigned char CHR :1;                   /* CHR                  */
unsigned char PE  :1;                   /* PE                   */
unsigned char PM  :1;                   /* PM                   */
unsigned char STOP:1;                   /* STOP                 */
unsigned char MP  :1;                   /* MP                   */
unsigned char CKS :2;                   /* CKS                  */
} BIT;                                  /*                      */
} SMR;                                  /*                      */
unsigned char BRR;                      /* BRR                  */
union {                                 /* SCR3                 */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char TIE :1;                   /* TIE                  */
unsigned char RIE :1;                   /* RIE                  */
unsigned char TE  :1;                   /* TE                   */
unsigned char RE  :1;                   /* RE                   */
unsigned char MPIE:1;                   /* MPIE                 */
unsigned char TEIE:1;                   /* TEIE                 */
unsigned char CKE :2;                   /* CKE                  */
} BIT;                                  /*                      */
} SCR3;                                 /*                      */
unsigned char TDR;                      /* TDR                  */
```

```
union {                                 /* SSR                  */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char TDRE:1;                   /* TDRE                 */
unsigned char RDRF:1;                   /* RDRF                 */
unsigned char OER :1;                   /* OER                  */
unsigned char FER :1;                   /* FER                  */
unsigned char PER :1;                   /* PER                  */
unsigned char TEND:1;                   /* TEND                 */
unsigned char MPBR:1;                   /* MPBR                 */
unsigned char MPBT:1;                   /* MPBT                 */
} BIT;                                  /*                      */
} SSR;                                  /*                      */
unsigned char RDR;                      /* RDR                  */
};                                      /*                      */
struct st_ad {                          /* struct A/D           */
unsigned int DRA;                       /* ADDRA                */
unsigned int DRB;                       /* ADDRB                */
unsigned int DRC;                       /* ADDRC                */
unsigned int DRD;                       /* ADDRD                */
union {                                 /* ADCSR                */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char ADF :1;                   /* ADF                  */
unsigned char ADIE:1;                   /* ADIE                 */
unsigned char ADST:1;                   /* ADST                 */
unsigned char SCAN:1;                   /* SCAN                 */
unsigned char CKS :1;                   /* CKS                  */
unsigned char CH  :3;                   /* CH                   */
} BIT;                                  /*                      */
} CSR;                                  /*                      */
union {                                 /* ADCR                 */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char TRGE:1;                   /* TRGE                 */
} BIT;                                  /*                      */
} CR;                                   /*                      */
};                                      /*                      */
struct st_wdt {                         /* struct WDT           */
union {                                 /* TCSRWD               */
```

```
unsigned char BYTE;                    /* Byte Access                */
struct {                               /* Bit  Access                */
unsigned char B6WI  :1;                /* B6WI                       */
unsigned char TCWE  :1;                /* TCWE                       */
unsigned char B4WI  :1;                /* B4WI                       */
unsigned char TCSRWE:1;                /* TCSRWE                     */
unsigned char B2WI  :1;                /* B2WI                       */
unsigned char WDON  :1;                /* WDON                       */
unsigned char B0WI  :1;                /* B0WI                       */
unsigned char WRST  :1;                /* WRST                       */
} BIT;                                 /*                            */
} TCSRWD;                              /*                            */

unsigned char TCWD;                    /* TCWD                       */
union {                                /* TMWD                       */
unsigned char BYTE;                    /* Byte Access                */
struct {                               /* Bit  Access                */
unsigned char wk :4;                   /*                            */
unsigned char CKS:4;                   /* CKS                        */
} BIT;                                 /*                            */
} TMWD;                                /*                            */
};                                     /*                            */
struct st_iic {                        /* struct IIC                 */
union {                                /* ICCR                       */
unsigned char BYTE;                    /* Byte Access                */
struct {                               /* Bit  Access                */
unsigned char ICE :1;                  /* ICE                        */
unsigned char IEIC:1;                  /* IEIC                       */
unsigned char MST :1;                  /* MST                        */
unsigned char TRS :1;                  /* TRS                        */
unsigned char ACKE:1;                  /* ACKE                       */
unsigned char BBSY:1;                  /* BBSY                       */
unsigned char IRIC:1;                  /* IRIC                       */
unsigned char SCP :1;                  /* SCP                        */
} BIT;                                 /*                            */
} ICCR;                                /*                            */
union {                                /* ICSR                       */
unsigned char BYTE;                    /* Byte Access                */
struct {                               /* Bit  Access                */
unsigned char ESTP:1;                  /* ESTP                       */
```

```
unsigned char STOP:1;                   /* STOP                         */
unsigned char IRTR:1;                   /* IRTR                         */
unsigned char AASX:1;                   /* AASX                         */
unsigned char AL  :1;                   /* AL                           */
unsigned char AAS :1;                   /* AAS                          */
unsigned char ADZ :1;                   /* ADZ                          */
unsigned char ACKB:1;                   /* ACKB                         */
} BIT;                                  /*                              */
} ICSR;                                 /*                              */
union {                                 /*                              */
struct {                                /*                              */
union {                                 /* SARX                         */
unsigned char BYTE;                     /* Byte Access                  */
struct {                                /* Bit  Access                  */
unsigned char SVAX:7;                   /* SVAX                         */
unsigned char FSX :1;                   /* FSX                          */
} BIT;                                  /*                              */
} UN_SARX;                              /*                              */
union {                                 /* SAR                          */
unsigned char BYTE;                     /* Byte Access                  */
struct {                                /* Bit  Access                  */
unsigned char SVA:7;                    /* SVA                          */
unsigned char FS :1;                    /* FS                           */
} BIT;                                  /*                              */
} UN_SAR;                               /*                                */
} ICE0;                                 /*                              */
struct {                                /*                              */
unsigned char UN_ICDR;                  /* ICDR                         */
union {                                 /* ICMR                         */
unsigned char BYTE;                     /* Byte Access                  */
struct {                                /* Bit  Access                  */
unsigned char MLS :1;                   /* MLS                          */
unsigned char WAIT:1;                   /* WAIT                         */
unsigned char CKS :3;                   /* CKS                          */
unsigned char BC  :3;                   /* BC                           */
} BIT;                                  /*                              */
} UN_ICMR;                              /*                              */
} ICE1;                                 /*                              */
} EQU;                                  /*                              */
};                                      /*                              */
```

```
struct st_abrk {                                /* struct ABRK                  */
union {                                         /* ABRKCR                       */
unsigned char BYTE;                             /* Byte Access                  */
struct {                                        /* Bit  Access                  */
unsigned char RTINTE:1;                         /*    RTINTE                    */
unsigned char CSEL  :2;                         /* CSEL                         */
unsigned char ACMP  :3;                         /* ACMP                         */
unsigned char DCMP  :2;                         /* DCMP                         */
} BIT;                                          /*                              */
} CR;                                           /*                              */
union {                                         /* ABRKSR                       */
unsigned char BYTE;                             /* Byte Access                  */
struct {                                        /* Bit  Access                  */
unsigned char ABIF:1;                           /* ABIF                         */
unsigned char ABIE:1;                           /* ABIE                         */
} BIT;                                          /*                              */
} SR;                                           /*                              */
void *BAR;                                      /* BAR                          */
unsigned int BDR;                               /* BDR                          */
};                                              /*                              */
struct st_io {                                  /* struct IO
    */union {                                   /* PUCR1                        */
unsigned char BYTE;                             /* Byte Access                  */
struct {                                        /* Bit  Access                  */
unsigned char B7:1;                             /* Bit 7                        */
unsigned char B6:1;                             /* Bit 6                        */
unsigned char B5:1;                             /* Bit 5                        */
unsigned char B4:1;                             /* Bit 4                        */
unsigned char   :1;                             /* Bit 3                        */
unsigned char B2:1;                             /* Bit 2                        */
unsigned char B1:1;                             /* Bit 1                        */
unsigned char B0:1;                             /* Bit 0                        */
} BIT;                                          /*                              */
} PUCR1;                                        /*                              */
union {                                         /* PUCR5                        */
unsigned char BYTE;                             /* Byte Access                  */

struct {                                        /* Bit  Access                  */
unsigned char wk:2;                             /* Bit 7,6                      */
unsigned char B5:1;                             /* Bit 5                        */
```

```
unsigned char B4:1;                    /* Bit 4                    */
unsigned char B3:1;                    /* Bit 3                    */
unsigned char B2:1;                    /* Bit 2                    */
unsigned char B1:1;                    /* Bit 1                    */
unsigned char B0:1;                    /* Bit 0                    */
} BIT;                                 /*                          */
} PUCR5;                               /*                          */
char wk1[2];                           /*                          */
union {                                /* PDR1                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char B7:1;                    /* Bit 7                    */
unsigned char B6:1;                    /* Bit 6                    */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
unsigned char   :1;                    /* Bit 3                    */
unsigned char B2:1;                    /* Bit 2                    */
unsigned char B1:1;                    /* Bit 1                    */
unsigned char B0:1;                    /* Bit 0                    */
} BIT;                                 /*                          */
} PDR1;                                /*                          */
union {                                /* PDR2                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char wk:5;                    /* Bit 7-3                  */
unsigned char B2:1;                    /* Bit 2                    */
unsigned char B1:1;                    /* Bit 1                    */
unsigned char B0:1;                    /* Bit 0                    */
} BIT;                                 /*                          */
} PDR2;                                /*                          */
char wk2[2];                           /*                          */
union {                                /* PDR5                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char B7:1;                    /* Bit 7                    */
unsigned char B6:1;                    /* Bit 6                    */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
unsigned char B3:1;                    /* Bit 3                    */
unsigned char B2:1;                    /* Bit 2                    */
```

```
unsigned char B1:1;                    /* Bit 1                    */
unsigned char B0:1;                    /* Bit 0                    */
} BIT;                                 /*                          */
} PDR5;                                /*                          */
char wk3;                              /*                          */

union {                                /* PDR7                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char wk:1;                    /* Bit 7                    */
unsigned char B6:1;                    /* Bit 6                    */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
} BIT;                                 /*                          */
} PDR7;                                /*                          */
union {                                /* PDR8                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char B7:1;                    /* Bit 7                    */
unsigned char B6:1;                    /* Bit 6                    */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
unsigned char B3:1;                    /* Bit 3                    */
} BIT;                                 /*                          */
} PUCR1;                               /*                          */
union {                                /* PUCR5                    */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char wk:2;                    /* Bit 7,6                  */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
unsigned char B3:1;                    /* Bit 3                    */
unsigned char B2:1;                    /* Bit 2                    */
unsigned char B1:1;                    /* Bit 1                    */
unsigned char B0:1;                    /* Bit 0                    */
} BIT;                                 /*                          */
} PUCR5;                               /*                          */
char wk1[2];                           /*                          */
union {                                /* PDR1                     */
unsigned char BYTE;                    /* Byte Access              */
```

```
struct {                                /* Bit  Access                  */
unsigned char B7:1;                     /* Bit 7                        */
unsigned char B6:1;                     /* Bit 6                        */
unsigned char B5:1;                     /* Bit 5                        */
unsigned char B4:1;                     /* Bit 4                        */
unsigned char   :1;                     /* Bit 3                        */
unsigned char B2:1;                     /* Bit 2                        */
unsigned char B1:1;                     /* Bit 1                        */
unsigned char B0:1;                     /* Bit 0                        */
} BIT;                                  /*                              */
} PDR1;                                 /*                              */
union {                                 /* PDR2                         */
unsigned char BYTE;                     /* Byte Access                  */
struct {                                /* Bit  Access                  */
unsigned char wk:5;                     /* Bit 7-3                      */
unsigned char B2:1;                     /* Bit 2                        */
unsigned char B1:1;                     /* Bit 1                        */
unsigned char B0:1;                     /* Bit 0                        */
} BIT;                                  /*                              */
} PDR2;                                 /*                              */

char wk2[2];                            /*                              */
union {                                 /* PDR5                         */
unsigned char BYTE;                     /* Byte Access                  */
struct {                                /* Bit  Access                  */
unsigned char B7:1;                     /* Bit 7                        */
unsigned char B6:1;                     /* Bit 6                        */
unsigned char B5:1;                     /* Bit 5                        */
unsigned char B4:1;                     /* Bit 4                        */
unsigned char B3:1;                     /* Bit 3                        */
unsigned char B2:1;                     /* Bit 2                        */
unsigned char B1:1;                     /* Bit 1                        */
unsigned char B0:1;                     /* Bit 0                        */
} BIT;                                  /*                              */
} PDR5;                                 /*                              */
char wk3;                               /*                              */
union {                                 /* PDR7                         */
unsigned char BYTE;                     /* Byte Access                  */
struct {                                /* Bit  Access                  */
unsigned char wk:1;                     /* Bit 7                        */
```

```
unsigned char B6:1;                    /* Bit 6                    */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
} BIT;                                 /*                          */
} PDR7;                                /*                          */
union {                                /* PDR8                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char B7:1;                    /* Bit 7                    */
unsigned char B6:1;                    /* Bit 6                    */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
unsigned char B3:1;                    /* Bit 3                    */
unsigned char B2:1;                    /* Bit 2                    */
unsigned char B1:1;                    /* Bit 1                    */
unsigned char B0:1;                    /* Bit 0                    */
} BIT;                                 /*                          */
} PDR8;                                /*                          */
char wk4;                              /*                          */
union {                                /* PDRB                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char B7:1;                    /* Bit 7                    */
unsigned char B6:1;                    /* Bit 6                    */
unsigned char B5:1;                    /* Bit 5                    */
unsigned char B4:1;                    /* Bit 4                    */
unsigned char B3:1;                    /* Bit 3                    */
unsigned char B2:1;                    /* Bit 2                    */
unsigned char B1:1;                    /* Bit 1                    */
unsigned char B0:1;                    /* Bit 0                    */
} BIT;                                 /*                          */
} PDRB;                                /*                          */
char wk5[2];                           /*                          */
union {                                /* PMR1                     */
unsigned char BYTE;                    /* Byte Access              */
struct {                               /* Bit  Access              */
unsigned char IRQ3:1;                  /* IRQ3                     */
unsigned char IRQ2:1;                  /* IRQ2                     */
unsigned char IRQ1:1;                  /* IRQ1                     */
unsigned char IRQ0:1;                  /* IRQ0                     */
```

```
unsigned char     :2;                    /*                          */
unsigned char TXD :1;                    /* TXD                      */
unsigned char TMOW:1;                    /* TMOW                     */
} BIT;                                   /*                          */
} PMR1;                                  /*                          */
union {                                  /* PMR5                     */
unsigned char BYTE;                      /* Byte Access              */
struct {                                 /* Bit  Access              */
unsigned char wk  :2;                    /*                          */
unsigned char WKP5:1;                    /* WKP5                     */
unsigned char WKP4:1;                    /* WKP4                     */
unsigned char WKP3:1;                    /* WKP3                     */
unsigned char WKP2:1;                    /* WKP2                     */
unsigned char WKP1:1;                    /* WKP1                     */
unsigned char WKP0:1;                    /* WKP0                     */
} BIT;                                   /*                          */
} PMR5;                                  /*                          */
char wk6[2];                             /*                          */
unsigned char PCR1;                      /* PCR1                     */
unsigned char PCR2;                      /* PCR2                     */
char wk7[2];                             /*                          */
unsigned char PCR5;                      /* PCR5                     */
char wk8;                                /*                          */
unsigned char PCR7;                      /* PCR7                     */
unsigned char PCR8;                      /* PCR8                     */
};                                       /*                          */
union un_syscr1 {                        /* union SYSCR1             */
unsigned char BYTE;                      /* Byte Access              */
struct {                                 /* Bit  Access              */
unsigned char SSBY :1;                   /* SSBY                     */
unsigned char STS  :3;                   /* STS                      */
unsigned char NESEL:1;                   /* NESEL                    */
} BIT;                                   /*                          */
};                                       /*                          */
union un_syscr2 {                        /* union SYSCR2             */
unsigned char BYTE;                      /* Byte Access              */
struct {                                 /* Bit  Access              */
unsigned char SMSEL:1;                   /* SMSEL                    */
unsigned char LSON :1;                   /*    LSON                  */
unsigned char DTON :1;                   /*    DTON                  */
```

```
unsigned char MA   :3;                  /* MA                   */
unsigned char SA   :2;                  /* SA                   */
} BIT;                                  /*                      */
};                                      /*                      */

union un_iegr1 {                        /* union IEGR1          */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char NMIEG:1;                  /* NMIEG                */
unsigned char      :3;                  /*                      */
unsigned char IEG3 :1;                  /* IEG3                 */
unsigned char IEG2 :1;                  /* IEG2                 */
unsigned char IEG1 :1;                  /* IEG1                 */
unsigned char IEG0 :1;                  /* IEG0                 */
} BIT;                                  /*                      */
};                                      /*                      */
union un_iegr2 {                        /* union IEGR2          */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char wk   :2;                  /*                      */
unsigned char WPEG5:1;                  /* WPEG5                */
unsigned char WPEG4:1;                  /* WPEG4                */
unsigned char WPEG3:1;                  /* WPEG3                */
unsigned char WPEG2:1;                  /* WPEG2                */
unsigned char WPEG1:1;                  /* WPEG1                */
unsigned char WPEG0:1;                  /* WPEG0                */
} BIT;                                  /*                      */
};                                      /*                      */
union un_ienr1 {                        /* union IENR1          */
unsigned char BYTE;                     /* Byte Access          */
struct {                                /* Bit  Access          */
unsigned char IENDT:1;                  /* IENDT                */
unsigned char IENTA:1;                  /* IENTA                */
unsigned char IENWP:1;                  /* IENWP                */
unsigned char      :1;                  /*                      */
unsigned char IEN3 :1;                  /* IEN3                 */
unsigned char IEN2 :1;                  /* IEN2                 */
unsigned char IEN1 :1;                  /* IEN1                 */
unsigned char IEN0 :1;                  /* IEN0                 */
} BIT;                                  /*                      */
```

```
};                                          /*                          */
union un_irr1 {                             /* union IRR1               */
unsigned char BYTE;                         /* Byte Access              */
struct {                                    /* Bit  Access              */
unsigned char IRRDT:1;                      /* IRRDT                    */
unsigned char IRRTA:1;                      /* IRRTA                    */
unsigned char      :2;                      /*                          */
unsigned char IRRI3:1;                      /* IRRI3                    */
unsigned char IRRI2:1;                      /* IRRI2                    */
unsigned char IRRI1:1;                      /* IRRI1                    */
unsigned char IRRI0:1;                      /* IRRI0                    */
} BIT;                                      /*                          */
};                                          /*                          */

union un_iwpr {                             /* union IWPR               */
unsigned char BYTE;                         /* Byte Access              */
struct {                                    /* Bit  Access              */
unsigned char wk   :2;                      /*                          */
unsigned char IWPF5:1;                      /* IWPF5                    */
unsigned char IWPF4:1;                      /* IWPF4                    */
unsigned char IWPF3:1;                      /* IWPF3                    */
unsigned char IWPF2:1;                      /* IWPF2                    */
unsigned char IWPF1:1;                      /* IWPF1                    */
unsigned char IWPF0:1;                      /* IWPF0                    */
} BIT;                                      /*                          */
};                                          /*                          */
union un_mstcr1 {                           /* union MSTCR1             */
unsigned char BYTE;                         /* Byte Access              */
struct {                                    /* Bit  Access              */
unsigned char wk    :1;                     /*                          */
unsigned char MSTIIC:1;                     /* MSTIIC                   */
unsigned char MSTS3 :1;                     /* MSTS3                    */
unsigned char MSTAD :1;                     /* MSTAD                    */
unsigned char MSTWD :1;                     /* MSTWD                    */
unsigned char MSTTW :1;                     /* MSTTW                    */
unsigned char MSTTV :1;                     /* MSTTV                    */
unsigned char MSTTA :1;                     /* MSTTA                    */
} BIT;                                      /*                          */
};                                          /*                          */
union un_tscr {                             /* union TSCR               */
```

```
unsigned char BYTE;                                          /* Byte Access                */
struct {                                                     /* Bit  Access                */
unsigned char wk    :6;                                      /*                            */
unsigned char IICRST:1;                                      /* IICRST                     */
unsigned char IICX  :1;                                      /* IICX                       */
} BIT;                                                       /*                            */
};                                                           /*                            */
#define FLASH           (*(volatile struct st_flash *)0xFF90)        /* FLASH Address      */
#define TA              (*(volatile struct st_ta    *)0xFFA6)        /* TA    Address      */
#define TV              (*(volatile struct st_tv    *)0xFFA0)        /* TV    Address      */
#define TW              (*(volatile struct st_tw    *)0xFF80)        /* TW    Address      */
#define SCI3            (*(volatile struct st_sci3  *)0xFFA8)        /* SCI3  Address      */
#define AD              (*(volatile struct st_ad    *)0xFFB0)        /* A/D   Address      */
#define WDT             (*(volatile struct st_wdt   *)0xFFC0)        /* WDT   Address      */
#define IIC             (*(volatile struct st_iic   *)0xFFC4)        /* IIC   Address      */
#define ICDR            EQU.ICE1.UN_ICDR                             /* ICDR  Change       */
#define ICMR            EQU.ICE1.UN_ICMR                             /* ICDR  Change       */
#define SAR             EQU.ICE0.UN_SAR                              /* SAR   Change       */
#define SARX            EQU.ICE0.UN_SARX                             /* SARX  Change       */
#define ABRK            (*(volatile struct st_abrk  *)0xFFC8)        /* ABRK  Address      */
#define IO              (*(volatile struct st_io    *)0xFFD0)        /* IO    Address      */
#define SYSCR1          (*(volatile union  un_syscr1*)0xFFF0)        /* SYSCR1Address      */
#define SYSCR2          (*(volatile union  un_syscr2*)0xFFF1)        /* SYSCR2Address      */


#define IEGR1           (*(volatile union  un_iegr1 *)0xFFF2)        /* IEGR1 Address      */
#define IEGR2           (*(volatile union  un_iegr2 *)0xFFF3)        /* IEGR2 Address      */
#define IENR1           (*(volatile union  un_ienr1 *)0xFFF4)        /* IENR1 Address      */
#define IRR1            (*(volatile union  un_irr1  *)0xFFF6)        /* IRR1  Address      */
#define IWPR            (*(volatile union  un_iwpr  *)0xFFF8)        /* IWPR  Address      */
#define MSTCR1          (*(volatile union  un_mstcr1*)0xFFF9)        /* MSTCR1Address      */
#define TSCR            (*(volatile union  un_tscr  *)0xFFFC)        /* TSCR  Address      */

#define EKR             (*(volatile unsigned char  *)0xFF10)         /* EKR   Address      */
```

7. Program Listing

```
File : INIT.SRC
;
;/*     Assembler routine                                                              */
;
.EXPORT      _INIT
.IMPORT      _main
.SECTION     P,CODE
_INIT:
MOV.W        #H'FF80,R7
LDC.B        #B'10000000,CCR
JMP          @_main
;
.END

File : EPR.c
/*****************************************************************************************************************/
/*                                                                                                               */
/*  FILE                :EPR.c                                                                                   */
/*  DATE                :Jun 29, 2001                                                                            */
/*  DESCRIPTION         :Main Program                                                                            */
/*  CPU TYPE            :H8/3664F                                                                                */
/*                                                                                                               */
/*  This file is generated by Hitachi Project Generator (Ver.1.0)                                               */
/*                                                                                                               */
/*****************************************************************************************************************/
#include <machine.h>
#include "H8_3664_C.H"

extern     void INIT( void );
extern     unsigned char Read_page_EEPROM( unsigned short adrs , unsigned char *rd_ptr );
extern     unsigned char Write_page_EEPROM( unsigned short adrs , unsigned char *wr_ptr );
extern     unsigned char Write_byte_EEPROM( unsigned short adrs , unsigned char wr_data );
extern     unsigned char Read_byte_EEPROM( unsigned short adrs );
extern     unsigned char Read_n_EEPROM( unsigned short adrs, unsigned char *rd_ptr, unsigned short no );

void main ( void );
void wait_ack( unsigned short limit ) ;
void wait ( unsigned short limit );
void Test_EEPROM( void );
```

```
/*;************************************************************************************************/
/*; Vector Address                                                                                 */
/*;************************************************************************************************/
#pragma section          V1                                          /* VECTOR SECTOIN SET          */
void (*const VEC_TBL1[])(void) = {
/* 0x00 - 0x0f */
INIT,                                                                /* 00 RESET                    */
};

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

#pragma section                                                      /* section:B                   */
unsigned char eeprom_buf[512];
unsigned char test_code;
unsigned char dummy;

#pragma section                                                      /* section:P                   */
/*;************************************************************************************************/
/*; Main Program                                                   */
/*;************************************************************************************************/
void main(void)
{
test_code = 1;

while (1) {

Test_EEPROM( );
}
}
```

```
/*;*************************************************************************************************************/
/*; IIC EEPRPM                                                                                                  */
/*  test_code           test job                                                                                 */
/*  0                   nop                                                                                      */
/*  1                   byte write/read                                                                          */
/*  2                   page write 0-ff                                                                          */
/*  3                   page read                                                                                */
/*  4                   page write all"ff"                                                                       */
/*  5                   all read                                                                                 */
/*;*************************************************************************************************************/
void Test_EEPROM( void  ) {
unsigned char           ng_flag,wr_data;
unsigned short          i;

if (test_code == 1) {
i = 0;
while(i < 512)          {
wr_data = (unsigned char)(i & 0xff);
ng_flag = Write_byte_EEPROM(i , wr_data );
eeprom_buf[i] = Read_byte_EEPROM(i);
i++;
}
test_code = 0;
}

else if (test_code == 2) {
for (i = 0;i < 512;i++) {
eeprom_buf[i] = (unsigned char)((i>>1) & 0x00ff);                       /* buf set                      */
}
for (i = 0;i < 512;i+=8) {
ng_flag = Write_page_EEPROM( i , &eeprom_buf[i] );                      /* write:0-511                  */
}
test_code = 0;
}
else if (test_code == 3) {
for (i = 0;i < 512;i+=8) {
ng_flag = Read_page_EEPROM( i , &eeprom_buf[i] );                       /* read:0-511                   */
wait(100);
}
test_code = 0;
```

```
}
else if (test_code == 4) {
for (i = 0;i < 8;i++) {
eeprom_buf[i] = 0xff;                                   /* buf:FF set                   */
}
for (i = 0;i < 512;i+=8) {
ng_flag = Write_page_EEPROM( i , &eeprom_buf[0] );      /* write:0-511                  */
}
test_code = 0;
}
else if (test_code == 5) {
for (i = 0;i < 512;i++) {                               /* buf clear                    */
eeprom_buf[i] = 0x00;
}
ng_flag = Read_n_EEPROM( 0x0000 , &eeprom_buf[0] ,512); /* read:0-511                   */
test_code = 0;
}
}

void wait( unsigned short limit ) {
unsigned int             cnt;

cnt = 0;
while (cnt < limit ) {
cnt++;
}
}

void abort(void)
{

}
```

```
File : IIC_EEPROM.c
/*------------------------------------------------------------------------------------------------------*/
/*  IIC EEPROM Read/Write                                                                               */
/*  Sub routine for H8/3664N built in EEPROM                                                            */
/*------------------------------------------------------------------------------------------------------*/
#include <machine.h>
#include "H8_3664_C.H"

#define DEVICE_CODE     0xa0                                    /* EEPROM DEVICE CODE:101       */
#define SLAVE_ADRS      0x00                                    /* SLAVE ADRS:0                 */
#define IIC_DATA_W      0x00                                    /* WRITE_DATA                   */
#define IIC_DATA_R      0x01                                    /* READ_DAT                     */

#define WR_RETRY_CNT    10                                      /* tWC(1ms)*10=10ms             */
#define WR_OK           11                                      /* loop break                   */


unsigned char           Write_data_EEPROM( unsigned char wr_data );
unsigned char           Set_adrs_EEPROM( unsigned short adrs );
unsigned char           Recv_data1_EEPROM( void );
unsigned char           Recv_datan_EEPROM( unsigned char *rd_ptr , unsigned short no );
void       wait_e( unsigned int loop);

extern     unsigned char dummy;

/*------------------------------------------------------------------------------------------------------*/
/*  Read_page_EEPROM    (8byte)                                                                         */
/*  Argument1: Read address(unsigned short)                                                             */
/*  Argument2: Storing read data address (unsigned char *)                                              */
/*  Return value: 1: OK / 0: NG EEPROM NOACK(unsigned char)                                             */
/*------------------------------------------------------------------------------------------------------*/
unsigned char Read_page_EEPROM( unsigned short adrs , unsigned char *rd_ptr ) {
unsigned char           i,ack;

IIC.ICCR.BYTE = 0x89;                                           /* ICE=1 P57,P56->SCL,SDA       */
IIC.ICMR.BYTE = 0x08;
TSCR.BYTE = 0xfc;

ack = 0;
while (IIC.ICCR.BIT.BBSY != 0)                                  /* Bus busy?                    */
```

```
;

ack = Set_adrs_EEPROM(adrs);                                    /* Addressing(dummy write)       */
if (ack == 0)           {
IIC.ICCR.BIT.IRIC = 0;
IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;                           /* Generate stop condition       */
return(0);
}

ack = Recv_datan_EEPROM(rd_ptr,8);                              /* Data reading of 8 bytes       */

return(ack);
}

/*-------------------------------------------------------------------------------------------------------*/
/*  Read_page_EEPROM    (8byte)                                                                          */
/*  Argument1: Read address (unsigned short)                                                             */
/*  Argument2: Storing read data address (unsigned char *)                                               */
/*  Argument3: Number of read data (unsigned short)                                                      */
/*  Return value: 1: OK / 0: NG EEPROM NOACK(unsigned char)                                              */
/*-------------------------------------------------------------------------------------------------------*/
unsigned char Read_n_EEPROM( unsigned short adrs , unsigned char *rd_ptr , unsigned short no ) {
unsigned char           i,ack;

IIC.ICCR.BYTE = 0x89;                                           /* ICE=1 P57,P56->SCL,SDA        */
IIC.ICMR.BYTE = 0x08;
TSCR.BYTE = 0xfc;

ack = 0;
while (IIC.ICCR.BIT.BBSY != 0)                                  /* Bus busy?                     */
;

ack = Set_adrs_EEPROM(adrs);                                    /* Addressing (dummy write)      */
if (ack == 0) {
IIC.ICCR.BIT.IRIC = 0;
IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;                           /* Generate stop condition       */
return(0);
}

ack = Recv_datan_EEPROM(rd_ptr,no);                             /* Data reading of n bytes       */
```

```
return(ack);
}
/*-----------------------------------------------------------------------------------------------------------*/
/*  Write_page_EEPROM    (8byte)                                                                             */
/*  Argument1: Write address (unsigned short)                                                                */
/*  Argument2: Storing write data address (unsigned char *)                                                  */
/*  Return value: 1: OK / 0: NG EEPROM NOACK(unsigned char)                                                  */
/*-----------------------------------------------------------------------------------------------------------*/
unsigned char Write_page_EEPROM( unsigned short adrs , unsigned char *wr_ptr ) {
unsigned char            loop,i,ack;

ack = 0;
loop = 0;
while (loop <= WR_RETRY_CNT) {
IIC.ICCR.BYTE = 0x89;                                          /* ICE=1(P57,P56->SCL,SDA)       */
IIC.ICMR.BYTE = 0x08;
TSCR.BYTE = 0xfc;

while (IIC.ICCR.BIT.BBSY != 0)                                 /* Bus busy?                     */
;

ack = Set_adrs_EEPROM(adrs);                                   /* Addressing                    */
if ( ack == 1 ) {
loop = WR_OK;
}
else {
IIC.ICCR.BIT.IRIC = 0;
IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;                          /* Generation of stop condition  */
wait_e(1000);                                                  /* tWC=1ms                       */
loop++;
}
}

if (ack == 1) {
IIC.ICCR.BIT.IRIC = 0;
i = 0;
while ((i < 8) && (ack == 1)) {
ack = Write_data_EEPROM(*wr_ptr);                              /* Writing of data of 8 bytes    */
wr_ptr++;
```

```
i++;
}
IIC.ICCR.BIT.IRIC = 0;
IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;                                   /* Generation of stop condition   */
}

return(ack);
}
/*-----------------------------------------------------------------------------------------------------------*/
/*  wait_e (Wait sub routine for EEPROM)                                                                     */
/*  Argument1: Number of loops (unsigned short)                                                              */
/*  Return value: None                                                                                       */
/*-----------------------------------------------------------------------------------------------------------*/
void wait_e( unsigned int loop) {
unsigned int            i;

for (i=0;i < loop;i++) {
dummy++;
}
}

/*-----------------------------------------------------------------------------------------------------------*/
/*  Read_byte_EEPROM                                                                                         */
/*  Argument1: Read address (unsigned short)                                                                 */
/*  Return value: Read data (unsigned char)                                                                  */
/*-----------------------------------------------------------------------------------------------------------*/
unsigned char Read_byte_EEPROM( unsigned short adrs ) {
unsigned char           data,ack;

IIC.ICCR.BYTE = 0x89;                                                   /* ICE=1 P57,P56->SCL,SDA        */
IIC.ICMR.BYTE = 0x08;
TSCR.BYTE = 0xfc;

while (IIC.ICCR.BIT.BBSY != 0)                                          /* Bus busy?                     */
;

ack = Set_adrs_EEPROM(adrs);                                            /* Addressing (dummy write)      */

if (ack == 1)           {
data = Recv_data1_EEPROM();
```

```
}

IIC.ICCR.BIT.IRIC = 0;
IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;                                   /* Generate stop condition        */

return(data);
}
/*-------------------------------------------------------------------------------------------------------*/
/*  Write_byte_EEPROM                                                                                    */
/*  Argument1: Write address (unsigned short)                                                            */
/*  Argument2: Write data (unsigned char)                                                                */
/*  Return value: 1: OK / 0: NG EEPROM NOACK(unsigned char)                                              */
/*-------------------------------------------------------------------------------------------------------*/
unsigned char Write_byte_EEPROM( unsigned short adrs , unsigned char wr_data ) {
unsigned char            loop,ack;

ack = 0;
loop = 0;
while (loop <= WR_RETRY_CNT) {

IIC.ICCR.BYTE = 0x89;                                                   /* ICE=1(P57,P56->SCL,SDA)       */
IIC.ICMR.BYTE = 0x08;
TSCR.BYTE = 0xfc;

while (IIC.ICCR.BIT.BBSY != 0)                                          /* Bus busy?                     */
;

ack = Set_adrs_EEPROM(adrs);
if ( ack == 1 ) {
loop = WR_OK;
}

else {
IIC.ICCR.BIT.IRIC = 0;
IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;                                   /* Generate stop condition        */
wait_e(1000);                                                           /* tWC=1ms                       */
loop++;
}
}
```

```
if (ack == 1) {
ack = Write_data_EEPROM(wr_data);
IIC.ICCR.BIT.IRIC = 0;
IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;                                   /* Generate stop condition       */
}

return(ack);
}

/*-----------------------------------------------------------------------------------------------------*/
/*  Write_data_EEPROM                                                                                  */
/*  Argument1: Write address (unsigned short)                                                          */
/*  Return value: 1: OK / 0: NG EEPROM NOACK(unsigned char)                                            */
/*-----------------------------------------------------------------------------------------------------*/
/*-----------------------------------------------------------------------------------------------------*/
/*  (BYTE WRITE ACTION)                                                                                */
/*                          <4>                                                                        */
/*                          123456789                                                                  */
/*                          000000000                                                                  */
/*                          Write data A                                                               */
/*-----------------------------------------------------------------------------------------------------*/
unsigned char Write_data_EEPROM( unsigned char wr_data ) {

                                                                        /* <4>                          */
IIC.ICDR = wr_data;                                                     /* <4> Set writing data         */
IIC.ICCR.BIT.IRIC = 0;
while (IIC.ICCR.BIT.IRIC == 0)                                          /* <4> Transmission complete?   */
;
if ( IIC.ICSR.BIT.ACKB != 0 ) {                                         /* ACK?                         */
return(0);
}

return(1);
}

/*-----------------------------------------------------------------------------------------------------*/
/*  Set_adrs_EEPROM                                                                                    */
/*  Argument1: Write / read address (unsigned short)                                                   */
/*  Return value: 1: OK / 0: NG EEPROM NOACK(unsigned char)                                            */
/*-----------------------------------------------------------------------------------------------------*/
```

```
/*-----------------------------------------------------------------------------------------------------*/
/*  ADDRESS SET ACTION / DUMMY WRITE ACTION)                                                           */
/*  <1>                 <2>          <3>                                                               */
/*  123456789           123456789    123456789                                                         */
/*  101000000           000000000    000000000                                                         */
/*  Slave WA            Address HI A Address LO A                                                      */
/*-----------------------------------------------------------------------------------------------------*/
unsigned char Set_adrs_EEPROM( unsigned short adrs ) {
unsigned char           ret;

IIC.ICCR.BYTE |= 0x30;                              /* Master transmission setting (MST=1,TRS=1)       */
IIC.ICCR.BYTE = ((IIC.ICCR.BYTE & 0xfe) | 0x04);
                                                    /* Generating of start condition                   */
while (IIC.ICCR.BIT.IRIC == 0)                      /* Transmission OK?                                */
;
                                                    /* <1>                                             */
IIC.ICDR = (unsigned char)(DEVICE_CODE | SLAVE_ADRS | IIC_DATA_W);
                                                    /* <1>Slave address set                            */
IIC.ICCR.BIT.IRIC = 0;

while (IIC.ICCR.BIT.IRIC == 0)                      /* <1> Transmission complete?                      */
;

if ( IIC.ICSR.BIT.ACKB != 0 ) {                     /* ACK?                                            */
return(0);
}
                                                    /* <2>                                             */
IIC.ICDR = (unsigned char)(adrs >> 8);              /* <2> Upper address set                           */
IIC.ICCR.BIT.IRIC = 0;
while (IIC.ICCR.BIT.IRIC == 0)                      /* <2> Transmission complete?                      */
;
if ( IIC.ICSR.BIT.ACKB != 0 ) {                     /* ACK?              */
    return(0);
}
                                                    /* <3>                                             */
IIC.ICDR = (unsigned char)(adrs & 0x00ff);          /* <3> Lower address set                           */
IIC.ICCR.BIT.IRIC = 0;
while (IIC.ICCR.BIT.IRIC == 0)                      /* <3> Transmission complete?                      */
;
```

```
if ( IIC.ICSR.BIT.ACKB != 0 ) {                                        /* ACK?                          */
return(0);
}

return(1);
}

/*------------------------------------------------------------------------------------------------------*/
/*  Recv_data1_EEPROM                                                                                   */
/*  Return value: Read data (unsigned char)                                                             */
/*------------------------------------------------------------------------------------------------------*/
/*------------------------------------------------------------------------------------------------------*/
/*  (CURRENT ADDRESS READ ACTION)                                                                       */
/*  <1>                 <2>                                                                             */
/*  123456789           123456789                                                                       */
/*  101000010           000000000                                                                       */
/*  Slave RA            Read data A                                                                     */
/*------------------------------------------------------------------------------------------------------*/
unsigned char Recv_data1_EEPROM( void ) {
unsigned char           recv;

IIC.ICCR.BYTE |= 0x30;                              /* Master transmission setting (MST=1,TRS=1)       */
IIC.ICCR.BYTE = ((IIC.ICCR.BYTE & 0xfe) | 0x04);
                                                    /* Generating of start condition                   */
while (IIC.ICCR.BIT.IRIC == 0)                      /* Transmission OK?                                */
;

IIC.ICDR = (unsigned char)(DEVICE_CODE | SLAVE_ADRS | IIC_DATA_R);
                                                    /* <1> Slave address set                           */
IIC.ICCR.BIT.IRIC = 0;

while (IIC.ICCR.BIT.IRIC == 0)                      /* <1> Transmission complete?                      */
;
if ( IIC.ICSR.BIT.ACKB != 0 ) {                     /* ACK?                                            */
return(0);
}

wait_e(30);                                         /* dummy wait                                      */
                                                    /* Master reception setting                        */
IIC.ICCR.BIT.TRS = 0;
```

```
IIC.ICMR.BIT.WAIT = 1;
IIC.ICSR.BIT.ACKB = 0;

recv = IIC.ICDR;                                                    /* dummy read                    */
IIC.ICCR.BIT.IRIC = 0;

while (IIC.ICCR.BIT.IRIC == 0)                                      /* Reception complete?           */
;

                                                                    /* < The last reception >        */
IIC.ICSR.BIT.ACKB = 1;
IIC.ICCR.BIT.TRS = 1;
IIC.ICCR.BIT.IRIC = 0;
while (IIC.ICCR.BIT.IRIC == 0)                                      /* Reception complete?           */
;

IIC.ICMR.BIT.WAIT = 0;

recv = IIC.ICDR;                                                    /* Received data read            */

return(recv);
}
/*-------------------------------------------------------------------------------------------------------*/
/*  Recv_datan_EEPROM                                                                                    */
/*  Argument1: Storing read data address (unsigned char *)                                               */
/*  Argument2: Read byte number1-512 (unsigned char)                                                     */
/*  Return value: 1: OK / 0: NG EEPROM NOACK (unsigned char)                                             */
/*-------------------------------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------------------------------------*/
/*  (CURRENT ADDRESS READ ACTION)                                                                        */
/*  <1>                 <2>          <3>          <n>                                                    */
/*  123456789           123456789    123456789    123456789                                              */
/*  101000010           000000000    000000000    000000000                                              */
/*  Slave RA            Read data A  Read data A   Read data A                                           */
/*-------------------------------------------------------------------------------------------------------*/
unsigned char Recv_datan_EEPROM( unsigned char *rd_ptr , unsigned short no ) {
unsigned char           recv;

IIC.ICCR.BYTE |= 0x30;                                          /* Master transmission setting (MST=1,TRS=1) */
IIC.ICCR.BYTE = ((IIC.ICCR.BYTE & 0xfe) | 0x04);
```

```
                                                /* Generating of start condition                    */
while (IIC.ICCR.BIT.IRIC == 0)                  /* Transmission OK?                                 */
;

IIC.ICDR = (unsigned char)(DEVICE_CODE | SLAVE_ADRS | IIC_DATA_R);
                                                /* <1> Slave address set                            */
IIC.ICCR.BIT.IRIC = 0;

while (IIC.ICCR.BIT.IRIC == 0)                  /* <1> Transmission complete?                       */
;

if ( IIC.ICSR.BIT.ACKB != 0 ) {                 /* ACK?                                             */
return(0);
}

wait_e(30);                                     /* dummy wait                                       */
                                                /* Master reception setting                         */
IIC.ICCR.BIT.TRS = 0;
IIC.ICMR.BIT.WAIT = 1;
IIC.ICSR.BIT.ACKB = 0;

recv = IIC.ICDR;                                /* dummy read                                       */
IIC.ICCR.BIT.IRIC = 0;
while (IIC.ICCR.BIT.IRIC == 0)                  /* Reception complete?                              */
;

IIC.ICCR.BIT.IRIC = 0;

while (1 < no) {
while (IIC.ICCR.BIT.IRIC == 0)                  /* Reception complete?                              */
;

*rd_ptr = IIC.ICDR;                             /* Received data read                               */

IIC.ICCR.BIT.IRIC = 0;
while (IIC.ICCR.BIT.IRIC == 0)                  /* Reception complete?                              */
;

rd_ptr++;                                       /* Storage address increment                        */
no--;
```

```
if (1 < no) {
IIC.ICCR.BIT.IRIC = 0;
}
}

                                                /* < The last reception >                              */
IIC.ICSR.BIT.ACKB = 1;
IIC.ICCR.BIT.TRS = 1;

IIC.ICCR.BIT.IRIC = 0;
while (IIC.ICCR.BIT.IRIC == 0)                  /* Reception complete?                                 */
;

IIC.ICMR.BIT.WAIT = 0;

*rd_ptr = IIC.ICDR;                             /* Received data read                                  */

IIC.ICCR.BIT.IRIC = 0;

IIC.ICCR.BYTE = IIC.ICCR.BYTE & 0xfa;           /* Generation of stop condition                        */

return(1);
}

/*-----------------------------------------------------------------------------------------------------*/
/*  IIC Register function                                                                              */
/*-----------------------------------------------------------------------------------------------------*/
/*-----------------------------------------------------------------------------------------------------*/
/* ICSR  7    6    5    4    3     2      1      0                                                      */
/*   ESTP STOP IRTR AASX AL   AAS   ADZ    ACKB                                                        */
/*-----------------------------------------------------------------------------------------------------*/
/*-----------------------------------------------------------------------------------------------------*/
/* ICCR  7    6    5    4    3     2      1      0                                                      */
/*       ICE  IEIC MST  TRS  ACKE  BBSY   IRIC   SCP                                                  */
/* 0x89  1    0    0    0    1     0      0      1                                                      */
/*       SCLSDA                 ACK Enable                                                           */
/*-----------------------------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------------*/
/* ICMR  7   6   5   4   3    2     1     0                                                              */
/*       MLS WAIT CKS2 CKS1 CKS0  BC2   BC1   BC0                                                        */
/* 0x08  0   0   0   0   1    0     0     0                                                              */
/*               CKS = 400kHz         format = 9bit                                                    */
/*-------------------------------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------------------------------------*/
/* TCSR  7   6   5   4   3    2     1     0                                                              */
/*       1   1   1   1   1    1     IICRST IICX                                                          */
/* 0xfc  1   1   1   1   1    1     0     0                                                              */
/*-------------------------------------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------------------------------------*/
/*  EEPROM Read/Write End                                          */
/*-------------------------------------------------------------------------------------------------------*/
```