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

I²C Bus Interface

Application Note

Renesas Single-Chip Microcomputer

Renesas Electronics
www.renesas.com

Rev.2.0 2001.11

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.

Contents

Section 1 Overview of the I²C Bus 1
1.1 Overview of the I²C Bus 1
1.1.1 Features of the I²C Bus 1
1.1.2 Differences with the Serial Communications Interface (SCI) 1
1.1.3 Connection Type of the I²C bus Interface 2
1.2 Method of Data Transfer over an I²C Bus 3
1.2.1 Basic Concepts and Elements of Data Transfer over an I²C Bus 3
1.2.2 Procedure for Data Transfer (Example: master transmission, slave reception) 6
1.3 The Single-Master and Multi-Master Configurations 8
1.3.1 Single-Master 8
1.3.2 Multi-Master 8
1.4 Procedure for Adjusting Communications 9

Section 2 Explanation of the Interface Functions of the I²C Bus 11
2.1 Lineup of Products that Incorporate the I²C Bus Interface 11
2.2 Specifications of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products [H8 Series] 13
2.2.1 Specifications of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products 13
2.2.2 Configuration of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products 14
2.2.3 Data Transfer Format of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products 15
2.2.4 Explanation of Functions of the Registers of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products 17
2.3 Specifications of the I²C Bus Interfaces Incorporated in H8S Series Products 19
2.3.1 Features of the I²C Bus Interfaces Incorporated in H8S Series Products 19
2.3.2 Internal Block Configuration of the H8S Series I²C Bus Interface 21
2.3.3 Data Format for the H8S Series I²C Bus 22
2.3.4 Description of Functions of the H8S Series I²C Bus Interface Incorporated Registers 24
2.3.5 Relationship between Flags of On-chip I²C Bus Interface and Transfer State in H8S Series (H8S/2138 Series) 42
2.4 Description of I²C Bus Interface Usage 43
2.5 Synchronization of the I²C Bus Communication 54
2.6 Description of Data Transfer in H8/300 and H8/300L Series [H8 Series] 56
2.6.1 Master transmission 56
2.6.2 Master Reception 58
2.6.3 Slave Reception 60

2.6.4 Slave Transmission 63
2.7 Description of Data Transfer in H8S Series (H8/2138 Series) [H8S Series] 65
2.7.1 Master Transmission 65
2.7.2 Master Reception 70
2.7.3 Slave Reception 75
2.7.4 Slave Transmission 78

Section 3 Examples of Application to the H8/300 and H8/300L Series 83
3.1 System Specifications 83
3.2 Circuit for Multi-Master Evaluation System 87
3.3 Design of Software 88
3.3.1 Description of Modules 88
3.3.2 Master 88
3.3.3 Slave 90
3.4 Flowcharts 92
3.4.1 Master Program 92
3.4.2 Slave Program 95
3.5 Program Listings 98
3.5.1 Master Program 98
3.5.2 Slave Program 105

Section 4 Example Applications for the H8S Series 113
4.1 Usage Guide to the Example Applications for the H8S Series 113
4.1.1 The Structure of the Example Applications for the H8S Series 113
4.1.2 Description of the Definition File for the Vector Table 114
4.1.3 Description of the Definition File for the Registers 118
4.1.4 Description of the Inclusion of Assembler Files in C Language Programs 148
4.1.5 Description of the Linkage of Files 149
4.2 Single-Master Transmission 150
4.2.1 Specification 150
4.2.2 Description of the Operation 152
4.2.3 Description of the Software 153
4.2.4 Flowchart 157
4.2.5 Program List 162
4.3 Single-Master Reception 167
4.3.1 Specifications 167
4.3.2 Operation Descriptions 169
4.3.3 Software Descriptions 172
4.3.4 Flowchart 175
4.3.5 Program List 183
4.4 One-Byte Data Transmission by Single-Master Transmission 189
4.4.1 Specifications 189
4.4.2 Operation Descriptions 191

4.4.3 Software Descriptions 192
4.4.4 Flowchart 195
4.4.5 Program List 200
4.5 One-Byte Data Reception by Single-Master Reception 204
4.5.1 Specifications 204
4.5.2 Operation Description 205
4.5.3 Software Description 207
4.5.4 Flowchart 210
4.5.5 Program List 217
4.6 Single-Master Transmission by DTC 222
4.6.1 Specifications 222
4.6.2 Operation Description 229
4.6.3 Software Description 229
4.6.4 Flowchart 235
4.6.5 Program List 240
4.7 Single-Master Reception by DTC 245
4.7.1 Specifications 245
4.7.2 Description of Operation 252
4.7.3 Description of Software 253
4.7.4 Flowchart 258
4.7.5 Program List 266
4.8 Slave Transmission 272
4.8.1 Specifications 272
4.8.2 Description of Operation 273
4.8.3 Description of Software 275
4.8.4 Flowcharts 279
4.8.5 Program List 282
4.9 Slave Reception 286
4.9.1 Specifications 286
4.9.2 Description of Operation 288
4.9.3 Description of Software 289
4.9.4 Flowcharts 293
4.9.5 Program List 297
4.10 Example of Processing Bus Disconnection 301
4.10.1 Specification 301
4.10.2 Description of Operation 303
4.10.3 Description of Software 304
4.10.4 Flowcharts 308
4.10.5 Program List 317
4.11 Bus Conflict 324
4.11.1 Specifications 324
4.11.2 Operation Description 330
4.11.3 Description of Software 331

4.11.4 Flowchart..... 335
4.11.5 Master-1 program List..... 344
4.11.6 Master-2 program List..... 351

Introduction

In recent times, the peripheral interfaces for all fields of application have been being unified and standardized because of the need for lower costs and greater utility. The I^2C bus* interface covered by this application note is one such standardized interface. It is for use as an interface with the control ICs of home appliances, and in controlling the battery packs of notebook-sized PCs, PC monitors, etc.

The I^2C bus is the standardized form of a bi-directional serial bus system which was developed by Philips in the Netherlands. In products based on this standard, two wires (a clock line and data line) are used to carry mutual data communications among multiple peripheral ICs.

The I^2C bus interfaces incorporated in Hitachi's 8-bit/16-bit H8/300-series, H8/300L-series, and H8S-series single-chip microcomputers are an implementation of a sub-set of the standard functions and conform to the I^2C bus interface method proposed by Philips, Ltd. (that is, note that some specifications of the I^2C bus interface are not completely implemented depending on the condition used).

In sections 1 and 2 of this application note, an outline of the I^2C bus is given and the specifications and functions of our I^2C bus-interface module are described. Examples of systems in multi-master configurations are introduced in section 3 and examples of the application of the I^2C bus interface with H8S-series products are given in section 4.

The operation of the examples of hardware and software described in this application note has been confirmed. However, when they are actually used, be sure to base this usage on a confirmation of their operation.

Note: * I^2C Bus: Inter-IC Bus

Section 1 Overview of the I²C Bus

1.1 Overview of the I²C Bus

1.1.1 Features of the I²C Bus

Features of the I²C bus are shown below.

- An I²C bus is made up of two bus lines; a serial data line (SDA) and a serial clock line (SCL). It is easy to extend an I²C bus so that it serves more devices.
- In the I²C bus, the master-slave relationships among devices is always set up and each device has a particular address. Specifying the particular address of the object of the communication forms a path along which data communications is enabled.
- Any device is able to act as a master (i.e., construction of a multi-master system is possible). A system to avoid competition for bus rights and thus prevent the loss of data has thus been defined for the I²C bus interface.
- The maximum data transfer rates are 100 kbps in normal mode and 400 kbps in high-speed mode (up to 3.4 Mbps is defined in version 2.0 of the I²C bus specification).
- The limit on the attachment of devices to an I²C bus system is defined as 400 pF, which is the upper limit of the bus-load capacity of the system.
- Examples of the standard's application are the SMBus*1 and Access.bus*2.

Notes: *1 SMBus is a form of serial bus devised by Duracell and Intel.
*2 ACCESS.bus is a form of serial bus devised by Digital Equipment.

1.1.2 Differences with the Serial Communications Interface (SCI)

Hitachi's serial interface is referred to as the serial communications interface (SCI). The differences between this interface and the standard I²C interface are listed in the table below.

As listed in table 1.1, an SCI is connected to two data lines, one for transmission and one for reception. Data communications is generally on a one-to-one basis.

On the other hand, communications on an I²C bus are bi-directional over a single data line by the equipment to a master. An object is selected for a communication by specifying that object's particular address. This allows the transmission and reception of data between any pair among multiple connected devices. The mechanism for avoiding conflicts over bus access that has been defined for the I²C bus means that the bus supports the operation of multi-master systems, in which any device is able to act as the master. The maximum transfer rates are 100 kbps in normal mode and 400 kbps in high-speed mode.

Table 1.1 Differences from SCI

	SCI		I^2C bus
	Clock synchronous	**Asynchronous**	
Used pins	Three-line method	Two-line method	Two-line method
	Transmission data output	Transmission data output	Transmission/reception data (input/output)
	Reception data input	Reception data input	
	Serial clock	Serial clock (when an external clock is used)	Serial clock
Transfer rate	100 bps to 4 Mbps	110 bps to 38.4 kpbs	100 kbps (normal mode) 400 kbps (high-speed mode)*
Transmission/reception with multiple ICs	Impossible	Impossible	Possible; slave devices have individual addresses

Note: * Hs mode (maximum transfer speed: 3.4 Mbps) which is defined in the I^2C Bus Specifications Ver. 2.0 is not supported.

1.1.3 Connection Type of the I^2C bus Interface

Figure 1.1 shows the form of a connection between I^2C bus interfaces. As shown in the drawing, the I^2C bus is made up of clock line SCL and data line SDA, and they are connected to the power source of the bus, VBB, via pull-up resistors. The SCL and SDA pins of devices 1 and 2 have wired-AND connections with the SCL and SDA lines, respectively.

In the figure, device 2 has been monitoring the state of the SCL line and thus confirms that another device is using the bus when device 1 drives the SCL line low. Furthermore, even while device 1 is using the bus and thus driving the SCL line, device 2 is able to drive SCL low and place the device 1 in its wait state, in terms of communications operations (for details, see the I^2C bus specification).

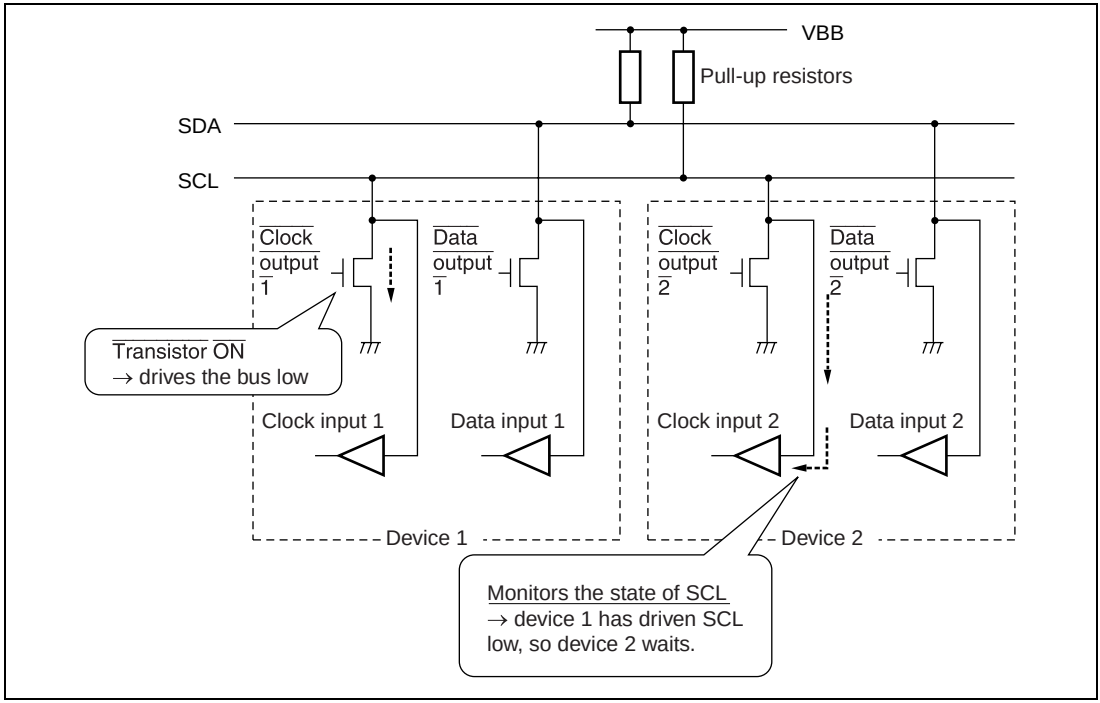


Figure 1.1 Form of a Connection between I²C Bus Interfaces (when device 1 initiates the connection by driving SCL low)

1.2 Method of Data Transfer over an I²C Bus

1.2.1 Basic Concepts and Elements of Data Transfer over an I²C Bus

To start with, the basic concepts and elements of data transfer over an I²C bus are given below.

(1) Master device

The master device generates the clock signals that synchronize data communications and sets the start and stop conditions that indicate the beginning and end of each data communication.

(2) Slave device

The slave device is a device other than a master device which is on the I²C bus.

(3) Transmission device

The transmission device is a device which is transmitting data. It may be a master device or a slave device.

(4) Reception device

The reception device is a device which is receiving data. It may be a master device or a slave device.

(5) Start condition

The start condition is set by changing the level on the SDA line from high to low while the SCL line is high. This is shown in figure 1.2. A data communication is initiated by this operation. The start condition is set by the master device.

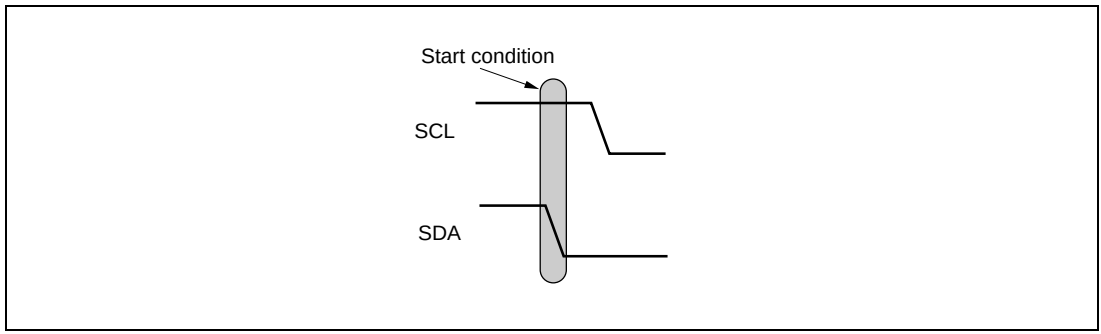


Figure 1.2 Start Condition

(6) Stop condition

The stop condition is set by changing the level on the SDA line from low to high while the SCL line is high. This is shown in figure 1.3. A data communication is stopped by this operation. The stop condition is set by the master device.

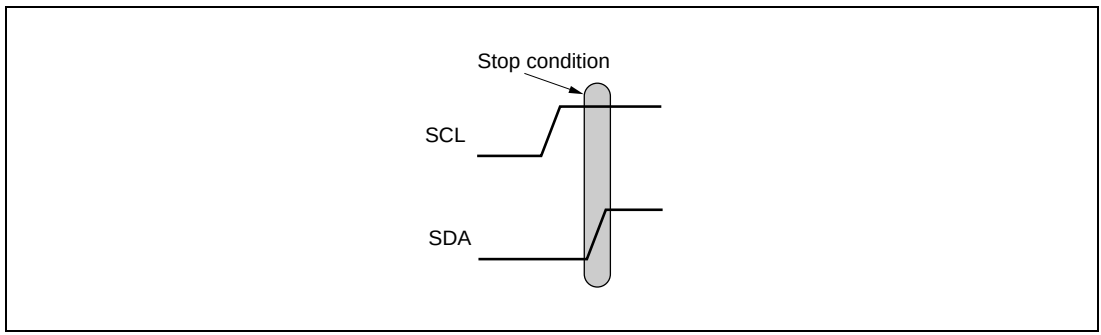


Figure 1.3 Stop Condition

(7) Output timing of the data

Figure 1.4 shows the timing of data output. The data on the SDA line is updated while the SCL line is low and the data on the SDA line is settled for placement on the SDA line while the SCL line is high. The signal on the SDA line only changes while the SCL line is high, that is, only from the setting of the start condition to the setting of the stop condition.

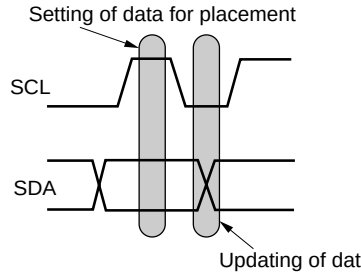


Figure 1.4 Timing of Data Output

(8) Master transmission

Master transmission is the activity when a master device is a transmission device. This is the activity when a slave address is transmitted after the start condition has been issued or a command is transmitted to the slave device, etc.

(9) Master reception

Master reception is the activity when a master device is a reception device.

(10) Slave transmission

Slave transmission is the activity when a slave device is a transmission device.

(11) Slave reception

Slave reception is the activity when a slave device is a reception device. A master device transmits a slave address after the start condition is in place to initiate slave-reception activity in the selected slave device.

(12) Bus-released state

This is the state in which no I^2C bus devices are in communication. While this state applies, both the SCL and SDA lines stay at the logic-high level.

(13) Bus-occupied state

This is the state in which something is communicated over the I^2C bus device. The system returns to the bus-released state after the transmission master device has set a stop condition.

(14) Format for data transfer

Figure 1.5 shows the format for the transfer of data over the I^2C bus. The start and stop condition signals and the SCL clock are generated by the master device. The first data after the start

condition carry the slave address. The eighth bit indicates the direction of communication. A zero value for this bit indicates that the subsequent data is transmitted from a master device while a one indicates that the communication after the second byte is for reception by a master device. The slave address is defined by 7 bits[*1], and is set between B'0000000 and H'1111111 by the user. However, address B'0000000 (referred to as the general call address) and certain other addresses are reserved.

Data is transferred in 1-byte (8-bit) units. The ninth bit is an acknowledge bit from the reception device. For example, when a slave address is transmitted from the master device, the corresponding slave device drives SDA low on the ninth clock cycle to return an acknowledgement to the master.

There is no limit on the number of bytes of data that can be transferred between the setting of a start condition and of the corresponding stop condition. A communication is completed when the stop condition is set.

Notes: *1 The I^2C bus specification describes 10-bit addresses. Hitachi's I^2C bus interface module does not support this 10-bit address specification.

*2 The general call address, B'0000000, is used to specify all slave addresses that are connected to the bus.

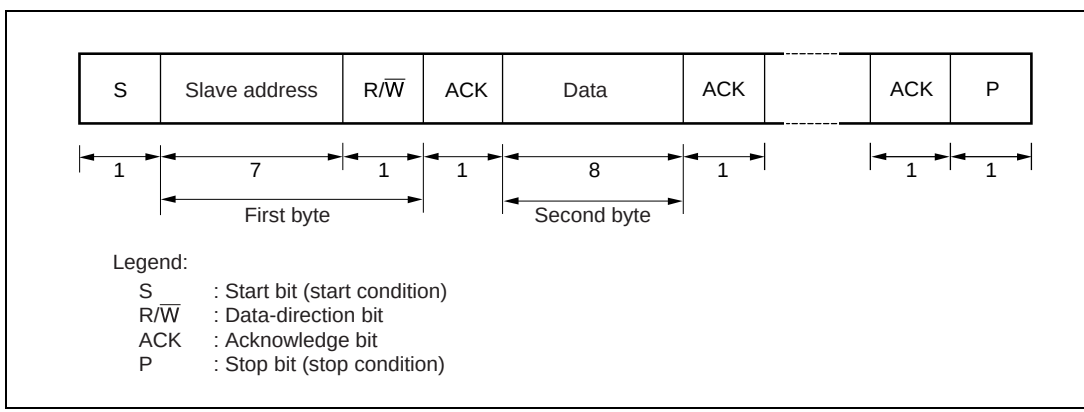


Figure 1.5 Format for Data Transfer

1.2.2 Procedure for Data Transfer (Example: master transmission, slave reception)

Figure 1.6 shows an example when the master device transmits 1 byte of data to the slave device.

In the first place, the master device sets the start condition by changing the level on the SDA line from high to low while the SCL line is high. Next, the master outputs a clock signal on the SCL line and outputs, on the SDA line, the address of the slave that will be the target of this communication. The address of the slave is defined by 7 bits. A bit to indicate the direction of the communication is added as an eighth bit.

The master device releases the SDA line in the ninth clock cycle so that it is able to receive an acknowledgement of selection from the slave device. The selected slave device drives the SDA line low during this clock cycle to return the acknowledgement.

The master device receives the acknowledgement from the slave at the specified address and keeps the SCL line low until the first byte of data is ready for transmission. When the first byte is ready, the master device outputs the data on the SDA line while outputting a clock signal on the SCL line. In the same way as for the slave address, the selected slave device returns an acknowledgement to the master device in the ninth clock cycle. This signal acknowledges that the slave device has received the data without problems.

The master device keeps the SCL line low while receiving this acknowledgement from the slave device. To set the stop condition, the level on the SDA line is then changed from low to high while the SCL line is high.

During the transmission of data, the slave device may become unable to receive the data because it is busy with some other processing. In this case, the slave device keeps the SCL line at its low level so that the master device stays in its wait state. The timing with which the slave device is able to drive SCL low is at the same time as the master device is driving SCL low.

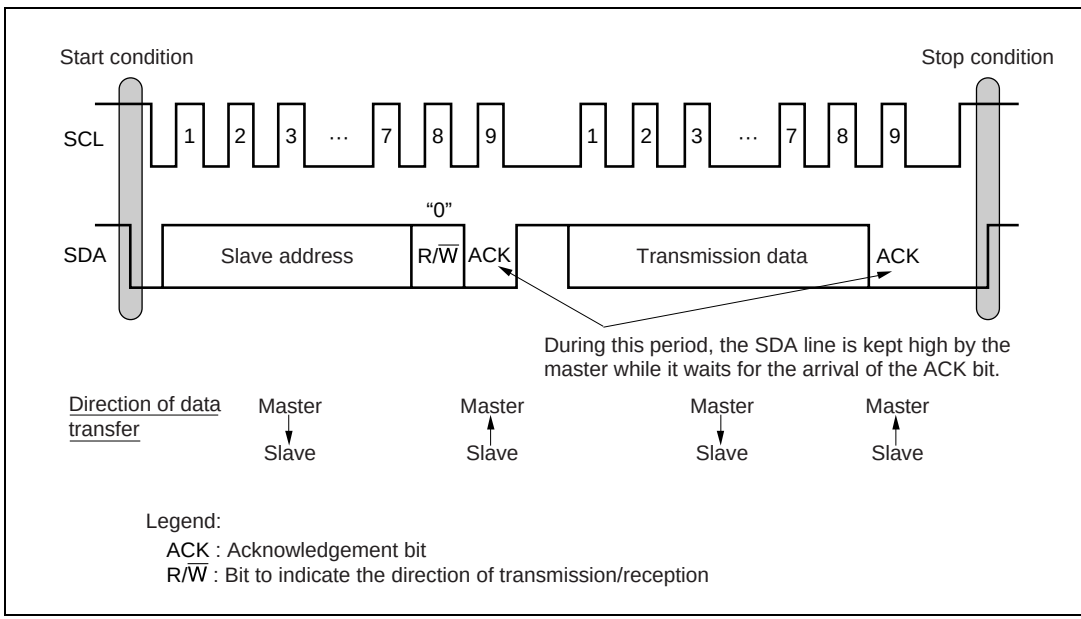


Figure 1.6 Format for Data Transfer (Master Transmission, Slave Reception)

1.3 The Single-Master and Multi-Master Configurations

1.3.1 Single-Master

The master device sets start and stop conditions to control data communications. It also outputs the synchronizing clock signal on the SCL line and slave addresses so that data can be transmitted and received. The system configuration shown in figure 1.7, in which a set device is always the master, is a single-master configuration.

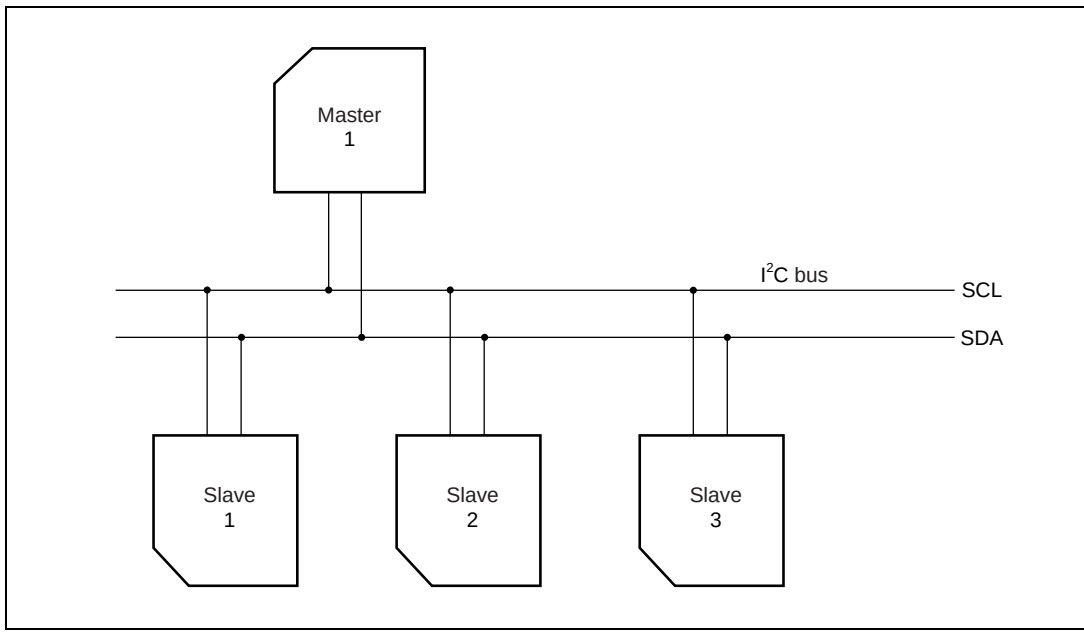


Figure 1.7 A Single-Master Configuration

1.3.2 Multi-Master

A configuration in which two or more devices are included as masters in one system is called a multi-master configuration.

The master device is only able to start the transfer of data after the bus has been released. However, in the multi-master configuration, multiple master devices may simultaneously attempt to start to transfer data. There is then a conflict over bus rights. The specifications of the I^2C bus thus include a procedure for adjusting communications when there is a conflict over bus rights. For details, see 1.4, Procedure for Adjusting Communications.

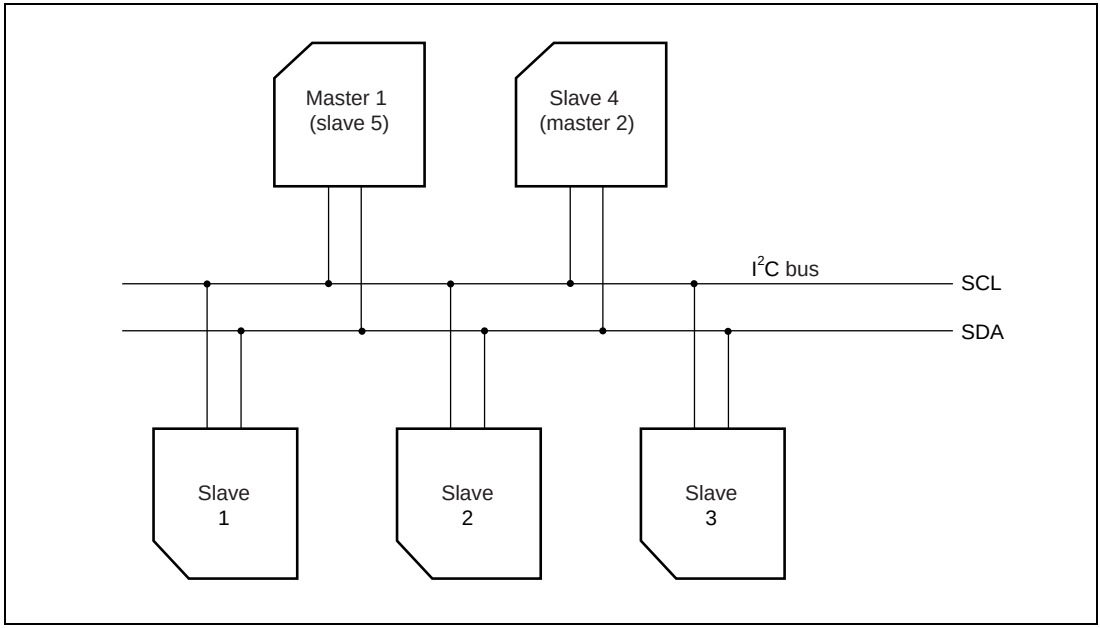


Figure 1.8 A Multi-Master Configuration

1.4 Procedure for Adjusting Communications

The specification of the I^{2}C bus interface includes a procedure for adjusting communications to prevent conflicts over bus rights. This supports systems in multi-task configurations.

Master devices monitor the bus line to confirm that the bus has been released before they set the start condition. When the bus is released, multiple master devices may attempt to set the start condition. A single valid master device is thus defined by the procedure shown in figure 1.9.

In the I^{2}C bus, the data is settled for placement on the SDA line while the SCL line is at its high level. Therefore, each device monitors for the rising edge of the SCL line after the start condition has been set and compares the state of the SDA line with the bit of data that each device is attempting to send (this initial data will be the slave address). If device 1 is driving SDA high while device 2 is driving SDA low, the actual SDA line will be low because of the wired-AND connection, so device 1 confirms that this differs from the bit which is attempting to output. Device 1 then switches the data output stage off. In this example, device 2 continues its operation as a master device (see figure 1.9). When all masters are trying to specify the address of the same slave device, the operation will proceed to the next step and the first bit of data will be compared, and so on.

For example, when the data to be transferred transfer data are H'01 and H'02 as shown in figure1.10, the datum H'01 is low over a longer period, and its transmission thus continues to be enabled. In the same way, the general call address (H'00) has the highest priority.

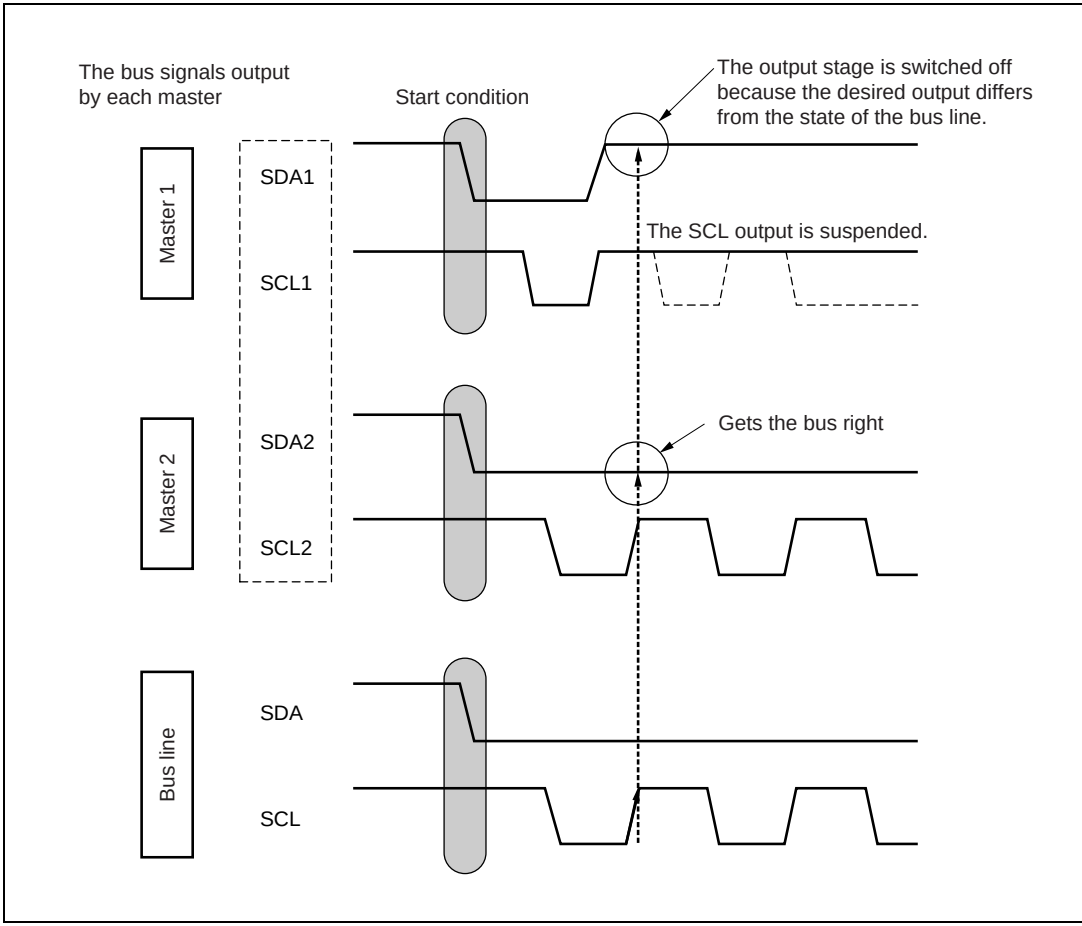


Figure 1.9 Procedure for Adjusting Communications (Detection of the Loss of Bus Arbitration)

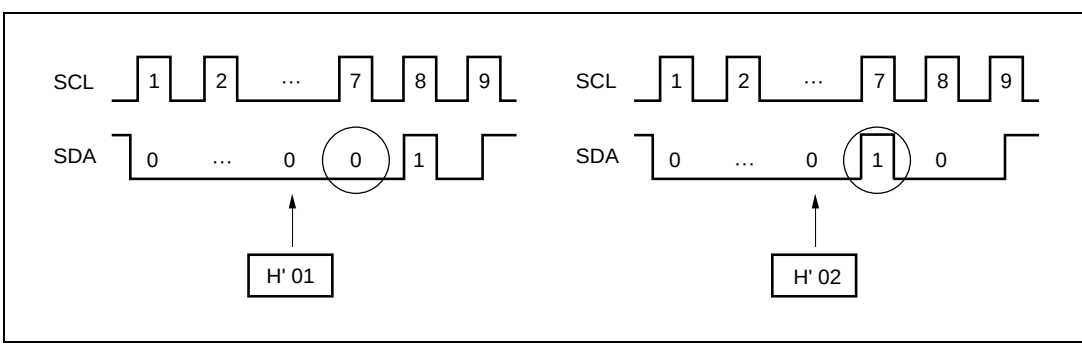


Figure 1.10 A Specific Example of the Adjustment of Communications

Section 2 Explanation of the Interface Functions of the I²C Bus

2.1 Lineup of Products that Incorporate the I²C Bus Interface

Our I²C bus interface modules may be roughly classified into two groups.

(1) H8 family: The models which feature the first I²C bus interface module to have been manufactured by Hitachi.

(2) H8S family: An enhanced version of the H8 family.

Table 2.1 lists Hitachi's products that incorporate the I²C bus interface and the types of the I²C bus interface modules.

Table 2.1 Products that Incorporate the I²C Bus Interface

Series		Product name	Number of pins	Channel	MASK*[1]	F-ZTAT™	ZTAT®	I²C module
H8/300 series	H8/3217 series	H8/3217	64, 80	2ch	○	—	○	H8 series
		H8/3216		2ch	○	—	—	
		H8/3214		2ch	○	—	○	
		H8/3212		2ch	○	—	—	
		H8/3202		1ch	○	—	—	
	H8/3337 series	H8/3337Y	80, 84	1ch	○	—	○	
		H8/3337YF		1ch	—	○	—	
		H8/3337SF		1ch	—	○	—	
		H8/3336Y		1ch	○	—	—	
		H8/3334Y		1ch	○	—	○	
		H8/3334YF		1ch	—	○	—	
	H8/3437 series	H8/3437	100	1ch	○	—	○	
		H8/3437YF		1ch	—	○	—	
		H8/3437SF		1ch	—	○	—	
		H8/3436		1ch	○	—	—	
		H8/3434		1ch	○	—	○	
		H8/3434F		1ch	—	○	—	

Table 2.1 Products that Incorporate the I²C Bus Interface (continued)

Series		Product name	Number of pins	Channel	MASK*[1]	F-ZTAT™	ZTAT®	I²C module
H8/300 series	H8/3567 series	H8/3567	42, 44	2ch	○	—	○	H8S series
		H8/3564		2ch	○	—	—	
		H8/3561		2ch	○	—	—	
		H8/3567U		2ch	○	—	○	
		H8/3564U		2ch	○	—	—	
	H8/3577 series	H8/3577	64	2ch	○	—	○	
		H8/3574		2ch	○	○	—	
H8/300L series	H8/3947 series	H8/3947	100	2ch	○	—	○	H8 series
		H8/3946		2ch	○	—	—	
		H8/3945		2ch	○	—	—	
H8/300H Tiny series*[2]	H8/3664 series	H8/3664	42, 64	1ch	○	○	—	H8S series
H8S series	H8S/2100 series	H8S/2127	64, 80	2ch	○	—	—	H8S series
		H8S/2126		2ch	○	—	—	
		H8S/2128F		2ch	—	○	—	
		H8S/2138	80	2ch	○	—	—	
		H8S/2137		2ch	○	—	—	
		H8S/2138F		2ch	—	○	—	
		H8S/2148	100	2ch	○	—	—	
		H8S/2147		2ch	○	—	—	
		H8S/2148F		2ch	—	○	—	
		H8S/2147NF		2ch	—	○	—	
		H8S/2149YVF		2ch	—	○	—	
		H8S/2169YVF	144	2ch	—	○	—	
		H8S/2194	112	1ch	○	—	—	
		H8S/2193		1ch	○	—	—	
		H8S/2192		1ch	○	—	—	
		H8S/2191		1ch	○	—	—	
		H8S/2194F		1ch	—	○	—	

Table 2.1 Products that Incorporate the I²C Bus Interface (continued)

Series		Product name	Number of pins	Channel	MASK*1	F-ZTAT™	ZTAT®	I²C module
		H8S/2199		2ch	○	—	—	
		H8S/2198		2ch	○	—	—	
		H8S/2197		2ch	○	—	—	
		H8S/2196		2ch	○	—	—	
		H8S/2199F		2ch	—	○	—	
H8S series	H8S/2200 series	H8S/2238	100	2ch	○*2	○*2	—	H8S series
		H8S/2236		2ch	○*2	—	—	
		H8S/2258		2ch	○*2	○*2	—	
		H8S/2256		2ch	○*2	—	—	
	H8S/2600 series	H8S/2633	120,128	2ch	—	○	—	
		H8S/2643*2	144	2ch	—	○	—	

Notes: *1 MASK versions are available.

*2 For details on the specification/usage of the I²C bus interface which is included in the H8/300H Tiny series, see the additional volume.

2.2 Specifications of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products [H8 Series]

2.2.1 Specifications of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products

The main specifications of the I²C bus interfaces incorporated in Hitachi's H8/300 series and H8/300L series 8-bit microcomputers are shown below. For the groups of products that incorporate this module, see table 2.1.

- Units for data transfer
  - number of bits on each transfer:1 to 8 bits
  - number of frames to be transferred: unlimited
- Automatic setting of start/stop conditions
- Automatic loading of acknowledge bits
- Wait function
- Internal clock signals can be selected from among eight types.
- Acknowledgement and serial modes are available.
- Selectable order of output for the data to be transmitted (selection of MSB/LSB first)

- The on-chip filter (noise canceller) keeps the data reliable.

2.2.2 Configuration of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products

Figure 2.1 is an internal block diagram of the I²C bus interface. It consists of a prescaler (PS), clock controller, data control circuit, bus-state decision circuit, bus-arbitration decision circuit, address comparator, interrupt controller, and a group of registers that store the bus information and data.

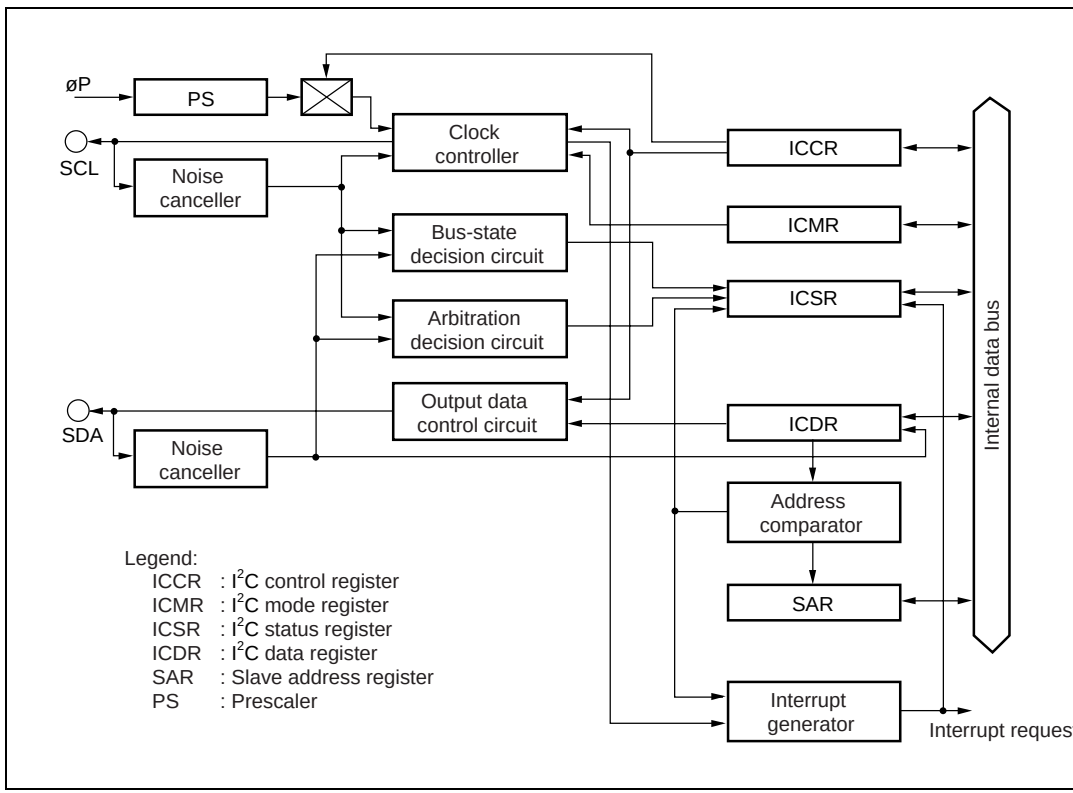


Figure 2.1 Block Diagram of I²C Bus Interface

Table 2.2 is a list of the registers.

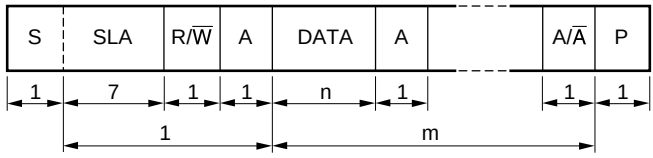
Table 2.2 Internal Registers of the I²C Bus Interface

Name	Abbrev.	Function
I²C bus control register	ICCR	Register for setting transfer mode
I²C bus status register	ICSR	The various state flags are set here
I²C bus data register	ICDR	Stores data for transmission/reception
I²C bus mode register	ICMR	Register to set the transfer format
Slave address register	SAR	Register to set the slave address

2.2.3 Data Transfer Format of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products

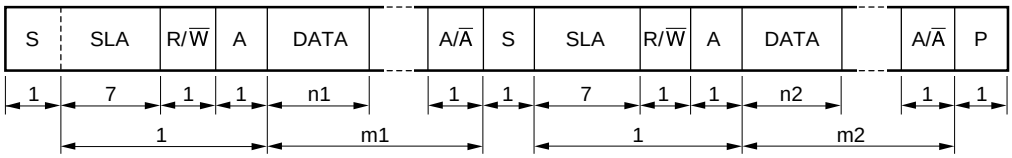
The I²C bus interface handles the following three formats for the transfer of data. There is no limit on the number of frames transferred.

(1) Addressing format



Number of transmission bits: n = 1 to 8
Number of transmission frames: m = 1 or more

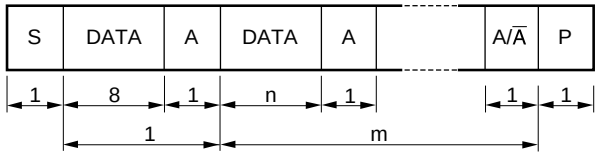
(2) Addressing format (with resending of the start condition signal)



Number of transmission bits: n1 and n2 = 1 to 8
Number of transmission frames: m1 and m2 = 1 or more

The addressing format with resending of the start condition is used in cases where the direction of the transfer must be changed during the transfer (structuring of the data transfer). After the resending start condition is sent, the slave address is made the same as that when the first start condition was set.

(3) Non-addressing format



Number of transmission bits:
n = 1 to 8

Number of transmission frames:
m = 1 or more

Legend:

S	: Start condition
SLA	: Slave address
R/$\overline{W}$	: Indicates the direction of transmission/reception
A	: Acknowledge (the reception device drives SDA low)
DATA	: Transmission/reception data
P	: Stop condition

The slave address and R/W bit are not recognized in this format.

2.2.4 Explanation of Functions of the Registers of the I²C Bus Interfaces Incorporated in H8/300 Series and H8/300L Series Products

Table 2.3 lists the function of each bit of the registers of this I²C bus interface.

Table 2.3 Functions of the Incorporated Registers of the I²C Bus Interface

Register name	Bit name	Function		Master	Slave	Site and Properties of this Setting
WSCR*[1]	CKDBL	Selects whether or not the frequency of the input clock to the peripheral module is divided by two.		○		A: Set in the initial setting routine. The values are retained. Confirm the completion of processing by the I²C bus interface when changing the settings in this register.
STCR*[1]	IICE	Enables access to the registers of the I²C bus interface.		○	○	
	IICX	Selects the transfer clock's frequency according to the settings of CSK2 to 0 in ICCR.		○		
SAR	FS	When ICE = 0	Selects whether or not the slave address of this interface is recognized	○	○	
	SLV6 to 0		Hold the slave address. Only enabled when FS =0.	○	○	
ICMR	MLS	When ICE = 1	Selects MSB or LSB first.	○	○	B: Set while the SCL clock has stopped (when the bus is released, and the transmission/reception of data is complete). The values are retained.
	WAIT		Selects whether a wait is inserted between the data and acknowledgement by the transmission equipment.	○		
	BC2 to 0		Specify the transfer bit. Set immediately before transfers other than 8 bits.	○	○	

Register name	Bit name	Function	Master	Slave	Site and Properties of this Setting
ICCR	ICE*[2]	1 is set after SAR is set. The I²C bus interface enters the transfer-enabled state.	○	○	
	IEIC	Disables/enables the interrupt.	○	○	
	MST	Sets master/slave and transmission/reception. The communications mode (transmission/reception) of the slave is automatically set according to the TRS bit setting in the master's interface.	○	○	
	TRS		○	○	
	ACK	Specifies whether the acknowledge bit is or is not inserted after 8-bit serial data has been transmitted.	○	○	
	CKS2 to 0	Specify the transfer rate.	○		
ICSR	BBSY	BBSY monitors the bus state.	○	○	C: Flags that are automatically set during the process of data communication. Clear them in order according to the communications protocol (BBSY and SCP are also used to set the start/stop conditions).
	SCP	Sets the start/stop condition. ·The start condition is set by setting BBSY = 1 and SCP = 0. ·The stop condition is set by setting BBSY = 0 and SCP = 0.	○		
	IRIC	Set to 1 when this interface is an interrupt source.	○	○	
	AL	Set to 1 when losing in bus arbitration.	○		
	AAS	Set to 1 when the slave address transmitted by the master matches the value in SAR.		○	
	ADZ	Set to 0 when the general call address (H'00) is recognized.		○	
	ACKB	Sets/recognizes the acknowledge bit.	○	○	As described under B above.
ICDR	ICDR7 to 0	Data register for transmission/reception.	○	○	Accessed in the transmission and reception of data.

Notes: *1 Only applies to H8/3337 series, H8/3437 series, and H8/3217 series products.

*2 The ICE bit is used to control the switching of the I/O port between operation as an I^2C bus module and as a general-purpose I/O port. When the ICE bit is switched, a clock signal or start/stop condition may be generated as a pseudo-state according to the state of the setting of the general-purpose I/O port. As a result, there is the possibility that a defect will be caused in some other device. When this bit is manipulated, the corresponding port is recommended to be set in the input state or to output a high level.

2.3 Specifications of the I^2C Bus Interfaces Incorporated in H8S Series Products

2.3.1 Features of the I^2C Bus Interfaces Incorporated in H8S Series Products

The main features of the I^2C bus interface incorporated in H8S series products are illustrated with Hitachi's 16-bit single chip H8S/2138 series microprocessor as an example.

- Selection of format as addressing or non-addressing
  - — I^2C bus format: addressing format with acknowledge bit, for master/slave operation.
  - — Serial format: non-addressing format without acknowledgement bit, for master operation only
- The I^2C bus format conforms to the specification of the Philips I^2C bus interface.
- There are two ways of setting the slave address in the I^2C bus format.
- Start and stop conditions are generated automatically in master mode in the I^2C bus format.
- Selection of acknowledge output levels when receiving in the I^2C bus format.
- Automatic loading of acknowledge bit when transmitting in the I^2C bus format
- Wait function in master mode in the I^2C bus format
  - — A wait can be inserted by driving the SCL pin low after transfers of data other than acknowledgement bits. The wait request is cleared when the next transfer becomes possible.
- Wait function in slave mode in the I^2C bus format
  - — A wait request can be generated by driving the SCL after the transfers of data other than acknowledgement bits. The wait request is cleared when the next transfer becomes possible.
- Five interrupt sources
  - — Detection of start condition (in master mode)
  - — End of data transfer : at the rising edge of the ninth clock of the SCL, including transmission mode transitions with I^2C bus format and address reception after loss of the master arbitration.

— Address match: when any slave address matches the address of this unit or the general call address is received while the unit is in the I^2C bus format's slave reception mode.

— Detection of stop condition (in slave mode)

— When the internal flag TDRE or RDRF is set to 1 (when data is transferred from ICDRT to ICDRS or from ICDRS to ICDRR)

- Selection from among 16 internal clocks while in master mode
- Direct bus drive (SCL/SDA pins)
  — Two pins, P52/SCL0 and P97/SDA0, normally function as NMOS push-pull outputs and function as NMOS open-drain outputs when the bus-drive function is selected.

  — Two pins, P86/SCL1 and P42/SDA1, normally function as CMOS pins and only function as NMOS outputs when the bus-drive function is selected.

- An on-chip-filter (noise canceller) is provided to maintain the reliability of data.
- The control function is supported in the standard DDC (display data channel) for PC monitors.
  — Automatic switching from format-less to I^2C bus format is possible (only on channel 0).

  — Format-less operation (i.e., without start/stop condition, non-addressing) in slave mode

  — Operation in the pin configuration of common data pin (SDA) and independent clock pins (VSYNCI and SCL).

  — Automatic switching from format-less mode to I^2C bus mode on the falling edge of SCL.

2.3.2 Internal Block Configuration of the H8S Series I²C Bus Interface

Figure 2.2 shows the internal block diagram of the I²C bus interface for H8S/2138 series products.

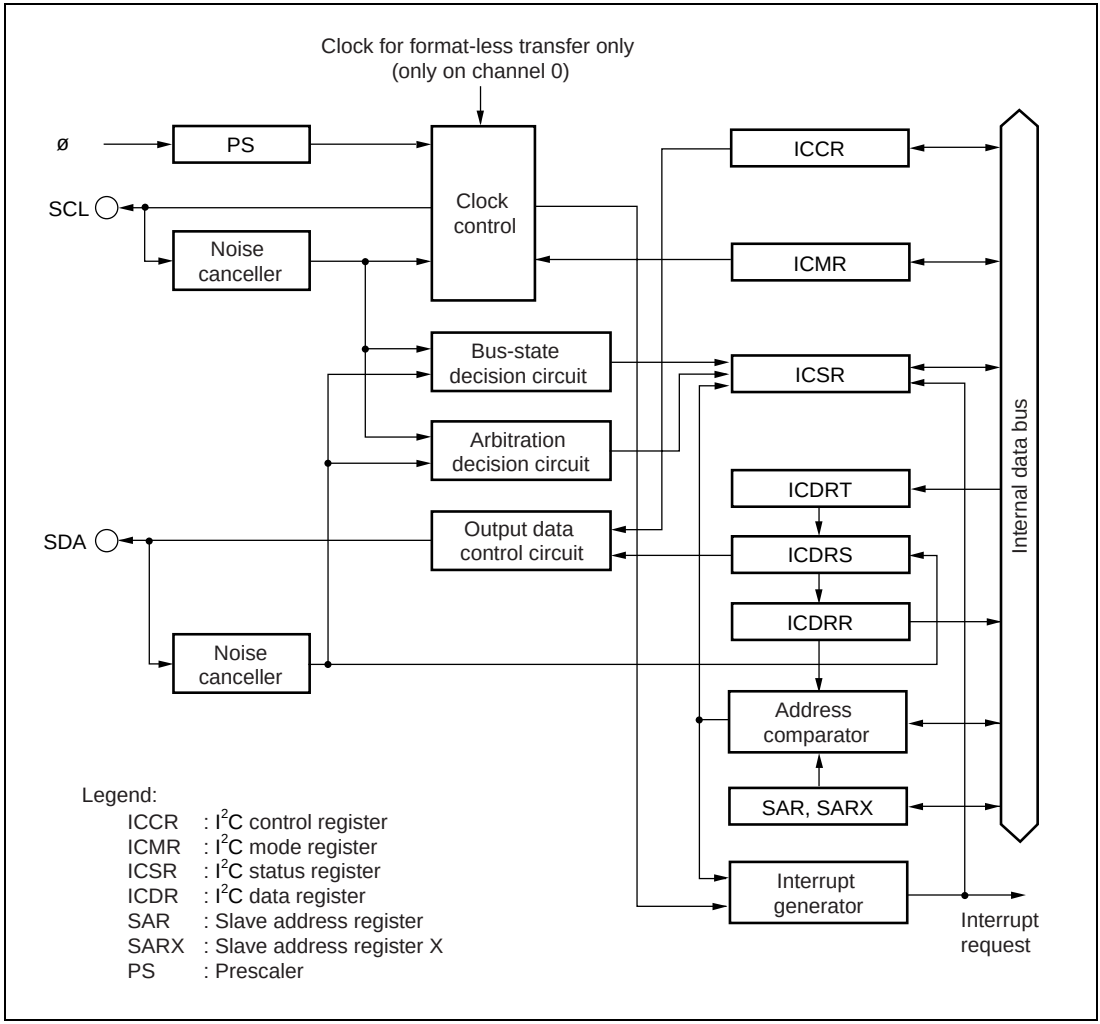


Figure 2.2 Block Diagram of the H8S/2138 Series I²C Bus Interface

The registers are described in table 2.4.

Table 2.4 The Registers of the H8S/2138 Series I²C Bus Interface

Name	Abbrev.	Function
I²C bus control register	ICCR	Register for setting transfer mode
I²C bus status register	ICSR	Each state flag is set.
I²C bus data register	ICDR	Stores received data and data for transmission
I²C bus mode register	ICMR	Register to set the transfer format
Slave address register	SAR	Register to set the slave address
Slave address register	SARX	Register to set the second slave address

2.3.3 Data Format for the H8S Series I²C Bus

Figures 2.3 shows the format for the I²C bus of H8S series products. The I²C bus format is made up of the start condition, the slave address field (7-bit addressing) that specifies the slave device's address, the R/W-bit field that indicates the direction of communications, the acknowledge-bit field, data field, and stop condition (for a description of the symbols, see table 2.5).

This I²C bus interface module allows the use of format-less and serial formats, as well as the I²C bus format itself (this is so for IIC channel 0 in H8S/2138 series products; for other products, confirm the details on this point in the respective hardware manuals). The additional modes are shown in figure 2.3, (c) and (d).

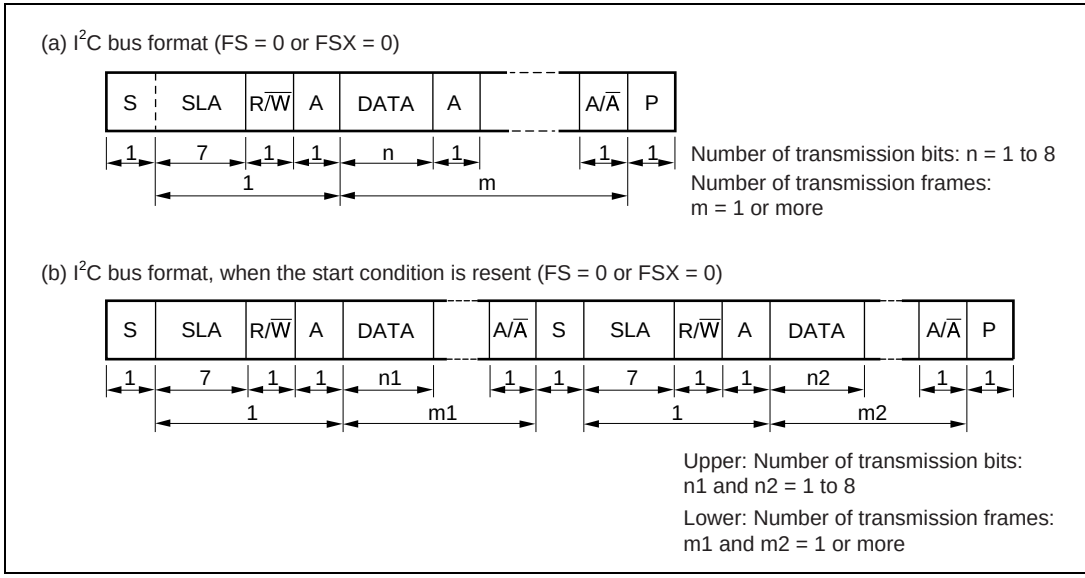


Figure 2.3 The I²C Bus Data Format

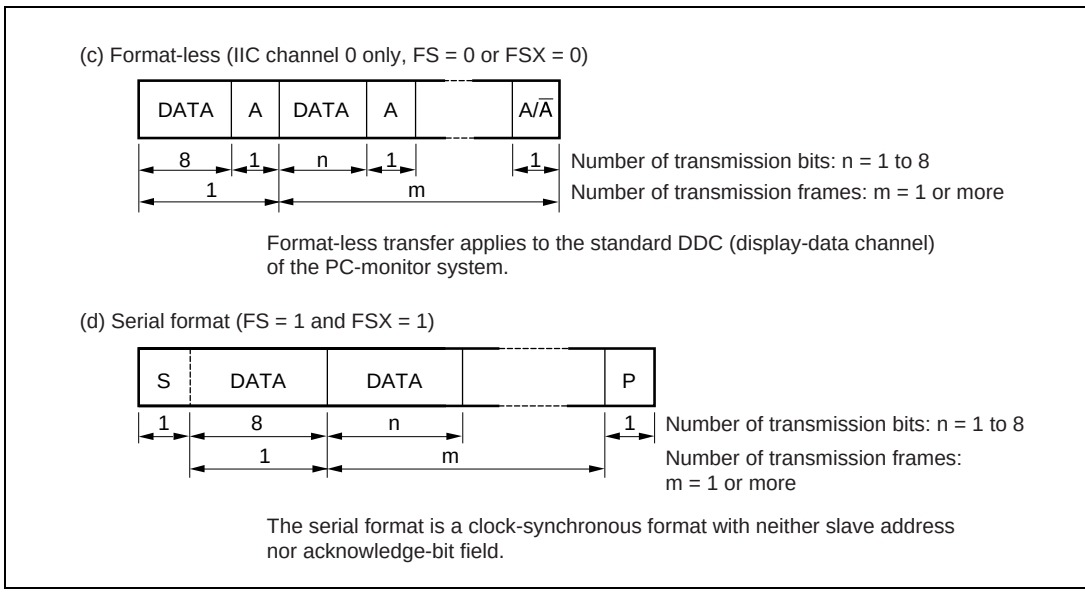


Figure 2.3 Other Data Formats

Table 2.5 lists the description of symbols in the I²C bus data format and I²C bus timing.

Table 2.5 Symbols

Symbol	Function
S	Start condition. The master device drives SDA from high to low while SCL is high.
SLA	Slave address, by which the master device selects a slave device.
R/$\overline{W}$	Indicates the direction of transmission/reception: from the slave device to the master device when the R/$\overline{W}$ bit is 1, or from the master device to the slave device when the R/$\overline{W}$ bit is 0.
A	Acknowledge. The reception device (the slave in master-transmit mode or the master in master-receive mode) drives SDA to its low level to acknowledge a transfer.
DATA	The data being transferred. The number of bits of data to be transmitted and received is set by bits BC2 to BC0 in ICMR. Either the MSB-first or LSB-first format is selected by the MLS bit in ICMR.
P	Stop condition. The master device drives SDA from low to high level while SCL is high.

Figure 2.4 shows the timing of the I^{2}C bus.

Start condition (S): Operation in which SDA is changed from high to low while SCL is high.

Stop condition (P): Operation in which SDA is changed from low to high while SCL is in its high state

Data (SLA/R/$\overline{W}$/DATA): Settled for placement on SDA while SCL is high.

For the ac characteristics of the bus, see the hardware manuals for the individual products.

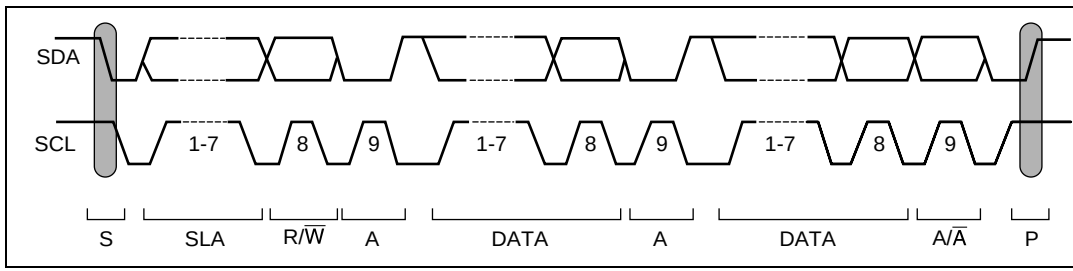


Figure 2.4 I^{2}C Bus Timing

2.3.4 Description of Functions of the H8S Series I^{2}C Bus Interface Incorporated Registers

Table 2.6 lists the functions of the H8S series I^{2}C bus interface incorporated registers of H8S series (H8S/2138 series).

Table 2.6 Description of functions of built-in registers

Register name	Bit name	Functions	R/W	Initial value
ICDR	ICDR7 to 0	ICDR is an 8-bit readable/writable register that is used as a transmit data register when transmission and reception data register when receiving. ICDR is divided internally into a shift register (ICDRS), receive buffer (ICDRR), and transmit buffer (ICDRT). ICDRS cannot be read or written to by the CPU, ICDRR is read-only, and ICDRT is write-only. Data transfer among the three registers is performed automatically in coordination with changes in the bus state, and affect the status of internal flags such as TDRE and RDRF. If IIC is in transmit mode and the next data is in ICDRT (the TDRE flag is 0) following transmission/reception of one frame of data using ICDRS, data is transferred automatically from ICDRT to ICDRS. If IIC is in receive mode and no previous data remains in ICDRR (the RDRF flag is 0) following reception of one frame of data using ICDRS, data is transferred automatically from ICDRS to ICDRR. ICDR is assigned to the same address as SARX, and can be written and read only when the ICE bit is set to 1 in ICCR.	R/W	—
—	TDRE	TDRE is a one bit internal flag that cannot be read/written. • TDRE = 0 indicates that transmission cannot be started or the next transmit data is in ICDR (ICDRT). [Clear conditions] (1) When transmit data is written in ICDR (ICDRT) in transmit mode (TRS = 1) (2) When a stop condition establishment is detected in the bus line state after a stop condition is set with the I^2C bus format or serial format selected (3) When a stop condition is detected with the I^2C bus format selected (4) In receive mode (TRS = 0) (A0 write to TRS during transfer is valid after reception of a frame containing an acknowledge bit.) • TDRE = 1 indicates that the next transmit data can be written in ICDR (ICDRT). [Set conditions] (1) In transmit mode (TRS = 1), when a start condition is detected in the bus line state after a start condition is set in master mode with the I^2C bus format or serial format selected	—	0

Register name	Bit name	Functions	R/W	Initial value
		(2) When using formatless mode in transmit mode (TRS = 1) (3) When data is transferred from ICDRT to ICDRS (Data transfer from ICDRT to ICDRS when TRS = 1 and TDRE = 0, and ICDRS is empty) (4) When a switch is made from receive mode (TRS = 0) to transmit mode (TRS = 1) after detection of a start condition		
—	RDRF	RDRF is a one bit internal flag that cannot be read/written. • RDRF = 0 indicates that the data in ICDR (ICDRR) is invalid. • RDRF = 1 indicates that the receive data in ICDR (ICDRR) can be read. [Clearing conditions] When ICDR (ICDRR) receive data is read in receive mode [Setting conditions] When data is transferred from ICDRS to ICDRR (Data transfer from ICDRS to ICDRR in case of normal transmission termination with TRS = 0 and RDRF = 0)	—	0
SAR		SAR is an 8-bit readable/writable register that selects the format and stores the slave address. When the chip is in slave mode (and the addressing mode is selected), if the upper 7 bits of SAR match the upper 7 bits of the first frame received after a start condition, the chip operates as the slave device specified by the master device. SAR is assigned to the same address as ICMR, and can be written and read only when the ICE bit is cleared to 0 in ICCR.	R/W	H'00
	SVA6 to 0	A unique address is set in bits SVA6 to SVA0, differing from the addresses of other slave devices connected to the I^2C bus.	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	FS	Used together with the FSX bit in SARX and the SW bit in DDCSWR to select the transfer format. • SW = 0, FS = 0, FSX = 0 I²C bus format (SAR and SARX slave address are recognizes) • SW = 0, FS = 0, FSX = 1 I²C bus format (SAR slave address is recognized and SARX slave address is ignored) • SW = 0, FS = 1, FSX = 0 I²C bus format (SAR slave address is ignored and SARX slave address is recognized) • SW = 0, FS = 1, FSX = 1 Clock synchronous serial format (SAR and SARX slave addresses ignored) • SW = 1, FS = 0, FSX = 0 • SW = 1, FS = 0, FSX = 1 • SW = 1, FS = 1, FSX = 0 Formatless (start condition/stop condition is not detected, with acknowledge bit) • SW = 1, FS = 1, FSX = 1 Formatless (start condition/stop condition is not detected, without acknowledge bit)	R/W	0
SARX		SARX is an 8-bit readable/writable register that selects the format and stores the second slave address. When the chip is in slave mode (and the addressing format is selected), if the upper 7 bits of the first frame received after a start condition and the upper 7 bits of SARX match, the chip operates as the slave device specified by the master device. SARX is assigned to the same address as ICDR, and can be written and read only when the ICE bit is cleared to 0 in ICCR.	R/W	H'01
	SVAX6 to 0	A unique address differing from the addresses of other slave devices connected to the I²C bus is set in bits SVAX6 to SVAX0.	R/W	0
	FSX	The FSX bit selects whether or not SARX slave address is recognized in slave mode. For details, see the description of the FS bit in SAR.	R/W	1

Register name	Bit name	Functions	R/W	Initial value
ICMR		ICMR is an 8-bit readable/writable register that selects whether the MSB or LSB is transferred first, performs master mode wait control, and selects the master mode transfer clock frequency, and the transfer bit count. ICMR is assigned to the same address as SAR. ICMR can be written and read only when the ICE bit is set to 1 in ICCR.	R/W	H'00
	MLS	MLS selects whether data is transferred MSB-first or LSB-first (if the number of bits in a frame, excluding the acknowledge bit, is less than 8, transmit data and receive data are stored differently. Transmit data should be written justified toward the MSB side when MLS = 0, and toward the LSB side when MLS = 1. Receive data bits read from the LSB side should be treated as valid when MLS = 0, and bits read from the MSB side when MLS = 1). MLS should not be set to 1 when they are used in the I^2C bus format. • MLS = 0 MSB-first • MLS = 1 LSB-first	R/W	0
	WAIT	WAIT selects whether to insert a wait between the transfer of data and the acknowledge bit, in master mode with the I^2C bus format. When WAIT is set to 1, after the fall of the clock for the final data bit, the IRIC flag is set to 1 in ICCR, and a wait state begins (with SCL at the low level). When the IRIC flag is cleared to 0 in ICCR, the wait ends and the acknowledge bit is transferred. If WAIT is cleared to 0, data and acknowledge bits are transferred consecutively with no wait inserted. The IRIC flag in ICCR is set to 1 on completion of the acknowledge bit transfer, regardless of the WAIT setting. • WAIT = 0 Data and acknowledge bits transferred consecutively • WAIT = 1 Wait inserted between data and acknowledge bits	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	CKS2 to CKS0	Bits CKS2 to CKS0, together with the IICX1 (channel 1) or IICX0 (channel 0) bit in the STCR register, select the transfer clock frequency in master mode. They should be set according to the required transfer rate. • IICX = 0, CKS2 = 0, CKS1 = 0, CKS0 = 0 The transfer clock is set to ϕ/28. • IICX = 0, CKS2 = 0, CKS1 = 0, CKS0 = 1 The transfer clock is set to ϕ/40. • IICX = 0, CKS2 = 0, CKS1 = 1, CKS0 = 0 The transfer clock is set to ϕ/48. • IICX = 0, CKS2 = 0, CKS1 = 1, CKS0 = 1 The transfer clock is set to ϕ/64. • IICX = 0, CKS2 = 1, CKS1 = 1, CKS0 = 0 The transfer clock is set to ϕ/80. • IICX = 0, CKS2 = 1, CKS1 = 0, CKS0 = 1 The transfer clock is set to ϕ/100. • IICX = 0, CKS2 = 1, CKS1 = 1, CKS0 = 0 The transfer clock is set to ϕ/112. • IICX = 0, CKS2 = 1, CKS1 = 1, CKS0 = 1 The transfer clock is set to ϕ/128. • IICX = 1, CKS2 = 0, CKS1 = 0, CKS0 = 0 The transfer clock is set to ϕ/56. • IICX = 1, CKS2 = 0, CKS1 = 0, CKS0 = 1 The transfer clock is set to ϕ/80. • IICX = 1, CKS2 = 0, CKS1 = 1, CKS0 = 0 The transfer clock is set to ϕ/96. • IICX = 1, CKS2 = 0, CKS1 = 1, CKS0 = 1 The transfer clock is set to ϕ/128. • IICX = 1, CKS2 = 1, CKS1 = 0, CKS0 = 0 The transfer clock is set to ϕ/160. • IICX = 1, CKS2 = 1, CKS1 = 0, CKS0 = 1 The transfer clock is set to ϕ/200. • IICX = 1, CKS2 = 1, CKS1 = 1, CKS0 = 0 The transfer clock is set to ϕ/224. • IICX = 1, CKS2 = 1, CKS1 = 1, CKS0 = 1 The transfer clock is set to ϕ/256.	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	BC2 to BC0	Bits BC2 to BC0 specify the number of bits to be transferred next time. With the I^2C bus format (when the FS bit in SAR or the FSX bit in SARX is 0), the data is transferred with one additional acknowledge bit. Bit BC2 to BC0 settings should be made during an interval between transfer frames. If bits BC2 to BC0 are set to a value other than 000, the setting should be made while the SCL line is low. Bits BC2 to BC0 are initialized to 000 by a reset and when a start condition is detected. The value returns to 000 at the end of a data transfer, including the acknowledge bit. • BC2 = 0, BC1 = 0, BC0 = 0 Clock synchronous serial = 8 bits/frame I^2C bus = 9 bits/frame • BC2 = 0, BC1 = 0, BC0 = 1 Clock synchronous serial = 1 bit/frame I^2C bus = 2 bits/frame • BC2 = 0, BC1 = 1, BC0 = 0 Clock synchronous serial = 2 bits/frame I^2C bus = 3 bits/frame • BC2 = 0, BC1 = 1, BC0 = 1 Clock synchronous serial = 3 bits/frame I^2C bus = 4 bits/frame • BC2 = 1, BC1 = 0, BC0 = 0 Clock synchronous serial = 4 bits/frame I^2C bus = 5 bits/frame • BC2 = 1, BC1 = 0, BC0 = 1 Clock synchronous serial = 5 bits/frame I^2C bus = 6 bits/frame • BC2 = 1, BC1 = 1, BC0 = 0 Clock synchronous serial = 6 bits/frame I^2C bus = 7 bits/frame • BC2 = 1, BC1 = 1, BC0 = 1 Clock synchronous serial = 7 bits/frame I^2C bus = 8 bits/frame	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	AAS	In I²C bus format slave receive mode, AAS is set to 1 if the first frame following a start condition matches bits SVA6 to SVA0 in SAR, or if the general call address (H'00) is detected. AAS is cleared by reading AAS after it has been set to 1, then writing 0 in AAS. In addition, AAS is reset automatically by write access to ICDR in transmit mode, or read access to ICDR in receive mode. • AAS = 0 Slave address or general call address is not recognized. [Clear conditions] (1) When ICDR data is written (transmit mode) or read (receive mode) (2) When 0 is written in AAS after reading AAS = 1 (3) In master mode • AAS = 1 Slave address or general call address is recognized. [Setting condition] When the slave address or general call address is detected in slave receive mode and FS = 0	R/(W)*1	0
	ADZ	In I²C bus format slave receive mode, ADZ is set to 1 if the first frame following a start condition is the general call address (H'00). ADZ is cleared by reading ADZ after it has been set to 1, then writing 0 in ADZ. In addition, ADZ is reset automatically by write access to ICDR in transmit mode, or read access to ICDR in receive mode. • ADZ = 0 General call address is not recognized. [Clearing conditions] (1) When ICDR data is written (transmit mode) or read (receive mode) (2) When 0 is written in ADZ after reading ADZ =1 (3) In master mode • ADZ = 1 General call address is recognized. [Setting condition] When the general call address is detected in slave receive mode and (FS = 0 or FSX = 0)	R/(W)*1	0

Register name	Bit name	Functions	R/W	Initial value
	ACKB	ACKB stores acknowledge data. In transmit mode, after the reception device receives data, it returns acknowledge data, and this data is loaded into ACKB. In receive mode, after data has been received, the acknowledge data set in this bit is sent to the transmission device. When this bit is read, in transmission (when TRS = 1), the value loaded from the bus line (returns by the reception device) is read. In reception (when TRS = 0), the value set is read. • ACKB = 0 • In receive mode, 0 is output at acknowledge output timing • In transmit mode, indicates that the reception device has acknowledged the data (signal is 0). • ACKB = 1 • In receive mode, 1 is output at acknowledge output timing. • In transmit mode, indicates that the reception device has not acknowledge the data (signal is 1).	R/W	0
ICCR		ICCR is an 8-bit readable/writable register that enables or disables the I^2C bus interface operation, enables or disables interrupts, selects master or slave mode and transmission or reception, enables or disables acknowledgement, confirms the I^2C bus interface bus status, sets start/stop conditions, and performs interrupt flag confirmation.	R/W	H'01
	ICE	ICE selects whether or not the I^2C bus interface is to be used. When ICE is set to 1, port pins function as SCL and SDA input/output pins and transfer operations are enabled in the I^2C bus interface module. When ICE is cleared to 0, the I^2C bus interface module is halted and its internal states are cleared. The SAR and SARX registers can be accessed when ICE is 0. The ICMR and ICDR registers can be accessed when ICE is 1. • ICE = 0 I^2C bus interface module is disabled (SCL and SDA signal pins set to port function). I^2C bus interface module internal states are initialized. SAR and SARX can be accessed. • ICE = 1 I^2C bus interface module is enabled for transfer operation (pins SCL and SCA are driving the bus). ICMR and ICDR can be accessed.	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	IEIC	IEIC enables or disables interrupts from the I^2C bus interface to the CPU. • IEIC = 0 I^2C bus interface interrupts are disabled. • IEIC = 1 I^2C bus interface interrupts are enabled.	R/W	0
	MST	MST selects whether the I^2C bus interface operates in master mode or slave mode. • MST = 0 Slave mode [Clearing conditions] (1) When 0 is written by software (2) When bus arbitration is lost after transmission is started in I^2C bus format master mode • MST = 1 Master mode [Setting conditions] (1) When 1 is written by software (in cases other than clearing condition 2) (2) When 1 is written in MST after reading MST = 0	R/W	0
	TRS	TRS selects whether the I^2C bus interface operates in transmit mode or receive mode. • TRS = 0 Reception mode [Clearing conditions] (1) When 0 is written by software (in cases other than setting condition 3) (2) When 0 is written in TRS after reading TRS = 1 (in case of setting condition 3) (3) When bus arbitration is lost after transmission is started in I^2C bus format master mode (4) When the SW bit in DDCSWR changes from 1 to 0 • TRS = 1 Transmit mode [Setting conditions] (1) When 1 is written by software (in cases other than clearing conditions 3 and 4)	R/W	0

Register name	Bit name	Functions	R/W	Initial value
		(2) When 1 is written in TRS after reading TRS = 0 (in case of clearing conditions 3 and 4) (3) When 1 is received as the R/$\overline{W}$ bit of the first frame in I^2C bus format slave mode		
	ACKE	ACKE specifies whether the value of the acknowledge bit returned from the reception device when using the I^2C bus format is to be ignored and continuous transfer is performed, or transfer is to be aborted and error handling will be performed if the acknowledge bit is 1. When the ACKE bit is 0, the value of the received acknowledge bit is not indicated by the ACKB bit, which is always 0. • ACKE = 0 The value of the acknowledge bit is ignored, and continuous transfer is performed. • ACKE = 1 If the acknowledge bit is 1, continuous transfer is aborted.	R/W	0
	BBSY	The BBSY flag can be read to check whether the I^2C bus (SCL, SDA) is busy or free. In master mode, this bit is also used to set start and stop conditions. A high-to-low transition of SDA while SCL is high is recognized as a start condition, setting BBSY to 1. A low-to-high transition of SDA while SCL is high is recognized as a stop condition, clearing BBSY to 0. To set a start condition, write 1 in BBSY and 0 in SCP. A retransmit start condition is set in the same way. To set a stop condition, write 0 in BBSY and 0 in SCP. It is not possible to write to BBSY in slave mode: the I^2C bus interface must be set to master transmit mode before issuing a start condition. MST and TRS should both be set to 1 before writing 1 in BBSY and 0 in SCP. • BBSY = 0 Bus is free. [Clearing condition] When a stop condition is detected • BBSY = 1 Bus is busy. [Setting condition] When a start condition is detected	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	IRIC	IRIC indicates that the I²C bus interface has issued an interrupt request to the CPU. IRIC is set to 1 at the end of a data transfer, when a slave address or general call address is detected in slave receive mode, when bus arbitration is lost in master transmit mode, and when a stop condition is detected. IRIC is set at different times depending on the FS bit in SAR and the WAIT bit in ICMR. The conditions under which IRIC is set also differ depending on the setting of the ACKE bit in ICCR. IRIC is cleared by reading IRIC after it has been set to 1, then writing 0 in IRIC. When the DTC is used, IRIC is cleared automatically and transfer can be performed continuously without CPU intervention. • IRIC = 0 Waiting for transfer, or transfer in progress [Clear conditions] (1) When 0 is written in IRIC after reading IRIC = 1 (2) When ICDR is written or read by the DTC (when the TDRE or RDFR flag is cleared to 0) • IRIC = 1 Interrupt requested. [Setting conditions] 1. I²C bus format master mode (1) When a start condition is detected in the bus line state after a start condition is set (when the TDRE flag is set to 1 because of first frame transmission) (2) When a wait is inserted between the data and acknowledge bit when WAIT = 1 (3) At the end of data transfer (at the rise of the 9th transmit/receive clock pulse, or at the fall of the 8th transmit/receive clock pulse when using wait insertion) (4) When a slave address is received after bus arbitration is lost (when the AL flag is set to 1) (5) When 1 is received as the acknowledge bit when the ACKE bit is 1 (when the ACKE bit is set to 1)	R/(W)*¹	0

Register name	Bit name	Functions	R/W	Initial value
		2. I^2C bus format slave mode (1) When the slave address (SVA, SVAX) matches (when the AAS and AASX flags are set to 1) and at the end of data transfer up to the subsequent retransmission start condition or stop condition detection (when the TDRE or RDRF flag is set to 1) (2) When the general call address is detected (when FS = 0 and the ADZ flag is set to 1) and at the end of data transfer up to the subsequent retransmission start condition or stop condition detection (when the TDRE or RDRF flag is set to 1) (3) When 1 is received as the acknowledge bit when the ACKE bit is 1 (when the ACKB bit is set to 1) (4) When a stop condition is detected (when the STOP or ESTP flag is set to 1) 3. Synchronous serial format and formatless (1) At the end of data transfer (when the TDRE or RDRF flag is set to 1) (2) When a start condition is detected with serial format selected (3) When the SW bit of DDCSWR is set to 1 (4) When any other condition arises in which the TDRE or RDRF flag is set to 1		
	SCP	The SCP bit controls the issuing of start and stop conditions in master mode. To set a start condition, write 1 in BBSY and 0 in SCP. A retransmit start condition is set in the same way. To set a stop condition, write 0 in BBSY and 0 in SCP. This SCP bit is always read as 1. If 1 is written, the data is not stored. • SCP = 0 Writing 0 sets a start or stop condition, in combination with the BBSY flag. • SCP = 1 Reading always returns a value of 1. Writing is ignored.	W	1

Register name	Bit name	Functions	R/W	Initial value
ICSR		ICSR is an 8-bit readable/writable register that performs flag confirmation and acknowledge confirmation and control.	R/W	H'00
	ESTP	The ESTP flag indicates that a stop condition has been detected during frame transfer in I²C bus format slave mode. • ESTP = 0 No error stop condition [Clearing conditions] (1) When 0 is written in ESTP after reading ESTP = 1 (2) When the IRIC flag is cleared to 0 • ESTP = 1 In I²C bus format slave mode, error stop condition is detected. [Setting condition] When a stop condition is detected during frame transfer • In I²C bus format slave mode No meaning	R/(W)*[1]	0
	STOP	The STOP flag indicates that a stop condition has been detected after completion of frame transfer in I²C bus format slave mode. • STOP = 0 No normal stop condition [Clearing conditions] (1) When 0 is written in STOP after reading STOP = 1 (2) When the IRIC flag is cleared to 0 • STOP = 1 • In I²C bus format slave mode Normal stop condition is detected. [Setting condition] When a stop condition is detected after completion of frame transfer • In mode other than slave mode in I²C bus format No meaning	R/(W)*	0

Register name	Bit name	Functions	R/W	Initial value
	IRTR	The IRTR flag indicates that the I²C bus interface has issued an interrupt request to the CPU, and the source is completion of reception/transmission of one frame in continuous transmission/reception operation for which DTC activation is possible. When the IRTR flag is set to 1, the IRIC flag is also set to 1 at the same time. IRTR flag setting is performed when the TDRE or RDRF flag is set to 1. IRTR is cleared by reading IRTR after it has been set to 1, then writing 0 in IRTR. IRTR is also cleared automatically when the IRIC flag is cleared to 0. • IRTR = 0 Waiting for transfer, or transfer in progress [Clearing conditions] (1) When 0 is written in IRTR after reading IRTR = 1 (2) When the IRIC flag is cleared to 0 • IRTR = 1 Continuous transfer state [Setting condition] • In I²C bus format slave mode When the TDRE or RDRF flag is set to 1 when AASX = 1 • In modes other than slave mode in I²C bus format When the TDRE or RDRF flag is set to 1	R/(W)*¹	0
	AASX	In I²C bus format slave receive mode, the AASX flag is set to 1 if the first frame following a start condition matches bits SVAX6 to SVAX0 in SARX. AASX is cleared by reading AASX after it has been set to1, then writing 0 in AASX. AASX is also cleared automatically when a start condition is detected. • AASX = 0 The second slave address is not recognized. [Clearing conditions] (1) When 0 is written in AASX after reading AASX = 1 (2) When a start condition is detected (3) In master mode • AASX = 1 The second slave address is recognized.	R/(W)*	0

Register name	Bit name	Functions	R/W	Initial value
		[Setting condition] When the second slave address is detected in slave receive mode and FSX = 0		
	AL	The AL flag indicates that arbitration was lost in master mode. The I²C bus interface monitors the bus. When two or more master devices attempt to seize the bus at nearly the same time, if the I²C bus interface detects data differing from the data it sent, it sets AL to 1 to indicate that the bus has been taken by another master. AL is cleared by reading AL after it has been set to 1, then writing 0 in AL. In addition, AL is reset automatically by write access to ICDR (transmit mode), or read access to ICDR (receive mode). • AL = 0 Bus arbitration won [Clearing condition] (1) When ICDR data is written (transmit mode) or read (receive mode) (2) When 0 is written in AL after reading AL= 1 • AL = 1 Arbitration lost [Set flag conditions] (1) If the internal SDA and SDA pin disagree at the rise of SCL in master transmit mode (2) If the internal SCL line is high at the fall of SCL in master transmit mode	R/(W)*1	0
STCR		STCR is an 8-bit readable/writable register that controls register access, the I²C interface operating mode (when the on-chip IIC option is included), and on-chip flash memory control (F-ZTAT version), and selects the input clock of TCNT. Details other than the I²C bus interface are omitted. If a module controlled by STCR is not used, do not write 1 to the corresponding bit.	R/W	H'00
	IICX1	The IICX1 bit, together with bits CKS2 to CKS0 in ICMR, selects the transfer rate in master mode of IIC channel 1. For details, see CSK2 to CSK0 in ICMR.	R/W	0
	IICX0	The IICX0 bit, together with bits CKS2 to CKS0 in ICMR, selects the transfer rate in master mode of IIC channel 0. For details, see CSK2 to CSK0 in ICMR.	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	IICE	The IICE bit controls CPU access to the I²C bus interface data and control registers (ICCR, ICSR, ICDR/SARX, ICMR/SAR). • IICE = 0 CPU access to I²C bus interface data and control registers is disabled. • IICE = 1 CPU access to I²C bus interface data and control registers is enabled.	R/W	0
DDCSWR		DDCSWR is an 8-bit readable/writable register that is used to control the format automatic switching of IIC channel 0 and controls the internal latch clear of IIC.	R/W	H'0F
	SWE	The SWE bit selects the automatic switching function from formatless to I²C bus format. • SWE = 0 Disables automatic switching of IIC channel 0 from formatless to I²C bus format. • SWE = 1 Enables automatic switching of IIC channel 0 from formatless to I²C bus format.	R/W	0
	SW	The SW bit selects formatless and I²C bus format in IIC channel 0. • SW = 0 IIC channel 0 is used in I²C bus format. [Clearing conditions] (1) When 0 is written by software (2) When a falling edge is detected in SCL when SWE = 1 • SW = 1 IIC channel 0 is used by formatless. [Setting conditions] When 1 is written after read in SW = 0	R/W	0

Register name	Bit name	Functions	R/W	Initial value
	IE	The IE bit enables/disables the interrupt request from CPU when the format's automatic switching is performed in IIC channel 0. • IE = 0 Interrupt when the format is automatically switched is disabled. • IE = 1 Interrupt when the format is automatically switched is enabled.	R/W	0
	IF	The IF bit is an interrupt request flag when the format is automatically switched in IIC channel 0. • IF = 0 Interrupt is not requested when format's automatic switching is carried out. [Clearing condition] When 0 is written after reading the sate of IF = 1 • IF = 1 Interrupt is requested when the format is automatically switched. [Setting condition] When a falling edge is detected in SCL when SWE = 1	R/W	0
	CLR3 to 0	Bits CLR3 to CLR0 control initialization of the internal state of IIC0 and IIC1. These bits can only be written to; if read, they will always return to a value of 1. When a write operation is performed on these bits, a clear signal is generated for the internal latch circuit of the corresponding module, and the internal state of the IIC module is initialized. The write data for these bits is not retained. To perform IIC clearance, bits CLR3 to CLR0 must be written to simultaneously using an MOV instruction. Do not use a bit manipulation instruction such as BCLR. When clearing is required again, all the bits must be written to in accordance with the setting. • CLR3 = 0, CLR2 = 0, CLR1 = *, CLR0 = *, setting is prohibited. • CLR3 = 0, CLR2 = 1, CLR1 = 0, CLR0 = 1, IIC0 internal latch is cleared.	W*[1]	1

Register name	Bit name	Functions	R/W	Initial value
		• CLR3 = 0, CLR2 = 1, CLR1 = 1, CLR0 = 0, IIC1 internal latch is cleared. • CLR3 = 0, CLR2 = 1, CLR1 = 1, CLR0 = 1, IIC0 and IIC1 internal latch is cleared • CLR3 = 1, CLR2 = *, CLR1 = *, CLR0 = *, setting is invalid. Note *: 0 or 1		
MSTPCRL	MSTP4	The MSTP4 bit specifies the module of IIC channel 0. • MSTP4 = 0 IIC channel 0 module stop mode is cleared. • MSTP4 = 1 IIC channel 0 module stop mode is set.	R/W	1
	MSTP3	The MSTP3 bit specifies IIC channel 1 module. • MSTP3 = 0 IIC channel 1 module stop mode is cleared. • MSTP3 = 1 IIC channel 1 module stop mode is set.	R/W	1

Note: *1 Always read as 1.

2.3.5 Relationship between Flags of On-chip I²C Bus Interface and Transfer State in H8S Series (H8S/2138 Series)

When an interruption occurs after the IRIC flag in ICCR has been set to 1 with the I²C bus format, it is necessary to check other flags to determine the cause of the IRIC flag being set to 1. Although each cause has its corresponding flag, special care must be taken at the end of a data transfer.

When the internal flags TDRE or RDRF are set, the readable IRTR flag can be either set or not set. Between the moment that the slave address (SVA) or general call address is matched and the moment that the restart condition or stop condition is detected in the slave mode of the I²C bus format, the IRTR flag, which is a DTC start request flag, is not set at the end of data transfer.

Even if the IRIC or IRTR flags are set, the internal flags TDRE or RDRF cannot be set. In the case of a continuous transfer using the DTC, the IRIC or IRTR flags are not cleared when the specified number of transfers has been completed. On the other hand, the flags TDRE or RDRF are cleared because the specified number of read/write actions of ICDR have been completed.

Table 2.7 shows the relationship between transfer states and flags.

Table 2.7 Relationship between Transfer States and Flags

MST	TRS	BBSY	ESTP	STOP	IRTR	AASX	AL	AAS	ADZ	ACKB	State
1/0	1/0	0	0	0	0	0	0	0	0	0	Idle state (flags must be cleared)
1	1	0	0	0	0	0	0	0	0	0	Setting the start condition
1	1	1	0	0	1	0	0	0	0	0	Start condition is satisfied
1	1/0	1	0	0	0	0	0	0	0	0/1	Master mode wait
1	1/0	1	0	0	1	0	0	0	0	0/1	Master mode transmit/receive end
0	0	1	0	0	0	1/0	1	1/0	1/0	0	Arbitration lost
0	0	1	0	0	0	0	0	1	0	0	Coincident with SAR in slave mode frame
0	0	1	0	0	0	0	0	1	1	0	Coincident with general call address
0	0	1	0	0	0	1	0	0	0	0	Coincident with SARX
0	1/0	1	0	0	0	0	0	0	0	0/1	End of slave mode transmission/reception (except for after SARX coincidence)
0	1/0	1	0	0	1	1	0	0	0	0	End of slave mode transmission/reception (after SARX coincidence)
0	1	1	0	0	0	1	0	0	0	1	
0	1/0	0	1/0	1/0	0	0	0	0	0	0/1	Stop condition detected

2.4 Description of I²C Bus Interface Usage

(1) How to confirm the bus state [H8 Series, H8S Series]

In the I²C bus, the master device must confirm whether or not the bus is in the open state (both SCL and SDA lines are constantly high) before starting to transfer data. This confirmation of the bus state can be performed by reading the BBSY bit in the ICSR register in the H8 series or in the ICCR register in the H8S series. When the BBSY bit is 0, which means that the bus is in the open state, the master device can start the data transfer.

(2) How to issue the start or stop conditions [H8 Series, H8S Series]

The start condition is the change from high to low in SDA when SCL is high. The stop condition is the change from low to high in SDA when SCL is high. The start condition can be generated by simultaneously writing BBSY=1 and SCP=0 into the register (ICSR in H8 series, ICCR in H8S series). Simultaneous writing BBSY=0 and SCP=0 allows the stop condition to be generated. Therefore, use the MOV instruction to issue the start/stop conditions.

Refer to section 2.4 (6), (7) “Continuous issuing of instructions”, and (8) “Notes on re-sending the start condition”.

(3) How to transmit data [H8 Series, H8S Series]

Master operation

Data transmission is started by writing data into the ICDR register. After the completion of the transmission (or after the start condition has been generated), the SCL line must be held low to generate the communication waiting state.

Slave operation

The low drive of the SCL line can be released by writing data into the ICDR register to prepare data transmission. Data must be transmitted to the master device by synchronizing the SCL clock that is sent from the master device. After the completion of the transmission, the SCL line must be held low to indicate the waiting state to the master device. After the completion of the last data transmission, release the SCL line by writing H'FF into the ICDR. This lets the master device issue the stop condition.

(4) How to receive data (H8 Series I^2C module) [H8 Series]

Master operation

Reading the ICDR register enables the SCL clock to be output and the data reception can be started. The first data reading is a dummy run. The actual data reception starts after the confirmation of the completion of the dummy data reception. After the completion of the data reception, the SCL line must be held low until the next read operation of ICDR to generate the communication waiting state. The last data must be read by setting TRS to 1 to enter transmit mode after confirming the end of the last data reception.

Slave operation

In the I^2C bus system, devices other than the master device start operation from slave reception mode. Since the first byte is a slave address + R/W bit, the SCL is made to be in high-impedance state and the slave address data is loaded in the data register (ICDR). When the eighth bit is loaded, the slave address register (SAR) is compared to the data register (ICDR). When addresses match, an acknowledge is returned to the master device at the ninth clock. At this time, if the IRIC flag is set and an I^2C bus interrupt is enabled (IEIC = 1), an interrupt occurs. When addresses do not match, the IRIC flag is not set and this I^2C module enters a wait state in slave mode.

The eighth bit in the slave address phase means an R/W bit. When this bit is 1, subsequent operations seen from the slave side are in transmit mode. When this bit is 0, subsequent operations are in receive mode. The eighth bit is automatically reflected to the TRS bit.

When the TRS bit is 0, slave reception mode is still entered. The SCL is driven to low until the CPU reads ICDR to indicate the waiting state to the master device (When the TRS bit is 1, slave transmission mode is entered. The SCL is driven to low until the CPU sets data in ICDR to indicate the waiting state to the master device).

(5) How to receive data (H8S Series I^2C module) [H8S Series]

For the H8S series I^2C module, a data reception buffer is composed of ICDRR (register which can be read by CPU, ICDR) and ICDRS (shift register). 2-byte-long data can be received after the data reception trigger (dummy reading of ICDR register) has been issued. The load on the CPU is thus reduced in application programs that read multiple data continuously.

Master operation

Reading the ICDR register enables the SCL clock to be output and the data reception can be started. The first data reading is a dummy run. The actual data reception starts after the confirmation of the completion of the dummy data reception. As the data buffer structure is doubled, the next data reception takes place when the ICDRR (ICDR) register is empty or when the CPU is reading the ICDRR (ICDR) register. When the data is stored in ICDRR (ICDR) and ICDRS, the SCL line is held low until the next read operation of ICDR to generate the communication waiting state. The last data must be received in the way shown below.

(a) For reception of multiple data (3 bytes or more)

- Store 2-byte data before receiving the last data in ICDRR (ICDR) and ICDRS.
- After setting the WAIT bit to 1, continuously read the 2-byte data mentioned above to make the buffer empty.
- Set the TRS bit to 1 (set the transmission mode) after IRIC interruption occurred at the falling edge of the eighth clock in the SCL for the last data reception. Set the ACKB bit to 1. Then clear the IRIC flag to output the ninth clock.
- After an IRIC interruption occurred for the last data reception, read the last data.
- Clear the WAIT bit, then the ACKB bit, and finally the IRIC flag to issue the stop condition.

(b) For reception of a datum (2 bytes or less)

- Set the WAIT bit to 1 before starting the data reception.
- Read the ICDRR (ICDR) register for the dummy run to start the data reception.
- Clear the IRIC flag after IRIC interruption occurred at the falling edge of the eighth clock in the SCL to output the ninth clock of the SCL.
- The data reception completes at the rising edge of the ninth clock.
- Read the ICDRR (ICDR) register to receive the data.

- Set the TRS bit to 1 (set the transmission mode) after IRIC interruption occurred at the falling edge of the eighth clock in the SCL for the second byte data reception. Set the ACKB bit to 1. Then clear the IRIC flag to output the ninth clock.
- After an IRIC interruption occurred for the last data reception, read the last data.
- Clear the WAIT bit, then the ACKB bit, and finally the IRIC flag to issue the stop condition.

For an example for the master reception, refer to section 4 “Example Applications for the H8S series”.

Slave operation:

In this I^2C module, the data register is a double-buffer configuration (ICDRS and ICDRR/ICDR). Therefore after a slave address which is the first data, the second data can be continuously received. First, a slave address after the start condition by the master device is input to the buffer (ICDRS), and the buffer is compared to the value of the slave address register (SAR or SARX). When addresses match, an acknowledge is returned to the master device at the ninth clock and the address data is loaded in the data register (ICDRR/ICDR). At this time, if the IRIC flag is set and an I^2C bus interrupt is enabled (IEIC = 1), an interrupt occurs. When addresses do not match, the address data is not loaded in ICDRR/ICDR and a wait state is entered in slave mode.

The eighth bit in the slave address phase means an R/W bit. When this bit is 1, subsequent operations seen from the slave side are in transmit mode. When this bit is 0, subsequent operations are in receive mode. The eighth bit is automatically reflected to the TRS bit.

When the TRS bit is 0, slave reception mode is still entered. ICDRS is now empty, therefore the next data is received continuously by outputting the SCL clock of the master device. When an acknowledge is returned to the master device at the ninth clock and the CPU reads slave address data from ICDR, data is shifted from ICDRS to ICDRR/ICDR. At this time, if the IRIC flag is set to 1 and an I^2C bus interrupt is enabled (IEIC = 1), an interrupt occurs. Then ICDRS is empty again and the next data is received continuously.

In the operation described above, if the I^2C bus interrupt processing is delayed since another interrupt processing is executed, and the CPU does not read the previously received data from ICDR (internal RDRF flag = 1), the next data is held by ICDRS at the end of the reception, the SCL is driven low, and the communication enters a waiting state for the master device. Therefore the received data is protected. The receive end interrupt of the first data is erased by the receive end interrupt of the second data. After the CPU reads the first data in ICDRR/ICDR, the second data in ICDRS is immediately shifted to ICDRR/ICDRS. Then IRIC is set again. When the I^2C bus interrupt is enabled (IEIC = 1), an interrupt occurs. A procedure for interrupts in slave reception is described below.

Example of procedure for interrupts in slave reception (H8S series)

(a) Confirms the contents of the status register (ICSR).

— Confirms the slave address matching (AAS or AASX = 1).
— Detects the stop condition (STOP = 1).
— Detects the error stop condition (ESTOP = 1).
— Detects the arbitration lost (AL = 1).
— Detects the general call address (b'0000000) (ADZ = 1).

(b) Clears the IRIC flag.

(c) Reads ICDR and fetches data.

(d) Judges the TRS bit in ICCR and confirms the subsequent operation mode (receive/transmit mode) after the slave address is received (When TRS = 1, the subsequent operations are in slave transmission mode. The SCL is driven to low until the CPU sets data in ICDR to indicate the waiting state to the master device).

(6) Continuous issuing of instructions (H8 Series I^2C module) [H8 Series]

A program that continuously issues instructions for start condition issuing, data transmission/reception, and stop condition issuing often does not work well. This is because internal competition often occurs among the data transmission instructions and an instruction is ignored, when the generation for the start condition by setting the start condition instruction is delayed due to the instruction timing and the load on the bus line. Some programming notes are shown below.

(a) Timing for issuing the data transmission instruction after the start condition has been issued:
after the instruction for setting the start condition has been issued, insert a wait time of one clock for the data transfer rate if any before executing the data transmit instruction.

(b) To issue the stop condition after the start condition has been issued:
confirm that BBSY = 1 and that bus authority has been obtained.

(c) To change the communication mode after the start condition has been issued:
confirm that BBSY = 1 and that bus authority has been obtained.

(d) To set the start condition after the stop condition has been issued:
confirm that BBSY = 0 and that the bus has been released.

(e) To change the communication mode after the stop condition has been issued:
confirm that BBSY = 0 and that the bus has been released.

(f) To start the next data transmission/reception after the completion of the current data transmission/reception:
For data transmission: confirm the completion of data transfer (IRIC = 1) and clear the IRIC to 0; then write the next data to ICDR.
For data reception: confirm the completion of data transfer (IRIC = 1) and read the ICDR; then clear the IRIC to 0. When TRS = 0, reading the ICDR acts as a trigger for the next data reception. To read the last data, set the TRS to 1 and read the ICDR to receive the reception data.

(g) To set the start condition again after the completion of data transmission/reception (to issue start condition for re-transmission):
this operation is applied when the master transmission is exchanged with the master reception. Confirm first that the data transmission has been ended (IRIC = 1), then clear the IRIC to 0, and finally execute the instruction for setting the start condition.

(h) To issue the stop condition after the completion of data transmission/reception:
For master transmission: confirm the completion of data transmission (IRIC = 1) and clear the IRIC to 0; then issue the stop condition.
For master reception: confirm the completion of data reception (IRIC = 1) and set the TRS to 1 (master transmission mode); then read the final data. After that, clear the IRIC to 0 and issue the stop condition.

(7) Continuous issuing of instructions (H8S Series I^2C module) [H8S Series]

(a) Timing for issuing data transmission instruction after the start condition has been issued:
after the instruction for setting the start condition has been executed, confirm that the start condition has been generated, by checking the IRIC flag; then execute the data transmission instruction.

(b) To issue the stop condition after the start condition has been issued:
after the instruction for setting the start condition has been executed, confirm that the start condition has been generated by checking the IRIC flag. After confirming that BBSY = 1, issue the stop condition.

(c) To change the communication mode after the start condition has been issued:
after the instruction for setting the start condition has been executed, confirm that the start condition has been generated by checking the IRIC flag. After confirming that BBSY = 1 and that the bus right is acquired, change the communication mode.

(d) To issue the start condition after the stop condition has been issued:
confirm that BBSY = 0 and that the bus has been released.

(e) To change the communication mode after the stop condition has been issued:
confirm that BBSY = 0 and that the bus has been released.

(f) To start the next data transmission/reception after the completion of data transmission/reception:
For data transmission: confirm the completion of data transfer (IRIC = 1) and write the next datum to ICDR. To confirm that the next data transfer is completed, clear the IRIC flag to 0.
For data reception: confirm the completion of data transfer (IRIC = 1) and read the ICDR; then clear the IRIC to 0. As the buffer for data reception has a two-stage structure in the H8S series I^2C module, 2-byte-long data is continuously received after reading the ICDRR (ICDR). To terminate the data reception, you must change the TRS bit to 1 (transmitting mode) during the last data reception (during the time period between the rising edge of the SCL first clock and the rising edge of the ninth clock). How to set this TRS bit is described in section 2.4 (13).

(g) To issue the start condition again after the completion of data transmission/reception (to issue the start condition for re-transmission):
this operation is applied when the master transmission is exchanged with the master reception. First, confirm that the data transmission has been ended (IRIC = 1), then clear the IRIC to 0, and finally execute the instruction for issuing the start condition.

(h) To issue the stop condition after the completion of data transmission/reception:
For master transmission: confirm the completion of data transmission (IRIC = 1) and clear the IRIC to 0; then issue the stop condition.
For master reception: confirm the completion of data reception (IRIC = 1) and set the TRS to 1 (master transmission mode); then read the final data. After that, clear the IRIC to 0 and issue the stop condition.

(8) Notes on re-sending the start condition [H8 Series, H8S Series]

When data is going to be transferred after the restart condition has been issued, the transfer instruction for the next byte should be executed by confirming that the SCL rose (point (A) in figure 2.5 after issuing the restart condition.

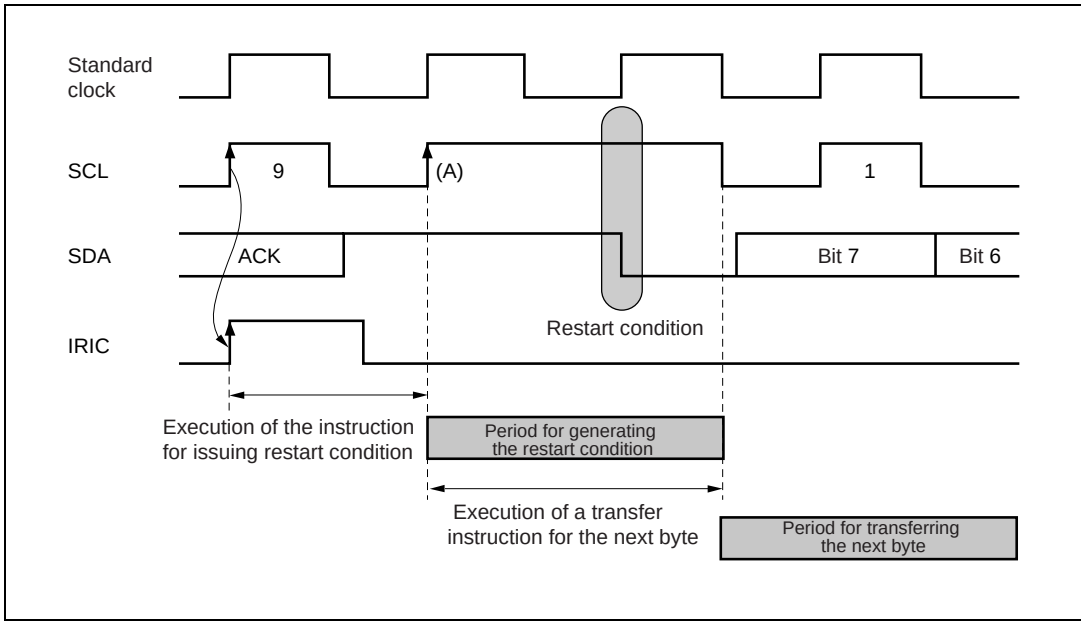


Figure 2.5 Execution Timing for Transfer Instruction for the Next Byte in the case of Resending the Start condition

The execution takes place as follows:

In the I^2C bus, the waiting state of a transfer operation in the case of the bus-occupied state is shown by SCL = low and SDA = high. Therefore, the instruction for issuing

the restart condition should be executed after confirming that SCL = low. Then confirm that the SCL = high (because the SCL is changed from low to high by the generation of the restart condition) and execute a transfer instruction for the next byte. In the H8S series, when the restart condition is satisfied, an interrupt is generated. Then the transfer instruction for the next byte must be executed.

(9) Confirmation of the coincidence of slave addresses [H8 Series, H8S Series]

Each bit of a slave address that was transmitted from the master device is compared with the corresponding bit of the SAR (in the H8S series I^2C module, two slave addresses, SAR and SARX, are available). If the slave address matches the SAR, the AAS bit (in the H8S series I^2C module: AAS or AASX bit) is set, and you can thus know that this device is the slave device that was specified by the master device in the IRIC interruption at the rising edge of the ninth SCL clock.

(10) Recognition of general call address [H8 Series, H8S Series]

The master device uses the general call address H'00 to specify all the I^2C devices as slave devices. The I^2C module sets the ADZ flag to 1 after recognizing the general call address. This flag is confirmed during the IRIC interruption at the rising edge of the ninth SCL clock.

(11) Recognition and setting of the acknowledge bit [H8 Series, H8S Series]

A data transmitting device receives the acknowledge bit from the data receiving device at the ninth SCL clock. This value is loaded in the ACKB bit and can be confirmed during the IRIC interruption at the rising edge of the ninth SCL clock. The data receiving device (TRS = “0”) outputs the value set in the ACKB bit to the SDA line at the ninth SCL clock. Note that when the TRS bit is set to 1, the value set in the ACKB bit in transmit mode is output. There are two internal ACKB bits according to whether the TRS bit is set to 1 or cleared to 0.

(12) Setting the transmit/receive mode in slave operation [H8 Series, H8S Series]

The R/$\overline{W}$ bit is automatically reflected to the TRS bit. If the R/$\overline{W}$ bit is 1 (read operation from the viewpoint of the master device) after the slave address sent out from the master device, the TRS bit is automatically set to 1 and slave transmit mode is entered.

(13) Wait operation [H8 Series, H8S Series]

A wait can be inserted between the eighth and ninth SCL clocks by setting the WAIT bit to 1 in master mode. An I^2C module holds the SCL line low after outputting the eighth clock. The ninth clock is sent out by clearing the IRIC flag to 0. In an I^2C bus and SMBus, a protocol that does not return an acknowledgment to slave devices upon receiving the last data in master operation is also available. Changing the ACKB bit from 0 to 1 by stopping the SCL clock at the eighth clock using this wait operation makes it easy to control the acknowledge bit.
This wait operation can be applied to the master receiving operation in a byte-wise manner in the

I²C module for the H8S series. The SCL clock can be stopped because the transmit mode becomes valid at the output timing of the SCL ninth clock after the IRIC flag was cleared by setting the transmit mode (TRS = 1) during this wait operation. Refer to (f) in section 2.4 (5) and 2.4 (7).

(14) How to confirm the number of transferred bits [H8 Series, H8S Series]

The bits BC2 to BC0 in the ICMR register are the bit counter that controls the number of SCL clocks. This counter decrements by 1 with each output of a clock. Reading the counter bits enables you to know how many bits were sent out. Writing back a value to the counter bits, however, needs special care. For example, when the same value as before is written back to the bits BC2 to BC0 immediately after the SCL clock has been output by the I²C module, an excess SCL clock is output. This generates a discrepancy among the bits for the slave device.

(15) Clearing the bits AL, AAS (AASX), and ADZ [H8 Series, H8S Series]

The bits AL (arbitration lost flag), AAS (AASX) (slave address recognition flag), and ADZ (general call address recognition flag) can be cleared by writing 0 to the respective bit after reading it. Reading from or writing to the ICDR automatically clears bits AAS and ADZ. Detecting the start condition automatically clears the AASX bit.

(16) Bus arbitration [H8 Series, H8S Series]

The I²C bus corresponds to multiple masters and has the structure for bus arbitration (refer to figure 1.9 for details). When multiple master devices simultaneously issue a start condition, each device compares the data of the SDA line and the internal SDA data at the rising edge of the SCL line clock. If these data are different from each other, the device stops the driving of the bus. In other words, the device that continues to output the low level to the SDA line until the final time can become the master device.
This I²C module sets the AL flag to 1 and turns the bus output off when the bus right is lost (bus arbitration lost). Also this I²C module automatically changes the operation mode from master transmission to slave reception, because the master device that got the bus right may specify the H8 as a slave device. When the slave addresses match (AAS or AASX = 1), an interrupt occurs at the rising edge of the ninth clock of the SCL. Therefore the AL flag can be confirmed to be 1. When the slave addresses do not match, an interrupt occurs by detecting the stop condition. Then the AL flag can be confirmed to be 1. Figure 2.6 shows an example of this bus arbitration processing flow.

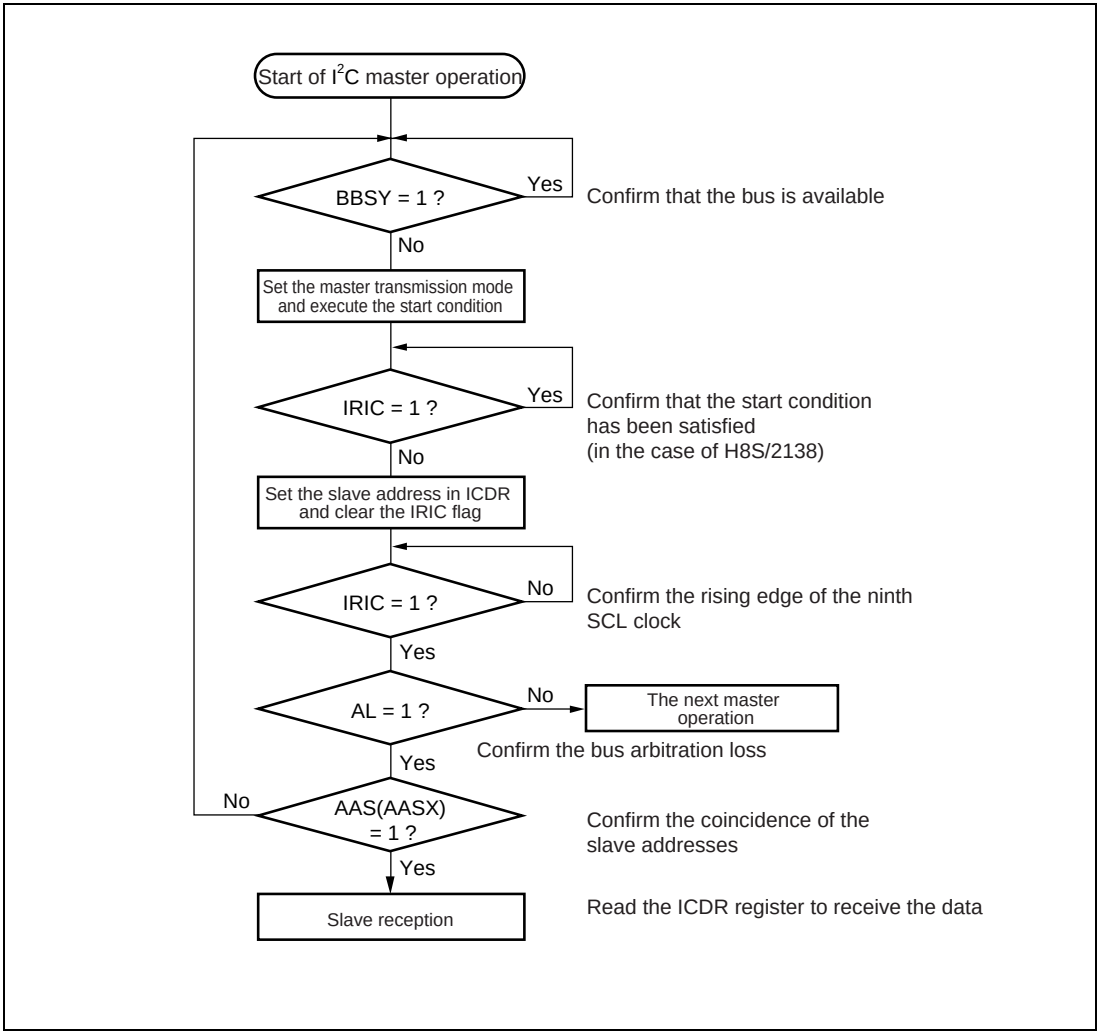


Figure 2.6 Bus Arbitration Processing

(17) The controllable ranges of the ICE bit [H8 Series, H8S Series]

The ICE bit controls:

(a) assignment of I/O addresses (changing the SAR or ICMR registers), and

(b) changing the pin functions of the SCL and SDA ports to general purpose I/O ports (in H8/3947 series: changing to the Hi-Z state).

Clearing the ICE bit can initialize the internal state of an H8S series I^{2}C module. This can be used to return the state to normal when the bus line of the microprocessor is stuck at low as a result of, for example, a communication malfunction.

(18) Using the serial communication interface together with the I²C bus [H8 Series] (H8/3337 Series and H8/3437 Series)

H8/3337 series and H8/3437 series have two serial communication interfaces (SCI0 and SCI1). SCI0 shares part of the register addresses with the I²C bus interface. The SCK1 pin (clock pin) of the SCI1 is also used as the SCL pin[*1]. When SCI (serial communication interface) is used as two channels and the I²C bus is used as one channel, care should be taken about the following points.

(a) As SCK1 shares pins with the SCL, use the SCI1 in the asynchronous mode (UART).

(b) When SCI0 and the I²C bus are used, set SCI0 to the state in which IICE = 0. The registers SMR and BBR share the addresses with the I²C bus interface register. These registers are used for the initial setting, so there is no need to set them again once they have been set unless the communication mode is changed. Then set the IICE to 1 and change to the accessing for I²C bus interface register to set the I²C bus.

Note: In the case of the H8/3217 series, two SCIs (one SCI in H8/3212) and two I²C bus interface (one I²C bus interface in H8/3202) are independently available.

(19) Using the serial communication interface together with the I²C bus [H8S Series] (H8S/2138 Series and H8S/2148 Series)

The H8S/2138 series and H8S/2148 series have three serial communication interfaces (SCI0, SCI1, and SCI2) and two I²C bus interfaces (IIC0 and IIC1). (SCI0 and SCI1 share part of the register addresses with the I²C bus interface). When SCI (serial communication interface) is used as three channels and I²C bus is used as two channels, care should be taken about the following points.

(a) As SCK (pins SCK0, SCK1, and SCK2) of the SCI shares pins with the I²C bus, use the SCI in the asynchronous mode (UART).

(b) When SCI and the I²C bus are used, set SCI0, SCI1, and SCI2 to the state in which IICE = 0. The registers SCMR and BRR are shared with the I²C bus interface register. These registers are used for initial setting. Therefore once these registers are set, resetting is not necessary unless the communication mode is changed. Then set IICE to 1 and change to the accessing for the I²C bus interface register to set the I²C bus.

2.5 Synchronization of the I²C Bus Communication

The format of the output port of the I²C bus is an open-drain. Therefore, the time taken to change from low to high depends on the load on a bus line. In the I²C bus specification, the rise time of the SCL line is decided to 1000 ns in normal mode (maximum data transfer rate is 100 kbps) and 300 ns in high-speed mode (maximum data transfer rate is 100 kbps). In the I²C bus, data must be fixed during the time period when the SCL line (clock line) is high. The actual data transfer rate is changed (synchronized communication) for the purpose of performing normal data transfer if the bus line load capacity and the value of the pull-up resistance connected between the bus line and power supply are inadequate.

Figure 2.7 shows an example of synchronized communication. This I²C module outputs the SCL clock on the SCL line in its master operation according to the internal standard clock that has the prescribed data transfer rate. Monitor the SCL line at the prescribed timing (refer to table 2.8) after the SCL line has risen from low to high to confirm that each bit of the SCL line has become high. If the rising edge of the SCL line is delayed or another device drives the SCL line to the low level, then the voltage level may not reach VIH (threshold voltage for recognizing the high level of I/O). In this case, delay the timing that drives the SCL line to the low level so that normal data communication takes place. After confirming that the SCL line has become high, drive the SCL line to the low level. As a result, the period of high level in the SCL line is prolonged and the data transfer rate becomes lower.

In other words, to get the prescribed data transfer rate, the pull-up resistance or bus line load capacity should be adjusted to adequate values.

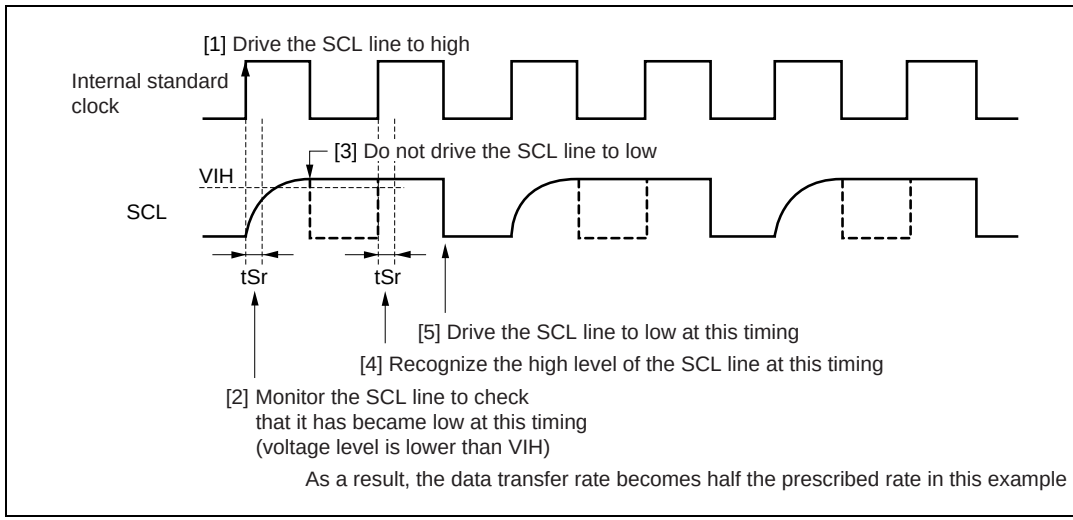


Figure 2.7 When the Rising Edge of the SCL Line is Delayed

Table 2.8 Monitoring Timing for Rising Edge of the SCL Line (H8S/2138 Series)

IICX bit	Monitoring timing for rising edge of the SCL Line tSr (tcyc expression)	Modes	tSr Time expression					
			Specification of I^2C bus (max)	ϕ=5MHz	8MHz	10MHz	6MHz	20MHz
0	7.5×tcyc*	Normal mode	1000	←	937	750	468	375
		High-speed mode	300	←	←	←	←	←
1	17.5×tcyc*	Normal mode	1000	←	←	←	←	875
		High-speed mode	300	←	←	←	←	←

Note: * The tcyc is the system clock period of this microprocessor.

(For reference only)

An example of the calculation for the pull-up resistance on the I^2C bus (H8S/2138 Series)

This is an example of the calculation for the pull-up resistance that connects the I^2C bus to the power supply.

- load capacity of the SCL line CB = 100 pF
- rise time of the SCL line tSr = 300 ns
- power supply voltage Vcc = 5.0 V
- voltage level for judging the high level of I/O VIH = Vcc x 0.7 = 3.5 V

using the calculation formula, Vcc x (1- exp(-t/(CB x R)) = VIH, gives the value of R as follows:
$\therefore R \cong 2.5\ k\Omega$.

2.6 Description of Data Transfer in H8/300 and H8/300L Series [H8 Series]

Data transfer should be done in the following conditions:

- Operation mode: addressing mode (a mode to recognize the slave address: FS = 0)
- Data transmission: MSB first (MLS = 0), no wait (WAIT = 0), and acknowledgement mode (a mode to recognize the acknowledgement: ACK = 0)

2.6.1 Master transmission

In the master transmission mode, the master device outputs the transmission clock (SCL line) and transmission data (SDA line), and slave devices return acknowledgments. Figure 2.8 describes the setting procedures and operation of the master transmission mode.

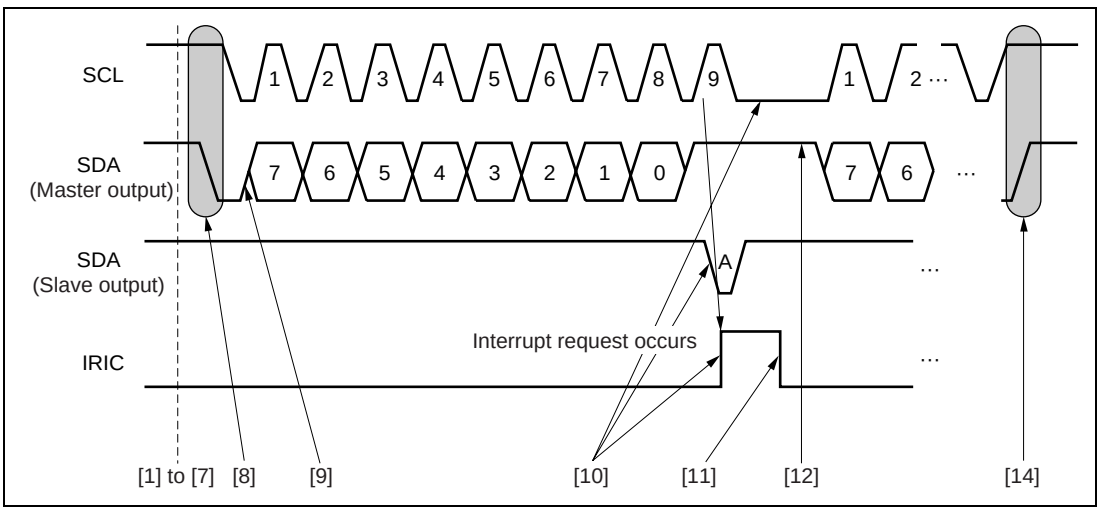


Figure 2.8 Operation Timing of the Master Transmission Mode (for MLS=WAIT=ACK=0)

Example of setting procedures of master transmission mode

[1] *

Software processing: Sets CKDBL.

Objective: Selects the system clock (ϕ) or clock of ½ division ratio (ϕ/2) for the peripheral clock.

[2] *

Software processing: Sets the IICE bit to 1.

Objective: Enables access to the I^2C bus interface registers.

Note: * This setting is only for the H8/3337, H8/3437, and H8/3217 series.

[3]

Software processing: Sets the SAR register. The uppermost 7 bits of the SAR are a slave address and the lowermost 1 bit (FS bit) are 0. This setting should be done in the case of a single master operation.

Objective: Sets the SAR register, because a slave mode may be set even in master mode when the system is in multi master mode.

[4]

Software processing: Sets the ICE bit to 1.

Objective: The SAR shares the address with the ICMR. An access to the SAR can thus be changed to an access to the ICMR by sharing the address. This change enables data transfer.

[5]

Software processing: Sets the ACKB bit.

Objective: Be sure to set the ACKB bit, because the mode automatically is shifted to slave reception by the bus arbitration even if the device is used in master mode.

[6]

Software processing: Clears the bits MLS, WAIT, and ACK to 0. Sets the bits CKS2 to 0, IICX, and IEIC so as to suit the operation mode.

Objective: Be sure to set the ACKB bit, because the mode automatically is shifted to slave reception by the bus arbitration even if the device is used in master mode.

[7]

Software processing: Reads the BBSY bit.

Objective: Confirms whether the bus has been released or is in use. If it has been released, BBSY equals 0. Then proceed to the next setting step.

[8]

Software processing: Sets the bits MSB and TRS to 1, writes 1 to the BBSY bit, and writes 0 to the SCP bit. The MOV instruction must be used to set these bits, because they must be simultaneously set.

Objective: Switches to the master transmission mode and sets the start condition.

Hardware processing: The SDA changes from high to low, when the SCL is high.

[9]

Software processing: Writes data to the ICDR register. The first data is a slave address and the R/W bit (= 0).

Objective: Starts the data transfer.

Hardware processing: The master device sequentially sends the transmission clock and the data written in the ICDR with the timing shown in figure 2.8.

[10]

Software processing: Sets the IRIC bit to 1 at the ninth clock when one byte of data has been transmitted. The master device receives an acknowledgment from the slave device, and sets the ACKB bit to 0. Fixes the SCL to low by synchronizing with the internal clock after transferring one frame of data.

Objective: The state in which the IRIC bit equals 1 means the end of a data transfer or bus arbitration. An interrupt request is issued to the CPU when the IEIC bit has been set to 1. The ACKB bit is used to confirm whether the acknowledge from the slave device has been received or not.

[11]

Software processing: Clears the IRIC bit.

Objective: Clears the IRIC bit for the subsequent data transmission.

[12]

Software processing: Writes data to the ICDR register.

Objective: Starts the data transfer.

[13]

Software processing: Repeats procedures [10] to [12].

Objective: Continues to transmit data.

[14]

Software processing: Writes 0 to the bits BBSY and SCP in the ICSR register. The MOV instruction must be used to set these bits, because they must be simultaneously set.

Objective: Issues the stop condition to terminate the transmission.

Hardware processing: The SDA changes from low to high, when the SCL is high.

2.6.2 Master Reception

In the master reception mode, the master device outputs the reception clock (SCL line) and receives data from slave devices. The master device returns acknowledgments to slave devices. In addressing mode, a slave address is firstly output with master transmission mode. The operation is the same as shown in “2.6.1 Master transmission mode” when the data transmission subsequently takes place. When data is going to be received, the mode should be switched to master reception after the first frame (one byte of data including the slave address) has been transferred. Figure 2.9 describes the setting procedures and operation of the master reception mode.

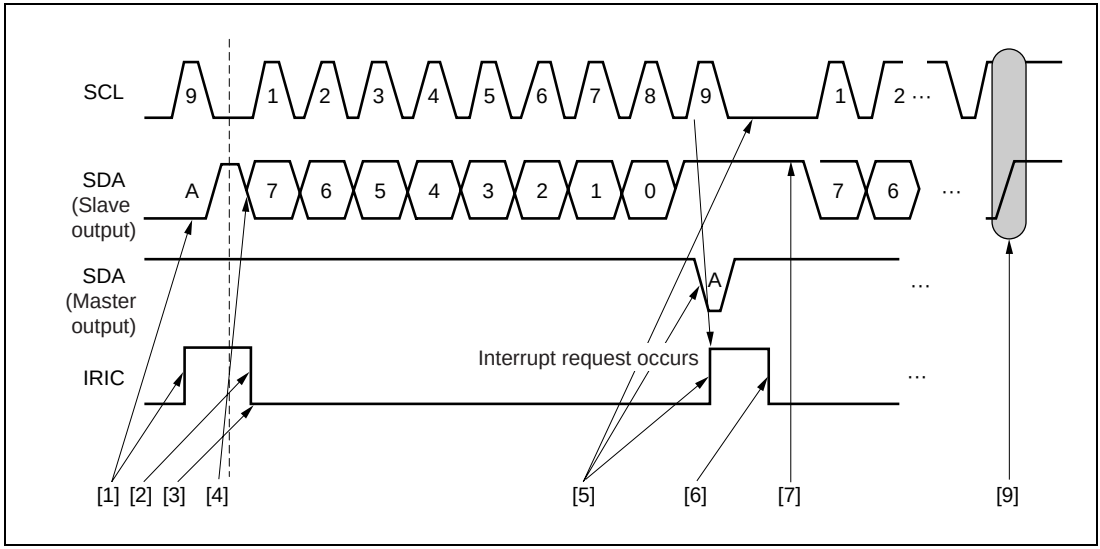


Figure 2.9 Operation Timing of the Master Reception Mode (for MLS=WAIT=ACKB=0)

Example of setting procedures of master reception mode

[1]

Hardware processing: The master device sets the start condition in the master transmission mode, and sends out the first byte including the slave address. The IRIC bit is set to 1 at the ninth clock. The master device receives an acknowledge from the slave device, and sets the ACCB bit to 0.

Objective: The state in which IRIC = 1 means the matching of the slave address.

[2]

Software processing: Clears the IRIC bit by the software.

Objective: Prepares for the subsequent data reception.

[3]

Software processing: Sets the TRS bit to 0.

Objective: Switches to the master reception mode.

[4]

Software processing: Reads the ICDR register (dummy reading).

Objective: This reading starts the reception of data.

Hardware processing: The master device outputs the reception clock by synchronizing with the internal clock and receives data.

[5]

Hardware processing: Sets the IRIC bit to 1 at the ninth clock, when one-byte data reception has ended. The master device simultaneously makes the SDA low and returns an acknowledgment. After transferring the one-frame data, the SCL is automatically fixed to low by synchronizing with the internal clock.

Objective: The state in which the IRIC bit equals 1 means the end of a data transfer. An interrupt request is issued to the CPU when the IEIC bit has been set to 1.

[6]

Software processing: Clears the IRIC bit to 0 by the software

Objective: Prepares the subsequent data reception

[7]

Software processing: Reads the ICDR register

Objective: The subsequent data reception is started by synchronizing with the internal clock. Set the ACKB bit to 1 before starting data reception, when an acknowledgment is not returned after the reception of the last byte.

[8]

Software processing: Repeats procedures [5] to [7]

Objective: Continues to receive data

[9]

Software processing: To stop the data reception, set the TRS bit to 1 and write 0 to bits BBSY and SCP after reading the ICDR register.

Objective: Switches the communication mode to the transmission mode so that the data is not received again. Issues the stop condition after releasing the SCL and SDA lines by reading the ICDR register.

Hardware processing: The SDA changes from low to high, when the SCL is high.

2.6.3 Slave Reception

In the slave reception mode, the master device outputs the transmission clock and transmission data, and slave devices receive the data and return acknowledgments.
Figure 2.10 describes the setting procedures and operation of the slave reception mode.

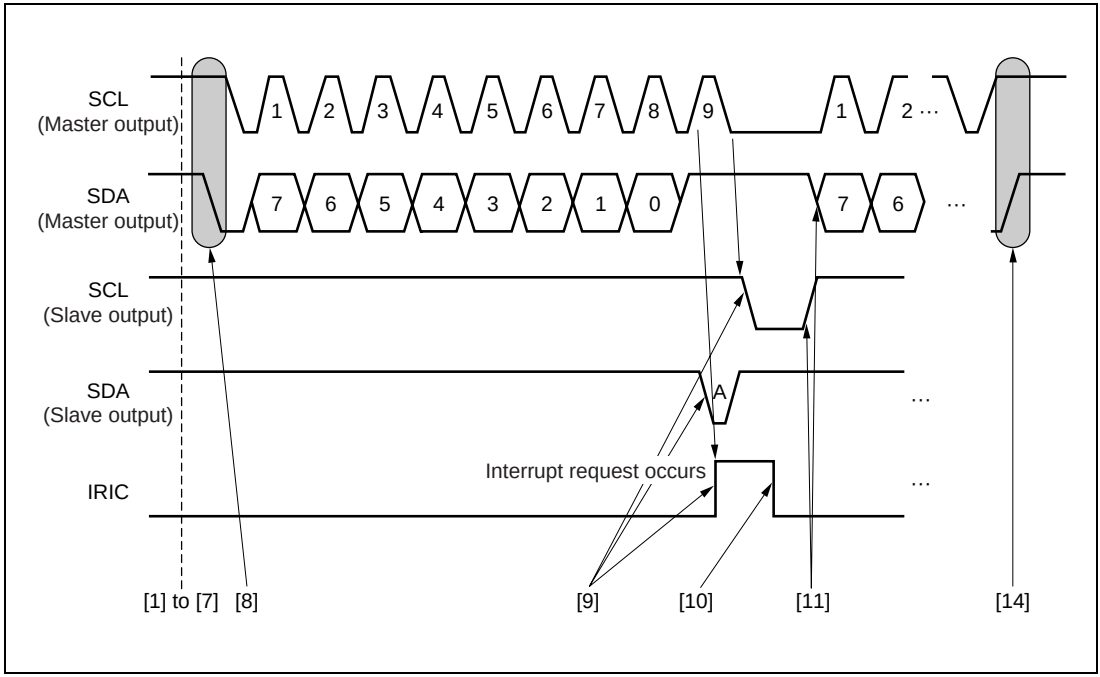


Figure 2.10 Operation Timing of the Slave Reception Mode (for MLS=WAIT=ACKB=0)

Example of setting procedures of slave reception mode

[1]

Software processing: Sets CKDBL.

Objective: Selects the system clock (ϕ) or clock of ½ division ratio (ϕ/2) for the peripheral clock.

[2]

Software processing: Sets the IICE bit to 1.

Objective: Enables access to the I^2C bus interface registers.

[3]

Software processing: Sets the SAR register. Writes the slave address to the uppermost 7 bits of the SAR, and 0 to the lowermost 1 bit (FS bit) (in addressing format).

Objective: Assigns an address to the slave device because the mode is an addressing mode.

[4]

Software processing: Sets the ICE bit to 1.

Objective: The SAR shares the address with the ICMR. An access to the SAR can thus be changed to an access to the ICMR by sharing the address. This change enables data transfer.

[5]

Software processing: Clears the bits MLS, WAIT, and ACK to 0. Sets the bits CKS2 to CKS0, IICX, and IEIC.

Objective: Sets the MSB first mode with the MLS bit, the no-wait mode with the WAIT bit, and the acknowledgement mode with the ACK bit. Defines the transfer clock frequency by the combination of the bits CKS2 to CKS0, and IICX. The IEIC bit defines the interrupt request of the I^2C bus interface as being enabled or disabled.

[6]

Software processing: Sets the bits MST and TRS to 0.

Objective: Sets the slave reception mode.

[7]

Software processing: Sets the ACKB bit to 0.

Objective: Sets the ACKB bit to 0 so that the master device will return an acknowledgment after receiving the data.

[8]

Hardware processing: After the start condition that was issued by the master device has been detected, the BBSY bit is set to 1.

Objective: Shows that the bus is in use (The master device outputs the first byte).

[9]

Hardware processing: The slave device confirms the matching of the slave address by reading the first byte after the start condition, and sets the IRIC bit to 1 at the ninth clock. It simultaneously makes the SDA low and returns an acknowledgment. It fixes the SCL to low from the falling edge of the ninth reception clock to the moment of reading data into the ICDR.

Objective: The state in which the IRIC bit equals 1 means the matching of the slave address. An interrupt request is issued to the CPU when the IEIC bit has been set to 1.

[10]

Software processing: Clears the IRIC bit by software.

Objective: Prepares for the subsequent data reception.

[11]

Software processing: Reads the ICDR register.

Objective: The slave device releases the SCL line, and the subsequent data reception starts.

[12]

Software processing: Repeats procedures [9] to [11].

Objective: Continues to receive data.

[13]

Software processing: The SDA changes from low to high when the SCL is high in response to the stop condition issued from the master device, and the BBSY bit is automatically cleared to 0 after the stop condition has been detected.

Objective: Terminates the data reception.

2.6.4 Slave Transmission

In slave transmission mode, a slave device outputs the reception data. The master device outputs the reception clock and returns an acknowledgment to the slave device.
In addressing mode, a slave address is first transferred from the master device to the slave device. At that time, the operation mode of the slave device is thus set to slave reception. The operation is the same as shown in “2.6.3 Slave reception” when the data reception subsequently takes place. When the slave device is going to transmit data to the master device, the mode should be switched to slave transmission mode.
Figure 2.11 describes the setting procedures and operation of slave transmission mode.

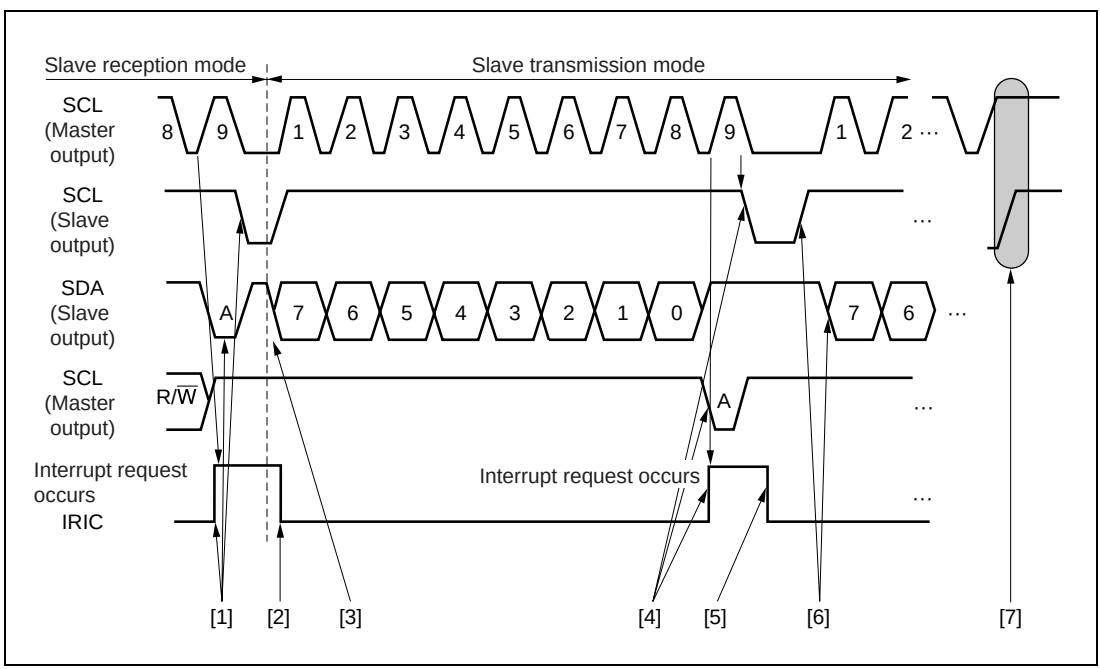


Figure 2.11 Operation Timing of Slave Transmission Mode (for MLS=WAIT=ACK=0)

Example of setting procedures of slave transmission mode

[1]

Hardware processing: The slave device confirms that the slave address matches by reading the first byte after detecting the start condition, and sets the IRIC bit to 1 at the ninth clock. The slave device simultaneously sets the SDA line to low and returns an acknowledgment. When the R/$\overline{W}$ bit (the eighth bit of the received data) is 1, the TRS bit is set to 1 and the operation mode automatically switches to slave transmission mode. The slave device fixes the SCL line to low from the falling edge of the ninth transmission clock to the start of writing data to the ICDR.

Objective: When IRIC bit equals 1, it means the slave address matches.

[2]

Software processing: Clears the IRIC bit to 0.

Objective: Prepares the subsequent data transmission.

[3]

Software processing: Writes the data in the ICDR register.

Objective: Starts the data transmission.

Hardware processing The slave device releases the SCL line by changing it to high and sequentially sends the data written in the ICDR according to the clock that is output by the master device with the timing shown in figure 2.8.

[4]

Hardware processing: After one byte of data has been transmitted, sets the IRIC bit to 1 at the rising edge of the ninth clock. The slave device receives an acknowledgment from the master device, and sets the ACKB bit to 0. The slave device automatically fixes the SCL line to low during the period from the falling edge of the ninth transmission clock to the start of writing data to the ICDR.

Objective: The state in which the IRIC bit equals 1 means the end of a data transfer. An interrupt request is issued to the CPU when the IEIC bit has been set to 1. The ACKB bit indicates whether or not the acknowledgment has been received from the master device.

[5]

Software processing: Clears the IRIC bit by software.

Objective: Prepares the subsequent data transmission.

[6]

Software processing: Writes the subsequent transmission data to the ICDR register.

Objective: The slave device releases the SCL line by changing it to high and starts the data transmission.

[7]

Software processing: Repeats procedures [4] to [6].

Objective: Continues the data transmission.

[8]

Software processing: Writes H'FF in the ICDR register.

Objective: Releases the SCL line so that the master device can issue the stop condition.

Hardware processing: The SCL line is released and allowed to go high. The SDA line changes from low to high when the SCL line is high by the issuing of the stop condition from the master device, and the BBSY bit is automatically cleared to 0 after the stop condition is detected.

2.7 Description of Data Transfer in H8S Series (H8/2138 Series) [H8S Series]

2.7.1 Master Transmission

In the master transmission mode using the I^2C bus format, the master device outputs the transmission clock and transmission data, and slave devices return acknowledgments. The setting procedures and operation of the master transmission mode are described below.

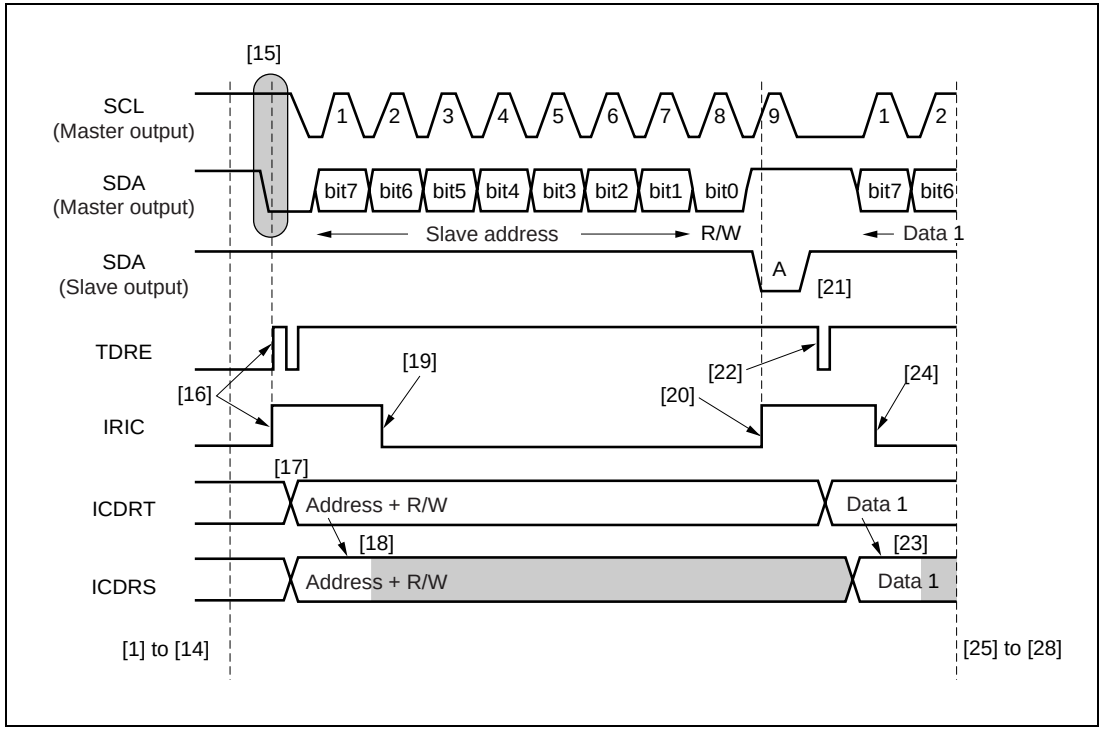


Figure 2.12 Operation Timing of Master Transmission Mode (for MLS=WAIT=0)

Example of setting procedures of master transmission mode

[1] Initial setting 1

Software setting: Clears the MSTP4 or MSTP3 bit in the MSTPCRL to 0.

Objective: Cancels the module stop mode of IIC channel 0 or IIC channel 1.

[2] Initial setting 2

Software setting: Sets the IICE bit in the STCR to 1.

Objective: Enables the CPU to access the data register and control register of the I^2C bus interface.

[3] Initial setting 3

Software setting: Sets the DDCSWR.

Objective: Selects enable/disable for the automatic switching function between format-less and I^2C bus format in IIC channel 0.
Selects format-less or I^2C bus format in IIC channel 0.
Selects enable/disable for interrupt requests to the CPU when automatic switching of the format takes place in IIC channel 0.

[4] Initial setting 4

Software setting: Clears the ICE bit in the ICCR to 0.

Objective: Enables access to the SAR and SARX.

[5] Initial setting 5

Software setting: Sets the SAR and SARX.

Objective: Sets the SW bit in the DDCSWR, the transfer format, and the slave address.

Note: Sets the slave address, because slave mode may be set even in master mode when the system is in multi-master mode.

[6] Initial setting 6

Software setting: Sets the ICE bit in the ICCR to 1.

Objective: Enables access to the ICMR and ICDR.
Puts the I^2C module in the transfer-enabled state.

[7] Initial setting 7

Software setting: Sets the ACKB bit in the ICSR.

Objective: Sets the acknowledgment data that is output during data reception.

Note: Be sure to set the ACKB bit, because the mode automatically shifts to slave reception if bus arbitration is lost even if the device was being used in master mode.

[8] Initial setting 8

Software setting: Sets the bits IICX1 or IICX0 in the STCR, and the bits CKS2 to CKS0 in the ICMR.

Objective: Selects the transfer clock frequency to be used.

[9] Initial setting 9

Software setting: Sets the bits MLS and WAIT in the ICMR to 0.

Objective: Sets the MSB-first mode and the no-wait mode in data transfer.

[10] Initial setting 10

Software setting: Sets the ACKE bit in the ICCR.

Objective: Selects one of the following two actions:
Transfer data continuously by ignoring the contents of the acknowledgment bit returned from the reception device in the I^2C bus format.
Perform the error processing by discontinuing the transfer operation when the acknowledgment bit equals 1.

[11] Initial setting 11

Software setting: Sets the IEIC bit in the IICR.

Objective: Selects enable/disable for interrupt request to the CPU from the I^2C bus interface.

[12] Confirmation of the bus state

Software setting: Reads the BBSY bit.

Objective: Confirms whether the bus is released or in use.

Hardware behavior: If the bus is released, the BBSY bit is equal to 0.

[13] Setting the master transmission mode

Software setting: Sets the bits MST and TRS in ICCR to 1.

Objective: Sets the operation mode of the I^2C bus interface to master transmission mode.

[14] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges whether the start condition was detected.

[15] Setting the start condition

Software setting: Sets the BBSY bit to 1, and clears the SCP bit to 0 in ICCR.

Objective: Sets the start condition.

Note: The MOV instruction must be used to set the BBSY bit to 1 and clear the SCP bit to 0, because these two bits must be simultaneously set.

Hardware behavior: The SDA changes from high to low, when the SCL is high.

[16] Confirmation that start condition has been satisfied

Software setting: Reads the IRIC bit.

Objective: Confirms that the start condition is detected from the bus line state.

Hardware behavior: If the start condition is detected, the bits IRIC and TDRE are equal to 1.

[17] Setting the slave address + R/W data

Software setting: Writes the slave address + R/W data to the ICDR.

Objective: Starts the data transfer.

Hardware behavior: If the data to be transmitted is written to the ICDR in transmission mode, the TDRE flag is cleared to 0.

[18] Data transfer from the ICDRT to the ICDRS

Hardware behavior: Clears the TDRE flag to 0.

Objective: Transfers data to be transmitted from the ICDRT to the ICDRS.

[19] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data transmission.

[20] Termination of one-byte data transmission

Hardware behavior: Sets the IRIC bit in the ICCR to 1 at the rising edge of the ninth transmission clock.

Objective: The state in which the IRIC bit equals 1 means the end of data transmission or that bus arbitration has been lost. An interrupt request is issued to the CPU when the IEIC bit in the ICCR has been set to 1.

[21] Confirmation of the acknowledgment

Software setting: Reads the ACKB bit in the ICSR.

Objective: Confirms the acknowledgment from the slave device.

Hardware behavior: Loads the acknowledgment, returned from the slave device, to the ACKB bit.

[22] Setting the transmit data

Software setting: Writes the transmit data to the ICDR.

Objective: Starts data transmission.

Hardware behavior: If the transmit data is written to the ICDR in transmission mode, the TDRE flag is cleared to 0.

[23] Data transfer from the ICDRT to the ICDRS

Hardware behavior: Clears the TDRE flag to 0.

Objective: Transfers transmit data from the ICDRT to the ICDRS.

[24] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data transfer.

[25] Termination of one-byte data transfer

Hardware behavior: Sets the IRIC bit in the ICCR to 1 at the rising edge of the ninth transmission clock.

Objective: The state in which the IRIC bit equals 1 means the end of data transmission or that bus arbitration has been lost. An interrupt request is issued to the CPU when the IEIC bit in the ICCR has been set to 1.

[26] Confirmation of the acknowledgment

Software setting: Reads the ACKB bit in the ICSR.

Objective: Confirms the acknowledgment from the slave device.

Hardware behavior: Loads the acknowledgment, returned from the slave device, to the ACKB bit.

[27] Continuation of the data transmission

Software setting: Repeats procedures [22] to [26].

Objective: Continues to transmit data.

[28] Issuing the stop condition

Software setting: Clears the bits BBSY and SCP to 0 in ICCR.

Objective: Issues the stop condition.

Note: The MOV instruction must be used to clear the bits BBSY and SCP to 0, because these two bits must be simultaneously set.

Hardware behavior: If the stop condition is detected from the bus line state, the TDRE flag is cleared to 0. If the bus is released, the BBSY bit is cleared to 0.

2.7.2 Master Reception

In master reception mode using the I^2C bus format, the master device outputs the reception clock, receives data, and returns an acknowledgment. The slave device transmits data. The setting procedures and operation of the master reception mode are described below.

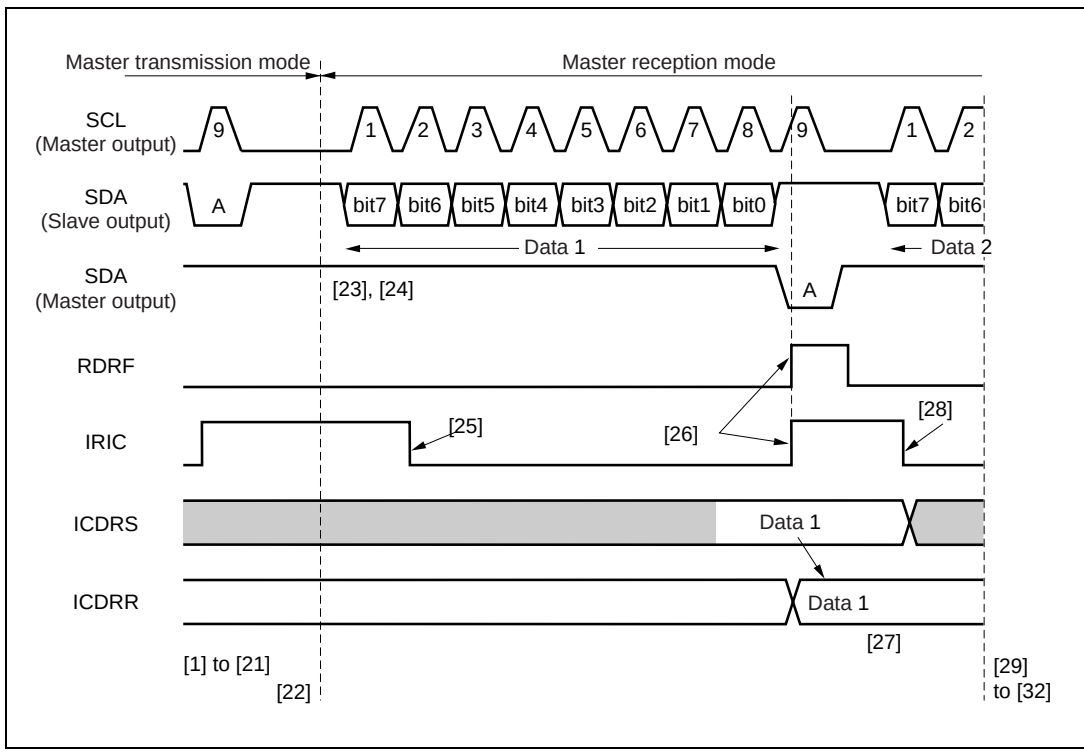


Figure 2.13 Operation Timing of Master Reception Mode (for MLS=WAIT=ACKB=0)

Example of setting procedures of master transmission mode

[1] Initial setting 1

Software setting: Clears the MSTP4 or MSTP3 bit in the MSTPCRL to 0.

Objective: Cancels the module stop mode of IIC channel 0 or IIC channel 1.

[2] Initial setting 2

Software setting: Sets the IICE bit in the STCR to 1.

Objective: Enables the CPU to access the data register and control register of the I^2C bus interface.

[3] Initial setting 3

Software setting: Sets the DDCSWR.

Objective: Selects enable/disable for the automatic switching function between format-less and I^2C bus format in IIC channel 0.
Selects format-less or I^2C bus format in IIC channel 0.
Selects enable/disable for interrupt request to the CPU when automatic switching of the format takes place in IIC channel 0.

[4] Initial setting 4

Software setting: Clears the ICE bit in the ICCR to 0.

Objective: Enables access to the SAR and SARX.

[5] Initial setting 5

Software setting: Sets the SAR and SARX.

Objective: Sets the SW bit in the DDCSWR, the transfer format, and the slave address.

Note: Sets the slave address, because slave mode may be set even in master mode when the system is in multi-master mode.

[6] Initial setting 6

Software setting: Sets the ICE bit in the ICCR to 1.

Objective: Enables access to the ICMR and ICDR.
Puts the I^2C module in the transfer-enabled state.

[7] Initial setting 7

Software setting: Sets the ACKB bit in the ICSR.

Objective: Sets the acknowledgment data that is output during data reception.

Note: Be sure to set the ACKB bit, because the mode automatically shifts to slave reception if bus arbitration is lost even if the device was being used in master mode.

[8] Initial setting 8

Software setting: Sets the bits IICX1 or IICX0 in the STCR, and the bits CKS2 to 0 in the ICMR.

Objective: Selects the transfer clock frequency to be used.

[9] Initial setting 9

Software setting: Sets the bits MLS and WAIT in the ICMR to 0.

Objective: Sets the MSB-first mode and the no-wait mode in data transfer.

[10] Initial setting 10

Software setting: Sets the ACKE bit in the ICCR.

Objective: Selects one of the following two actions:
Transfer data continuously by ignoring the contents of the acknowledgment bit returned from the reception device in the I^2C bus format.
Perform the error processing by discontinuing the transfer operation when the acknowledgment bit equals 1.

[11] Initial setting 11

Software setting: Sets the IEIC bit in the IICR.

Objective: Selects enable/disable for interrupt request to the CPU from the I^2C bus interface.

[12] Confirmation of the bus state

Software setting: Reads the BBSY bit.

Objective: Confirms whether the bus is released or in use.

Hardware behavior: If the bus is released, the BBSY bit is equal to 0.

[13] Setting the master transmission mode

Software setting: Sets the bits MST and TRS in ICCR to 1.

Objective: Sets the operation mode of the I^2C bus interface to master transmission mode.

[14] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the detection for the start condition.

[15] Setting the start condition

Software setting: Sets the BBSY bit to 1, and clears the SCP bit to 0 in ICCR.

Objective: Sets the start condition.

Note: The MOV instruction must be used to set the BBSY bit to 1 and clear the SCP bit to 0, because these two bits must be simultaneously set.

Hardware behavior: The SDA changes from high to low, when the SCL is high.

[16] Confirmation that the start condition has been satisfied

Software setting: Reads the IRIC bit.

Objective: Confirms that the start condition is detected from the bus line state.

Hardware behavior: If the start condition is detected, the bits IRIC and TDRE are equal to 1.

[17] Setting the slave address + R/W data

Software setting: Writes the slave address + R/W data to the ICDR.

Objective: Starts the data transfer.

Hardware behavior: If the transmit data is written to the ICDR in transmission mode, the TDRE flag is cleared to 0.

[18] Data transfer from the ICDRT to the ICDRS

Hardware behavior: Clears the TDRE flag to 0.

Objective: Transfers transmit data from the ICDRT to the ICDRS.

[19] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data transmission.

[20] Termination of one-byte data transmission

Hardware behavior: Sets the IRIC bit in the ICCR to 1 at the rising edge of the ninth transmission clock.

Objective: The state in which the IRIC bit equals 1 means the end of data transmission or that bus arbitration has been lost. An interrupt request is issued to the CPU when the IEIC bit in the ICCR has been set to 1.

[21] Confirmation of the acknowledgment

Software setting: Reads the ACKB bit in the ICSR.

Objective: Confirms the acknowledgment from the slave device.

Hardware behavior: Loads the acknowledgment, returned from the slave device, to the ACKB bit.

[22] Setting the master reception mode

Software setting: Clears the TRS bit in the ICCR to 0.

Objective: Switches from master transmission mode to master reception mode

[23] ACKB=0

Software setting: Clears the ACKB bit in the ICSR to 0.

Objective: Outputs 0 at the acknowledgment output timing.

[24] Dummy reading

Software setting: Reads the ICDR.

Objective: Starts the data reception.

[25] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data reception.

[26] Termination of one-byte data reception

Hardware behavior: Sets the IRIC bit in the ICCR and the RDRF flag to 1 at the rising edge of the ninth reception clock.

Objective: The state in which the IRIC bit equals 1 means the end of data transfer. An interrupt request is issued and sent to the CPU when the IEIC bit has been set to 1. Data reception will continue after setting the internal RDRF flag to 1, when the flag has been cleared to 0.

[27] Reading the received data

Software setting: Reads the ICDR.

Objective: Starts data reception.

Hardware behavior: Clears the RDRF flag to 0.

Note: Sets the ACKB bit to 1 before reading the ICDR when an acknowledgment is not returned after reception of the last byte.

[28] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data reception.

[29] Continuation of data reception

Software setting: Repeats procedures [26] to [28].

Objective: Continues to receive data.

[30] Termination of reception

Software setting: Sets the TRS bit in the ICCR to 1.

Objective: To terminate the data reception, sets the TRS bit in the ICCR to 1 before the rise up of the reception clock for the subsequent frame.

Hardware behavior: Sets the TDRE flag to 1.

[31] Reading the last data

Software setting: Reads the ICDR.

Objective: Reads the last byte of the reception data.

[32] Issuing the stop condition

Software setting: Clears the bits BBSY and SCP to 0 in ICCR.

Objective: Issues the stop condition.

Note: The MOV instruction must be used to clear the bits BBSY and SCP to 0, because these two bits must be simultaneously set.

Hardware behavior: If the stop condition is detected from the bus line state, the TDRE flag is cleared to 0. If the bus is released, the BBSY bit is cleared to 0.

2.7.3 Slave Reception

In slave reception mode using the I²C bus format, the master device outputs the transmission clock and transmission data, and slave devices return acknowledgments.
The setting procedures and operation of the slave reception mode are described below.

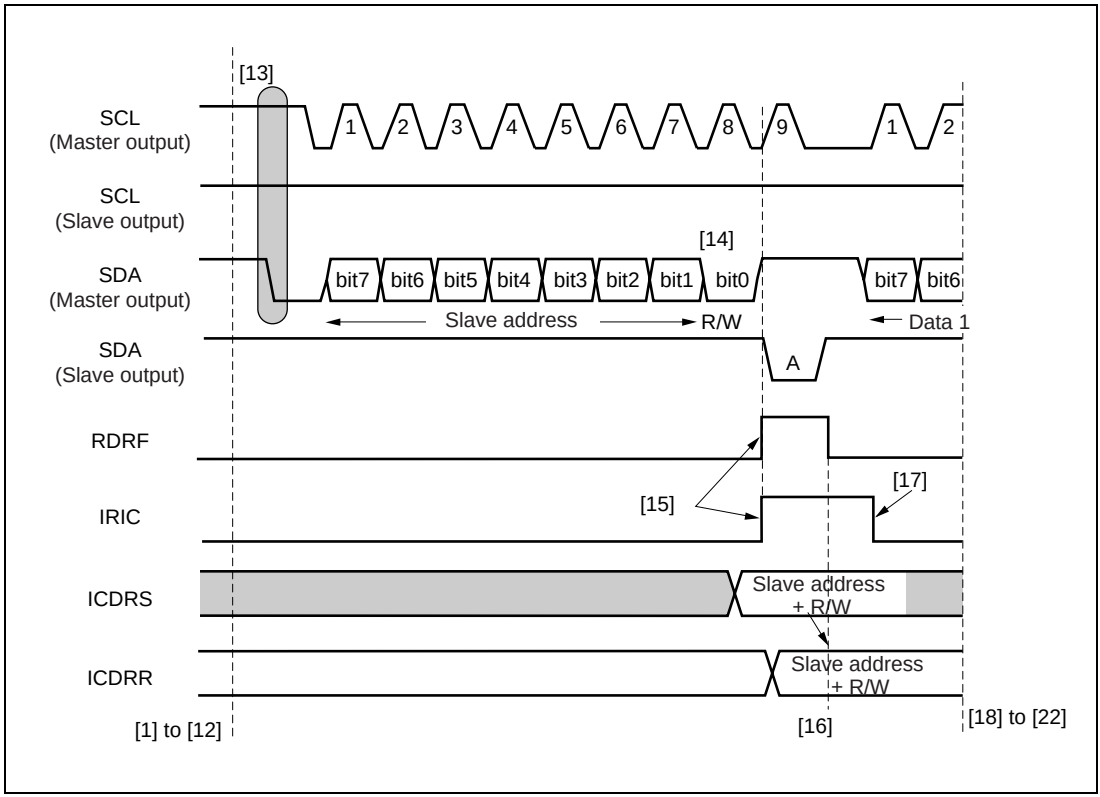


Figure 2.14 Operation Timing of Slave Reception Mode (for MLS=ACKB=0)

Example of setting procedures of the slave reception mode

[1] Initial setting 1

Software setting: Clears the MSTP4 or MSTP3 bit in the MSTPCRL to 0.

Objective: Cancels the module stop mode of IIC channel 0 or IIC channel 1.

[2] Initial setting 2

Software setting: Sets the IICE bit in the STCR to 1.

Objective: Enables the CPU to access the data register and control register of the I²C bus interface.

[3] Initial setting 3

Software setting: Sets the DDCSWR.

Objective: Selects enable/disable for the automatic switching function between format-less and I^2C bus format in IIC channel 0.
Selects format-less or I^2C bus format in IIC channel 0.
Selects enable/disable for interrupt request to the CPU when automatic switching of the format takes place in IIC channel 0.

[4] Initial setting 4

Software setting: Clears the ICE bit in the ICCR to 0.

Objective: Enables access to the SAR and SARX.

[5] Initial setting 5

Software setting: Sets the SAR and SARX.

Objective: Sets the SW bit in the DDCSWR, the transfer format, and the slave address.

[6] Initial setting 6

Software setting: Sets the ICE bit in the ICCR to 1.

Objective: Enables access to the ICMR and ICDR.
Puts the I^2C module in the transfer-enabled state.

[7] Initial setting 7

Software setting: Sets the ACKB bit in the ICSR.

Objective: Sets the acknowledgment data that is output during data reception.

Note: Be sure to set the ACKB bit, because the mode automatically shifts to slave reception if bus arbitration is lost even if the device was being used in master mode.

[8] Initial setting 8

Software setting: Sets the bits IICX1 or IICX0 in the STCR, and the bits CKS2 to 0 in the ICMR.

Objective: Selects the transfer clock frequency to be used.

[9] Initial setting 9

Software setting: Sets the bits MLS and WAIT in the ICMR to 0.

Objective: Sets the MSB-first mode and the no-wait mode in data transfer.

[10] Initial setting 10

Software setting: Sets the ACKE bit in the ICCR.

Objective: Selects one of the following two actions:
Transfer data continuously by ignoring the contents of the acknowledgment bit returned from the reception device in the I^2C bus format.
Perform the error processing by discontinuing the transfer operation when the acknowledgment bit equals 1.

[11] Initial setting 11

Software setting: Sets the IEIC bit in the IICR.

Objective: Selects enable/disable for interrupt request to the CPU from the I^2C bus interface.

[12] Initial setting 12

Software setting: Sets the bits MST and TRS to 0.

Objective: Sets the slave reception mode.

[13] Detecting the start condition

Hardware behavior: Sets the BBSY bit in the ICCR to 1.

Objective: Detects the start condition issued by the master device.

[14] Reception of the slave address

Hardware behavior: Clears the TRS bit in the ICCR to 0.

Objective: Acts as the slave device that is specified by the master device when the slave address has been matched at the first frame after the starting condition. When the eighth data (R/W) is equal to 0, the TRS bit in the ICCR remains 0 (unchanged), and slave reception operation takes place.

[15] Matching the slave address

Hardware behavior: The slave device sets the SDA to low and returns an acknowledgment at the ninth clock of the reception frame. The slave device simultaneously sets the IRIC bit in the ICCR and the RDRF flag to 1.

Objective: The state in which the IRIC bit equals 1 means the matching of the slave address. An interrupt request is issued to the CPU when the IEIC bit in the ICCR has been set to 1.

[16] Dummy reading

Software setting: Reads the ICDR (dummy reading).

Objective: Starts the data reception.

[17] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data reception.

[18] Termination of data reception

Hardware behavior: The slave device sets the SDA to low and returns an acknowledgment at the ninth clock of the reception frame. The slave device simultaneously sets the IRIC bit in the ICCR and the RDRF flag to 1.

Objective: The state in which the IRIC bit equals 1 means the termination of the data transfer. An interrupt request is issued and sent to the CPU when the IEIC bit in the ICCR has been set to 1.

[19] Reading the received data

Software setting: Reads the ICDR.

Objective: Reads the received data.

Hardware behavior: Clears the RDRF flag to 0 by reading the received data in the ICDR (ICDRR).

[20] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data reception.

[21] Continuation of the data reception

Software setting: Repeats procedures [18] to [20].

Objective: Continues to receive data.

[22] Termination of reception

Hardware behavior: The SDA changes from low to high when the SCL is high. Clears the BBSY bit in the ICCR to 0.

Objective: Detects the stop condition issued by the master device.

2.7.4 Slave Transmission

In slave transmission mode using the I^2C bus format, a slave device outputs the transmission data. The master device outputs the reception clock and returns acknowledgment. The setting procedures and operation of the slave transmission mode are described below.

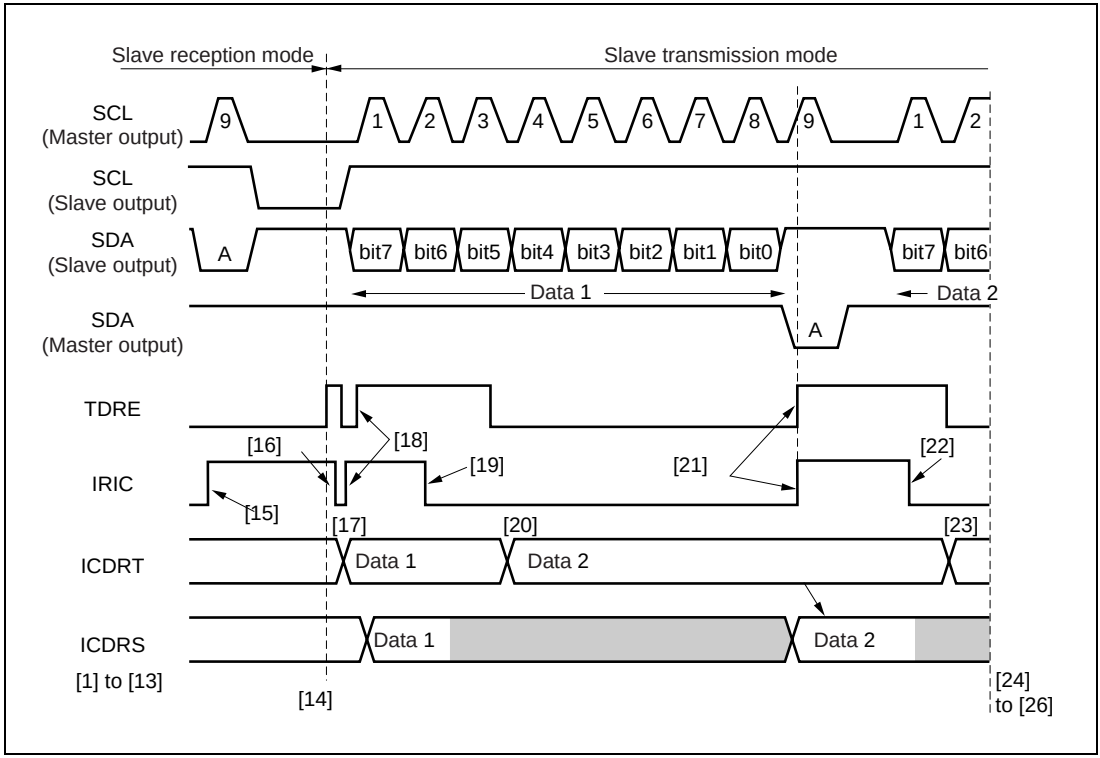


Figure 2.15 Operation Timing of Slave Transmission Mode (for MLS=0)

Example of setting procedures of the slave transmission mode

[1] Initial setting 1

Software setting: Clears the MSTP4 or MSTP3 bit in the MSTPCRL to 0.

Objective: Cancels the module stop mode of IIC channel 0 or IIC channel 1.

[2] Initial setting 2

Software setting: Sets the IICE bit in the STCR to 1.

Objective: Enables the CPU to access the data register and control register of the I^2C bus interface.

[3] Initial setting 3

Software setting: Sets the DDCSWR.

Objective: Selects enable/disable for the automatic switching function between format-less and I^2C bus format in IIC channel 0.
Selects format-less or I^2C bus format in IIC channel 0.
Selects enable/disable for interrupt request to the CPU when the automatic switching of the format takes place in IIC channel 0.

[4] Initial setting 4

Software setting: Clears the ICE bit in the ICCR to 0.

Objective: Enables access to the SAR and SARX.

[5] Initial setting 5

Software setting: Sets the SAR and SARX.

Objective: Sets the SW bit in the DDCSWR, the transfer format, and the slave address.

[6] Initial setting 6

Software setting: Sets the ICE bit in the ICCR to 1.

Objective: Enables access to the ICMR and ICDR.
Puts the I^2C module in the transfer-enabled state.

[7] Initial setting 7

Software setting: Sets the ACKB bit in the ICSR.

Objective: Sets the acknowledgment data that is output during data reception.

Note: Be sure to set the ACKB bit, because the mode automatically shifts to slave reception if bus arbitration is lost even if the device was being used in master mode.

[8] Initial setting 8

Software setting: Sets the bits IICX1 or IICX0 in the STCR, and the bits CKS2 to 0 in the ICMR.

Objective: Selects the transfer clock frequency to be used.

[9] Initial setting 9

Software setting: Sets the bits MLS and WAIT in the ICMR to 0.

Objective: Sets the MSB-first mode and the no-wait mode in data transfer.

[10] Initial setting 10

Software setting: Sets the ACKE bit in the ICCR.

Objective: Selects one of the following two actions:
Transfer data continuously by ignoring the contents of the acknowledgment bit returned from the reception device in the I^2C bus format.
Perform the error processing by discontinuing the transfer operation when the acknowledgment bit equals 1.

[11] Initial setting 11

Software setting: Sets the IEIC bit in the IICR.

Objective: Selects enable/disable for interrupt request to the CPU from the I^2C bus interface.

[12] Initial setting 12

Software setting: Sets the bits MST and TRS to 0.

Objective: Sets the slave reception mode.

[13] Detecting the start condition

Hardware behavior: Sets the BBSY bit in the ICCR to 1.

Objective: Detects the start condition issued by the master device.

[14] Reception of the slave address

Hardware behavior: Clears the TRS bit in the ICCR to 0, and sets the TDRE flag to 1.

Objective: Acts as the slave device that is specified by the master device when the slave address has been matched at the first frame after the starting condition. When the eighth data (R/W) equals 1, sets the TRS bit in the ICCR to 1, and automatically changes to slave transmission mode.

[15] Matching the slave address

Hardware behavior: The slave device sets the SDA to low and returns an acknowledgment at the ninth clock of the reception frame. The slave device simultaneously sets the IRIC bit in the ICCR to 1. The slave device fixes the SCL to low during the period from the falling edge of the transmission clock to the start of writing data to the ICDR.

Objective: The state in which the IRIC bit equals 1 means the matching of the slave address. An interrupt request is issued and sent to the CPU when the IEIC bit in the ICCR has been set to 1.

[16] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the data transfer from the ICDRT to the ICDRS.

[17] Writing the first byte of the transmission data

Software setting: Writes the first byte of the transmission data to the ICDR.

Objective: Starts the data transmission.

Hardware behavior: Clears the TDRE flag to 0.

[18] Data transfer from the ICDRT to the ICDRS

Hardware behavior: Sets the TDRE flag, the IRIC bit in the ICCR, and the IRTR in the ICSR to 1.

Objective: Transfers the data written in the ICDRT to the ICDRS.

[19] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data transmission.

[20] Writing the transmission data

Software setting: Writes the transmission data to the ICDR.

Objective: Starts the data transmission.

Hardware behavior: Clears the TDRE flag to 0.

[21] Termination of transmission

Hardware behavior: After one-frame data transmission ended, sets the IRIC bit in the ICCR to 1 at the rising edge of the ninth transmission clock. The slave device receives an acknowledgment from the master device, and stores it in the ACKB bit. The slave device automatically fixes the SCL to low during the period from the falling edge of the ninth transmission clock to the start of writing data to the ICDR.

Objective: The state in which the IRIC bit equals 1 means the end of a data transfer. An interrupt request is issued and sent to the CPU when the IEIC bit in the ICCR has been set to 1. The acknowledgment from the master device can be confirmed by reading the ACKB bit.

[22] Clearing the IRIC

Software setting: Clears the IRIC bit in the ICCR to 0.

Objective: Judges the termination of the data transmission.

[23] Writing the transmission data

Software setting: Writes the transmission data to the ICDR.

Objective: The slave device releases the SCL and allows it to go high, and starts the data transmission.

[24] Continuation of the data transmission

Software setting: Repeats procedures [21] to [23].

Objective: Continues to transmit data.

[25] Termination of transmission

Software setting: Clears the TRS bit in the ICCR to 0, and reads the ICDR (dummy reading).

Objective: Sets the slave reception mode by clearing the TRS bit to 0. Releases the SCL line by the dummy reading of the ICDR.

[26] Detecting the stop condition

Hardware behavior: The SDA changes from low to high when the SCL is high. Clears the BBSY bit in the ICCR to 0.

Objective: Detects the stop condition issued by the master device.

Section 3 Examples of Application to the H8/300 and H8/300L Series

3.1 System Specifications

The system specifications are described below. Figure 3.1 illustrates the system configuration.

- The system has a multi-master configuration comprising two masters and one slave. The H8/3434F, which has an on-chip flash memory, is used as a device.
- The 8-segment LED displays on its screen: ‘CPU1’ when switch-1 (SW1: master-1 side) is pressed and ‘CPU2’ when switch-2 (SW2: master-2 side) is pressed.
  (1) Master-1 sends H'01 to the slave when switch-1 is pressed, and the master-2 sends H'02 when the switch-2 is pressed.
  (2) The slave distinguishes the received data and displays on the screen of the 8-segment LED: ‘CPU1’ when the data is H'01 and ‘CPU2’ when the data is H'02.
- The transfer rate of the I^2C bus is 200 kbps.

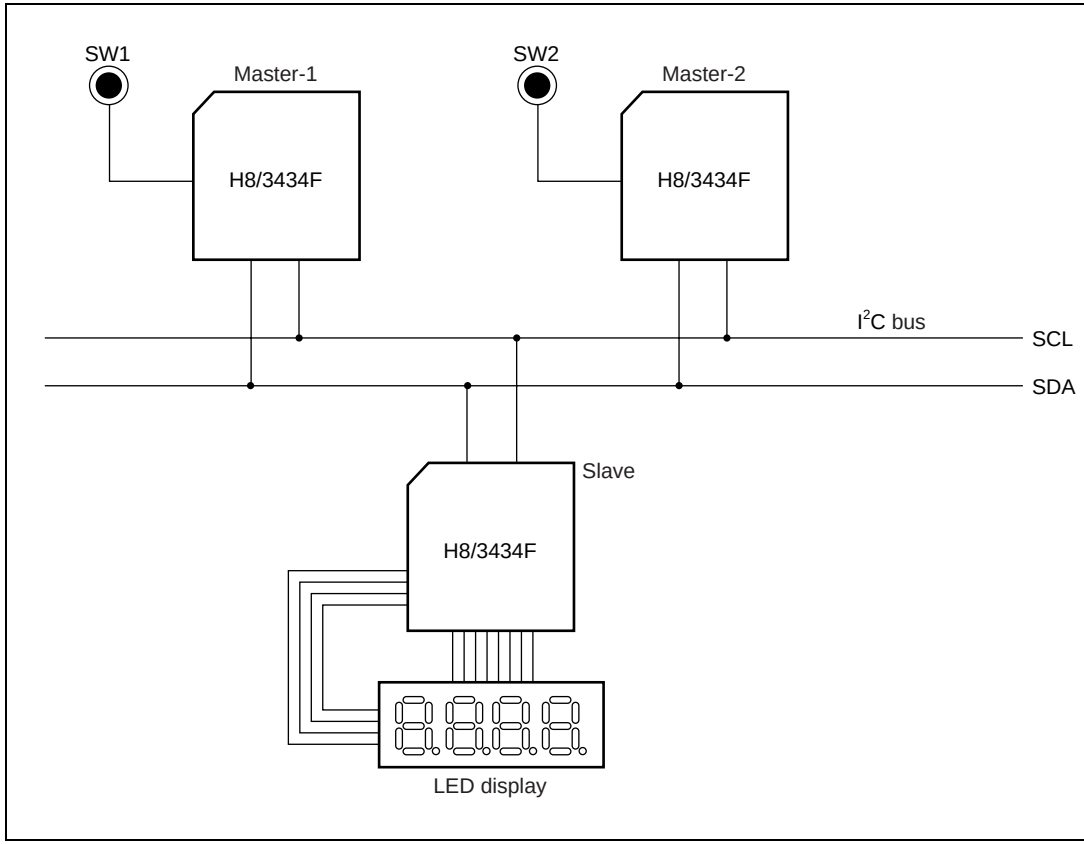


Figure 3.1 System for Evaluating the Multi-Master configuration

- In the on-chip I^{2}C bus interface of the H8/300 and H8/300L series, the adjustment procedures shown in figure 3.2 are performed as well as the communication adjustment procedures described in “1.4 Procedure for Communication Adjustment”. Each master device monitors the bus line at the falling edge of the SCL line, and switches off its output gates if the monitored level does not coincide with its own level.

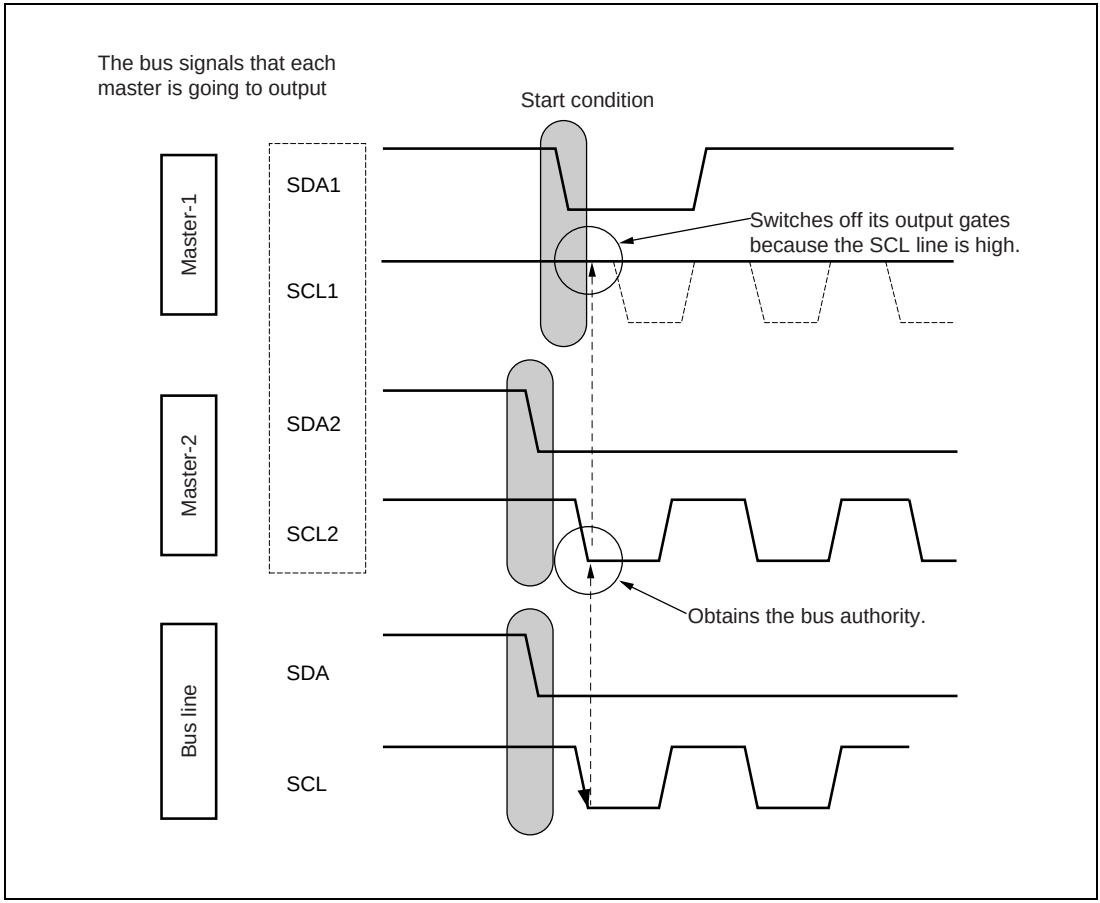


Figure 3.2 How to detect the bus arbitration

- When master-1 and master-2 start data transmission simultaneously (multi-master operation)
  (1) When a collision is detected, master-1 obtains the bus authority, because the period when the SDA line (data line) is low is longer for master-1 (transmitted data is H'01) than for master-2 (transmitted data is H'02). Refer to figure 3.3.

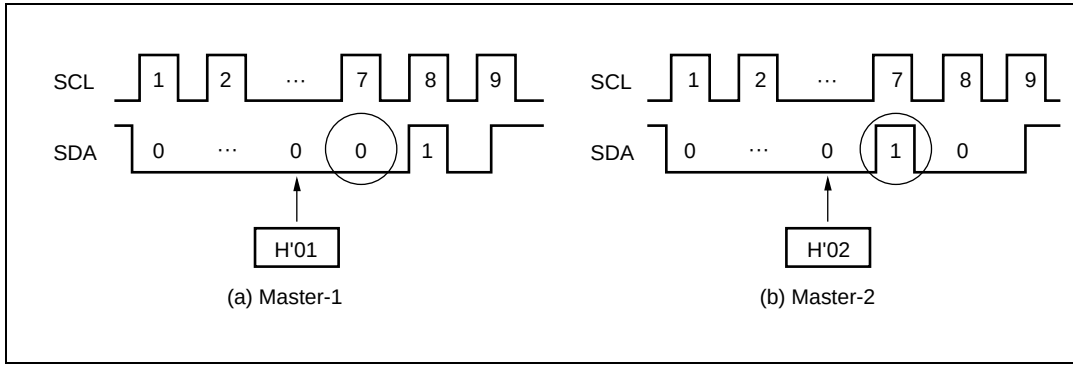


Figure 3.3 Why master-1 obtains the bus authority

(2) Master-2 loses the bus arbitration and automatically transits to slave reception mode. To use master-2 in master transmission mode again, the system must perform a reset. The data that was not transmitted must be written to the ICDR again. This system, therefore, calls the data transmission routine again regardless of the switch input, after confirming the bus arbitration loss of master-2.

3.2 Circuit for Multi-Master Evaluation System

Figure 3.4 illustrates the circuit diagram for evaluating a multi-master system that has the multi-master configuration.

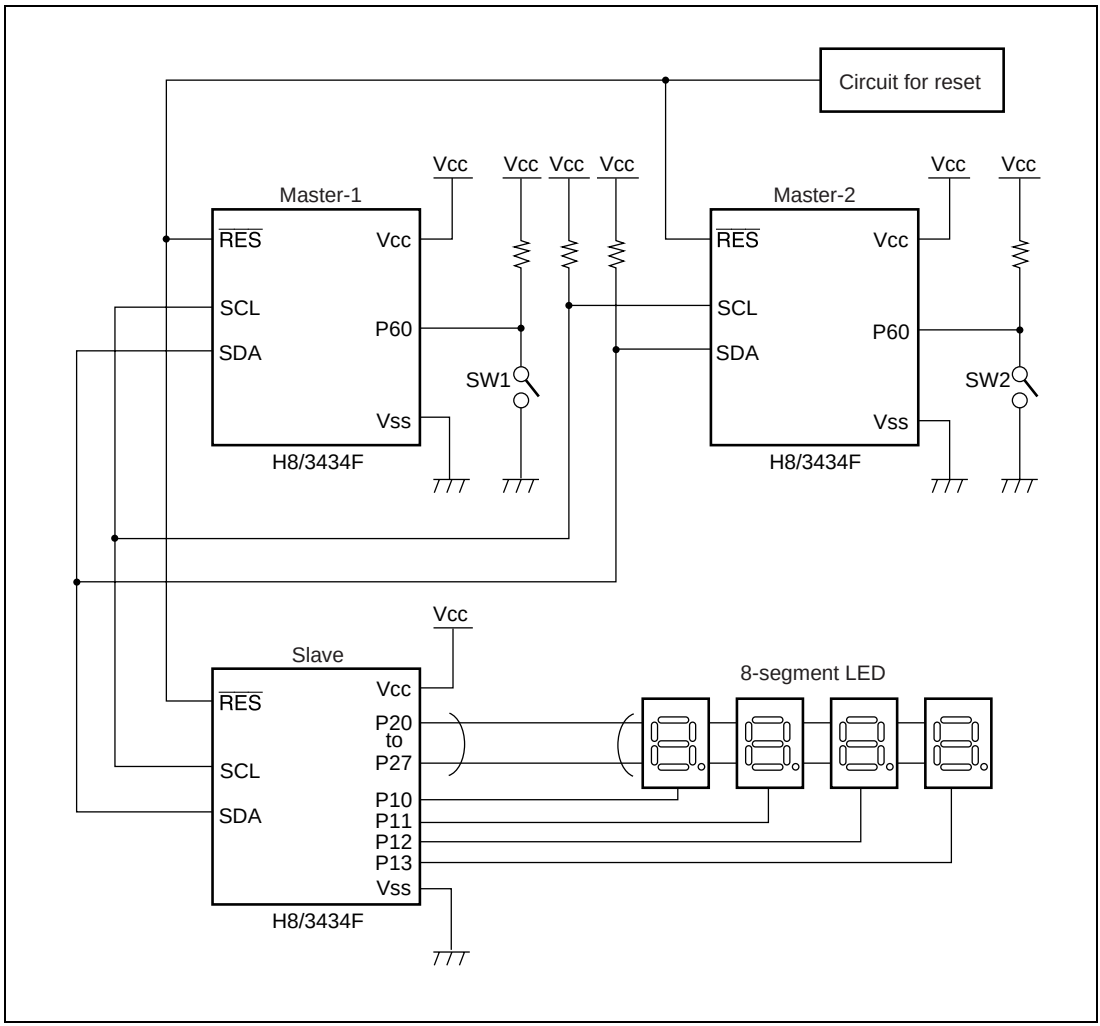


Figure 3.4 Circuit diagram of the system for performing a simple evaluation of the I²C bus

3.3 Design of Software

3.3.1 Description of Modules

This section presents an example of the software of the system that has the multi-master configuration. The divided program modules and their functions are listed in table 3.1.

Table 3.1 Description of modules

Module name	Label name	Functions
Master main program	Main	(1) Initial setting (stack pointer, I^2C bus interface, and 8-bit timer) (2) Enables interruption (3) Watches the switch and calls the master subroutine
Key scanning program (interruption program)	Compare	Reads the bit of I/O port 6 every 8 ms using the compare-match interruption for the 8-bit timer.
Data transmitting program	Master	Watches the bus and transmits the data
Slave main program	_Main	(1) Initial setting (stack pointer, I^2C bus interface, and 8-bit timer) (2) Enables interruption (3) Calls the slave subroutine
Program to display data on the 8-segment LED	_Display	Displays data on the 8-segment LED.
Data receiving program (interruption program)	_Receive	Receives data to make a decision.

3.3.2 Master

(1) Description of internal registers used by the master

Table 3.2 Description of internal registers used by the master

Registers	Functions	Names of modules using the registers
STCR	Selects the input clock for the 8-bit timer.	Data transmitting program
ICCR	Enables the I^2C bus interface. Sets for interruptions. Selects the communication mode. Selects the acknowledgment mode. Selects the frequency of the input clock.	

Registers	Functions	Names of modules using the registers
ICSR	Issues starting/stopping conditions. Recognizes and controls the acknowledgment.	
ICDR	Stores the transmission/reception data.	
ICMR	Selects MSB-first or LSB-first.	
SAR	Stores the slave address and selects the format.	
TCR	Selects the clock input. Selects the condition for clearing the counter. Enables compare-match interruption A.	Main program
TCSR	Clears the flag for the compare-match.	
TCORA	Sets the time for the compare-match.	
P6DR	Switches input port.	
P6DDR	Sets the port mode.	

(2) Description of the general-purpose registers used by the master

Table 3.3 Description of the general-purpose registers used by the master

Registers	Functions	Names of module using the registers
R1L,R2L	Working registers	Main program
R3L	Stores the transmission data temporarily.	Data transmitting program
R5L	Counts the bytes of transmitted data.	Data transmitting program
CCR	Checks the interruption flags.	Main program

(3) Description of the RAM used by the master

Table 3.4 Description of the RAM used by the master

Registers	Functions	Data length	Names of modules using the registers
Switch	Counts the jitter.	1 byte	Key scanning program (interruption program)

(4) Description of the ROM used by the master

Table 3.5 Description of the ROM used by the master

Label names	Functions	Data length	Names of modules using the registers
Table	Stores the transmission data.	2 bytes	Data transmitting program

3.3.3 Slave

(1) Description of internal registers used by the slave

Table 3.6 Description of internal registers used by the slave

Registers	Functions	Names of module using the registers
STCR	Selects the input clock for the 8-bit timer.	Main program
ICCR	Enables the I^2C bus interface. Sets for interruptions. Selects the communication mode. Selects the acknowledgment mode. Selects the frequency of the input clock.	
ICSR	Watches the data transmission/reception and checks whether or not an interruption occurred.	Data receiving program (interruption program)
ICDR	Stores the transmission/reception data.	
ICMR	Selects MSB-first or LSB-first.	
SAR	Stores the slave address and selects the format.	
TCR	Selects the clock input. Selects the clearing condition for the counter (clears the counter by compare-match interruption A).	Main program
TCSR	Checks the state of the flag for the compare-match.	
TCORA	Sets the time for compare-match A.	
TCORB	Sets the time for compare-match B.	
P1DDR	Sets the mode for port 1.	
P1DR	Digit data of the 8-segment LED	Program for displaying data on the 8-segment LED
P2DDR	Sets the mode for port 2.	Main program
P2DR	Segment data of the 8-segment LED	Program for displaying data on the 8-segment LED

(2) Description of the general-purpose registers used by the slave

Table 3.7 Description of the general-purpose registers used by the slave

Registers	Functions	Names of module using the registers
R1L	Working register	Main program
R1L	Working register	Program for displaying data on the 8-segment LED
R6	Temporary area for exchanging the data	
R1L	Working register	Data receiving program (interruption program)
R4	Sets the data table.	
CCR	Checks the interruption flags.	Main program

(3) Description of the RAM used by the slave

Table 3.8 Description of the RAM used by the slave

Label names	Functions	Data length	Names of module using the registers
_TABLE	Stores the starting address of the data table.	1 word	Data receiving program (interruption program)
_Count	Manages the display time for the LED.	1 byte	Program for displaying data on the 8-segment LED
_Count2	Manages the display time for the LED.	1 byte	
_D_DATA	Initial value of the digit data	1 byte	
_First	Stores the first byte of the reception data.	1 byte	Data receiving program (interruption program)
_Second	Stores the second byte of the reception data.	1 byte	

(4) Description of the ROM used by the slave

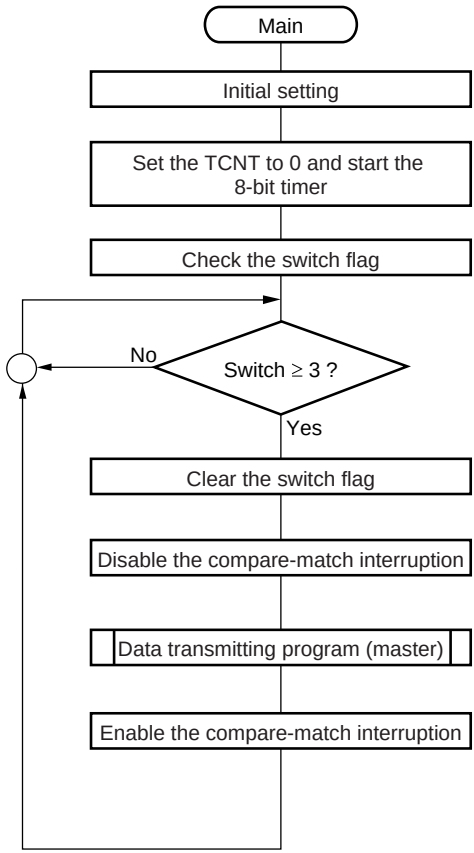
Table 3.9 Description of the ROM used by the slave

Label names	Functions	Data length	Names of module using the registers
_Table1	Stores the data for 8-segment.	1 byte	Data receiving program (interruption program)
_Table2	Stores the data for 8-segment.	1 byte	
_Table3	Stores the data for 8-segment.	1 byte	
_Table4	Stores the data for 8-segment.	1 byte	

3.4 Flowcharts

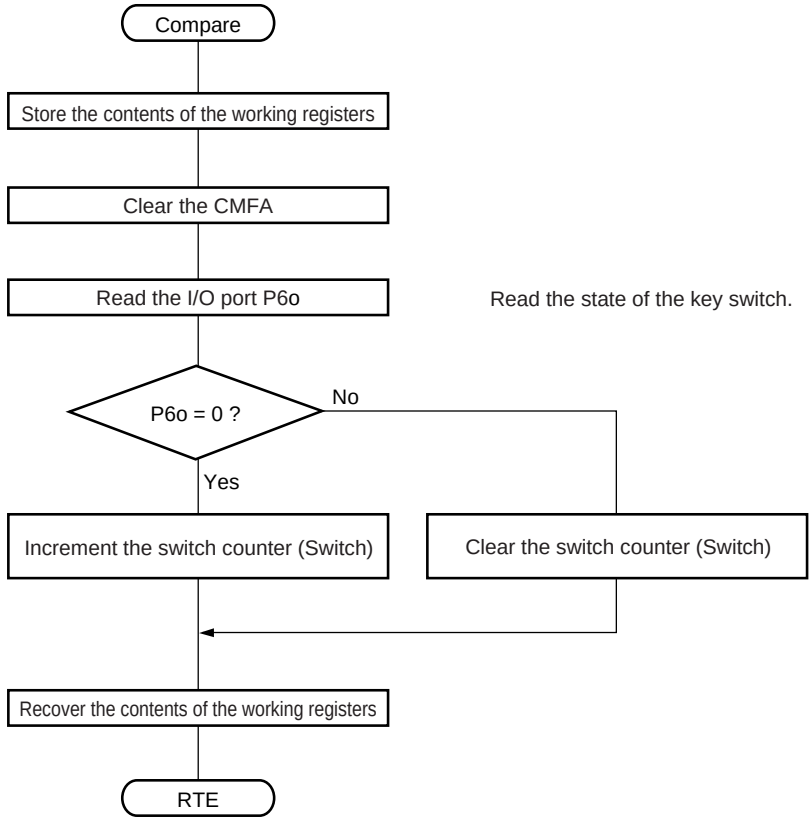
3.4.1 Master Program

(1) Main Program

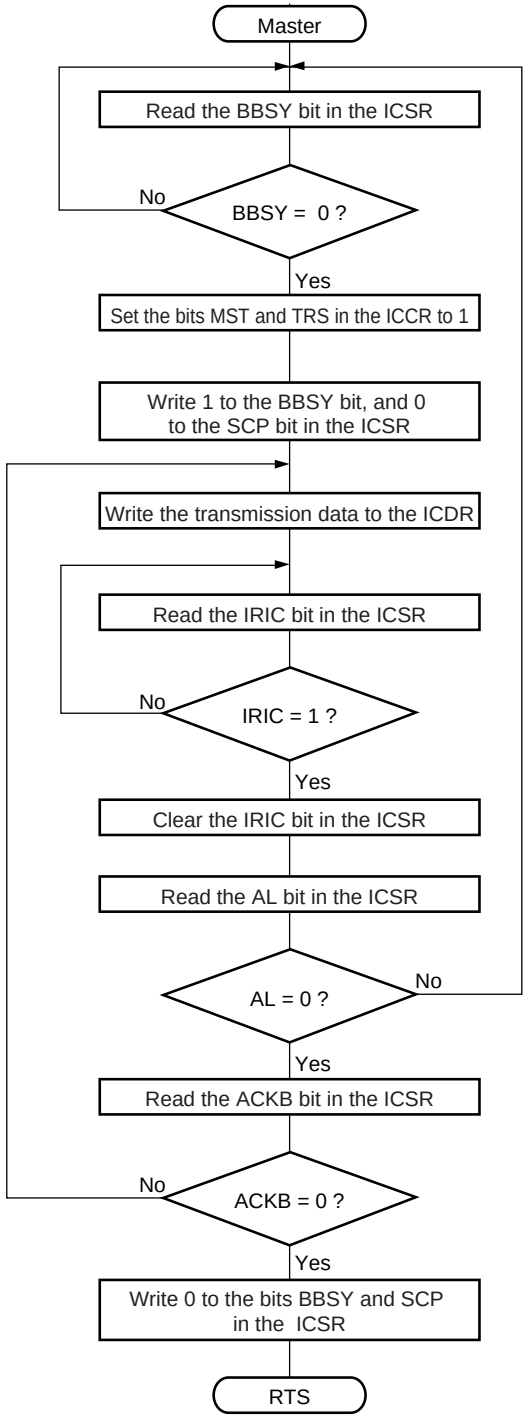


Initial setting
- Set the interruption vector
- Reserve the variable region
- Set the stack pointer
- Initialize the I^2C bus interface
- Initialize the 8-bit timer
- Initialize the I/O ports
- Enable the interruptions

(2) Key scanning program (interruption program)

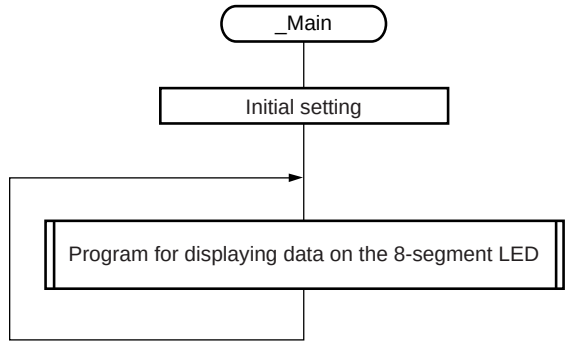


(3) Data transmitting program



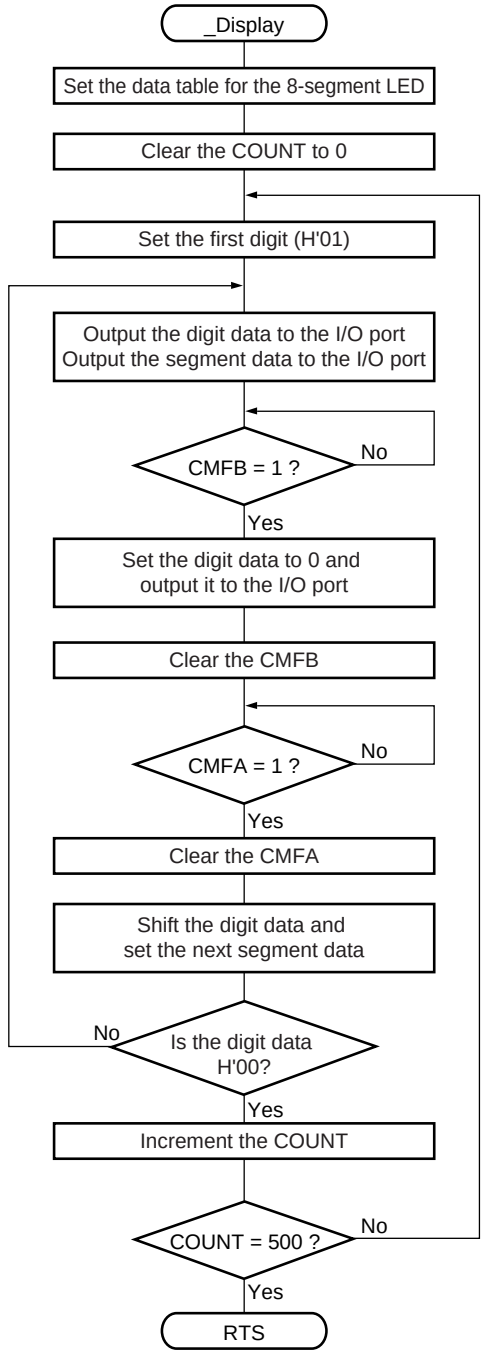
3.4.2 Slave Program

(1) Main Program

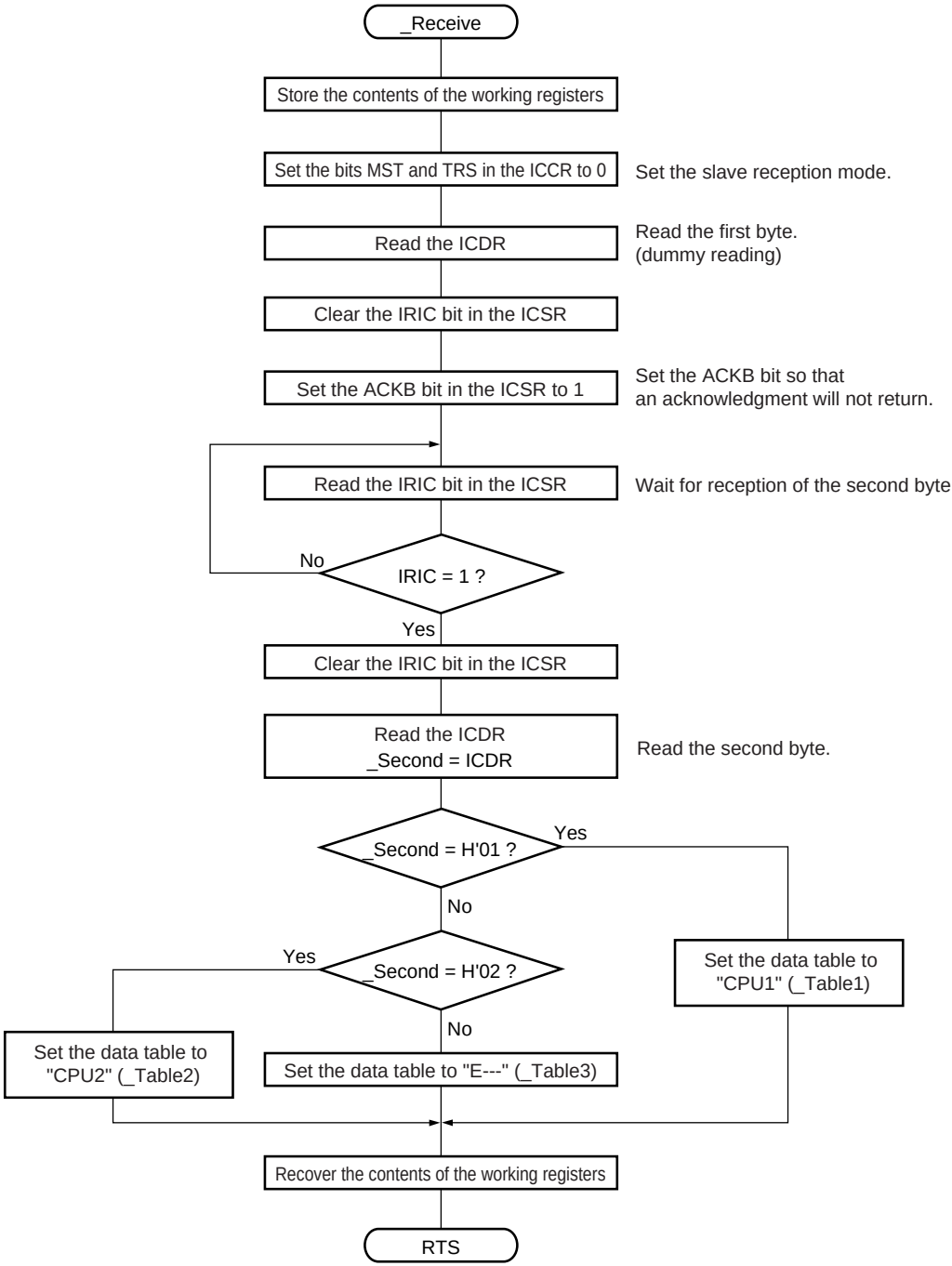


Initial setting
- Set the interruption vector
- Reserve the variable region
- Set the stack pointer
- Initialize the I^2C bus interface
- Initialize the 8-bit timer
- Initialize the I/O ports
- Enable the interruptions
- Set the data table (_Table) for the 8-segment LED to 000

(2) Program for displaying data on the 8-segment LED



(3) Data receiving program



3.5 Program Listings

3.5.1 Master Program

```
          .cpu 300
          .output  dbg
;*************************************************************************************************
; Master program of the evaluation system for the I²C bus
;         Key scanning
;         Data transmission
;*************************************************************************************************

;*****************************************************************
;         Vector addresses
;*****************************************************************
          .section      VECT,CODE,LOCATE=H'0000

Res       .DATA.W       Main

          .ORG          H'0006

NMI       .DATA.W       Main
IRQ0      .DATA.W       Main
IRQ1      .DATA.W       Main
IRQ2      .DATA.W       Main
IRQ3      .DATA.W       Main
IRQ4      .DATA.W       Main
IRQ5      .DATA.W       Main
IRQ6      .DATA.W       Main
IRQ7      .DATA.W       Main
ICIA      .DATA.W       Main
ICIB      .DATA.W       Main
ICIC      .DATA.W       Main
ICID      .DATA.W       Main
OCIA      .DATA.W       Main
OCIB      .DATA.W       Main
FOVI      .DATA.W       Main
```

```
CMI0A     .DATA.W       Compare
CMI0B     .DATA.W       Main
OVI0      .DATA.W       Main
CMI1A     .DATA.W       Main
CMI1B     .DATA.W       Main
OVI1      .DATA.W       Main
MREI      .DATA.W       Main
MWEI      .DATA.W       Main
ERI       .DATA.W       Main
RXI       .DATA.W       Main
TXI       .DATA.W       Main
RDI       .DATA.W       Main
;
;*********************************************************************
;        Definitions of the various interfaces
;*********************************************************************
;-------------------------------------------------
; Definition of the I2C bus registers
;-------------------------------------------------
_STCR     .EQU          H'FFC3              ; Serial timer control register
_ICCR     .EQU          H'FFD8              ; I2C bus control register
_ICSR     .EQU          H'FFD9              ; I2C bus state register
_ICDR     .EQU          H'FFDE              ; I2C bus data register
_ICMR     .EQU          H'FFDF              ; I2C bus mode register
_SAR      .EQU          H'FFDF              ; Slave-address register
;-------------------------------------------------
; Definition of the I/O registers
;-------------------------------------------------
_KMPCR    .EQU          H'FFF2              ; Port 6 input pull-up MOS control
                                            ; register
_P6DDR    .EQU          H'FFB9              ; Data-direction register
_P6DR     .EQU          H'FFBB              ; Data register (connects the switch)


;-------------------------------------------------
; Definition of the 8-bit timer register
;-------------------------------------------------
```

```
_TCR     .EQU          H'FFC8
_TCSR    .EQU          H'FFC9
_TCORA   .EQU          H'FFCA
_TCORB   .EQU          H'FFCB              ; Unused
_TCNT    .EQU          H'FFCC

;-----------------------------------------------
; Definition of the variables in RAM variables
;-----------------------------------------------
         .section  RAM,DATA,LOCATE=H'FB80

_Switch  .RES          1                   ; Variable to designate the switch's state
;***************************************************************
;        Start of the main program
;***************************************************************
         .section  program,data,locate=H'1000

Main     MOV.W         #H'FEFE,SP          ; Set the stack pointer.
;-----------------------------------------------
; Initialization of the I2C bus registers
;-----------------------------------------------

         MOV.B         #H'10,R1L
         MOV.B         R1L,@_STCR          ; IICE = 1
         MOV.B         #H'B4,R1L
         MOV.B         R1L,@_ICCR          ; ICE = 1,MST = 1,TRS = 1
                                           ; Set the transfer clock to 200 bps.
;-----------------------------------------------
; Initialization of the I/O registers
;-----------------------------------------------
         MOV.B         #H'00,R1L
         MOV.B         R1L,@_P6DDR

         MOV.B         #H'00,R1L
         MOV.B         R1L,@_KMPCR
```

```
;-------------------------------------------------
; Initialization of the 8-bit timer register
;-------------------------------------------------

          MOV.B         #H'4B,R1L
          MOV.B         R1L,@_TCR
          MOV.B         #H'7D,R1L
          MOV.B         R1L,@_TCORAe
;-------------------------------------------------
; Initialization of the switch counter
;-------------------------------------------------

          MOV.B         #H'00,R1L
          MOV.B         R1L,@_Switch         ; Initialize the switch counter to 0.
          MOV.B         #H'00,R1L            ; Reset the internal 8-bit counter to 0.

          MOV.B         R1L,@_TCNT           ; Start counting.

          ANDC          #H'7F,CCR            ; Clear the interrupt flag.

;-------------------------------------------------
; Judgement of the switch's state, ON or OFF
;-------------------------------------------------
          MOV.B         #H'03,R2L
SwOn      MOV.B         @_Switch,R1L
          CMP.B         R2L,R1L
          BLT           SwOn
          MOV.B         #H'00,R1L
          MOV.B         R1L,@_Switch         ; Clear the switch counter.
;-------------------------------------------------
; Data transmission
;-------------------------------------------------
          MOV.B         #H'0B,R1L
          MOV.B         R1L,@_TCR            ; Disable the CMPA interrupt.

          JSR           @Master              ; Jump to the data-transmission program.
```

```
        MOV.B       #H'4B,R1L
        MOV.B       R1L,@_TCR           ; Enable the CMFA interrupt.

        BRA         SwOn

;-------------------------------------------------
; Key-scanning routine (interrupt routine)
;-------------------------------------------------
Compare .EQU        $
        PUSH        R1
        BCLR        #6,@_TCSR           ; Clear the CMFA bit.
        BTST        #0,@_P6DR           ; Check the switch flag.
        BNE         Off                 ; Clear the switch counter.

        MOV.B       @_Switch,R1L        ; When the switch is off

        INC         R1L                 ; Increment the switch counter.
        MOV.B       R1L,@_Switch
        BRA         Clear

Off     MOV.B       #H'00,R1L           ; When the switch is off
        MOV.B       R1L,@_Switch        ; Clear the switch counter.

Clear   POP         R1

        RTE                             ; Return from the key-scanning routine.

        .include    "master.asm"        ; Combine the files.

;-------------------------------------------------
;       Set the initial value to the ROM
;-------------------------------------------------
_Table  .DATA.B     H'EE                ; The slave address (=H'77) and the R/W bit
                                        ; (=H'0) ——> B'11101110
        .DATA.B     H'01                ; The data to distinguish this master (master
                                        ; 2 is H'02)
```

```
        .END

;*****************************************************************************************
; Data-transmission program for the master
;       The first byte is the slave's address.
;       The second byte is the data that distinguishes this master.
;*****************************************************************************************
Master  BTST    #7,@_ICSR       ; Is the I2C bus free?
        BNE     Master

        BSET    #5,@_ICCR       ; Set the master-transmission mode.
        BSET    #4,@_ICCR       ;(MST = 0,TRS = 0)

        MOV.B   #H'90,R1L       ; Issue the start condition for
                                ; transmission.
        MOV.B   R1L,@_ICSR      ; ICSR : 1001 0000
        MOV.B   #H'00,R5L

Transmit MOV.B  @(_Table,R5),R3L ; Write the first byte (the slave address)
                                ; and the second byte (the data that
                                ; distinguishes the master).
        MOV.B   R3L,@_ICDR
        INC     R5L

ChkIRIC1 BTST   #6,@_ICSR
        BEQ     ChkIRIC1        ; IRIC = 1? (transmission completed?)
        BCLR    #6,@_ICSR       ; Clear the IRIC bit for the subsequent
                                ; transmission.

        BTST    #3,@_ICSR       ; AL = 0?
        BNE     Master

        BTST    #0,@_ICSR       ; ACKB = 0?
        BEQ     Transmit
```

```
MOV.B    #H'10,R1L              ; Issue the stop condition for transmission
MOV.B    R1L,@_ICS              ; ICSR : 0001 0000

RTS                             ; Return subroutine
```

3.5.2 Slave Program

```
          .cpu 300
          .output  dbg
;*************************************************************************************************
; Slave program of the evaluation system for the I2C bus
;         (1) LED display
;         (2) Data reception
;*************************************************************************************************

;*****************************************************************
; Definition of the on-chip registers
;*****************************************************************
_STCR     .EQU          H'FFC3               ; Serial timer control register
_ICCR     .EQU          H'FFD8               ; I2C bus control register
_ICSR     .EQU          H'FFD9               ; I2C bus state register
_ICDR     .EQU          H'FFDE               ; I2C bus data register
_ICMR     .EQU          H'FFDF               ; I2C mode register
_SAR      .EQU          H'FFDF               ; Slave-address register
_TCR      .EQU          H'FFC8               ; Timer control register
_TCSR     .EQU          H'FFC9               ; Timer control/state register
_TCORA    .EQU          H'FFCA               ; Time constant register
_TCORB    .EQU          H'FFCB               ; Time constant register
_P1DDR    .EQU          H'FFB0               ; Port 1 data-direction register
_P2DDR    .EQU          H'FFB1               ; Port 2 data-direction register
_P1DR     .EQU          H'FFB2               ; Port 1 data register
_P2DR     .EQU          H'FFB3               ; Port 2 data register

          .section      VECT,CODE,LOCATE=H'0000
;*************************************************************************************************
; Vector Address
;*************************************************************************************************
Res       .DATA.W       _Main
          .ORG          H'0006
NMI       .DATA.W       _Main
IRQ0      .DATA.W       _Main
IRQ1      .DATA.W       _Main
IRQ2      .DATA.W       _Main
```

```
IRQ3     .DATA.W       _Main
IRQ4     .DATA.W       _Main
IRQ5     .DATA.W       _Main
IRQ6     .DATA.W       _Main
IRQ7     .DATA.W       _Main
ICIA     .DATA.W       _Main
ICIB     .DATA.W       _Main
ICIC     .DATA.W       _Main
ICID     .DATA.W       _Main
OCIA     .DATA.W       _Main
OCIB     .DATA.W       _Main
FOVI     .DATA.W       _Main
CMI0A    .DATA.W       _Main
CMI0B    .DATA.W       _Main
OVI0     .DATA.W       _Main
CMI1A    .DATA.W       _Main
CMI1B    .DATA.W       _Main
OVI1     .DATA.W       _Main
IBF1     .DATA.W       _Main
IBF2     .DATA.W       _Main
ERI0     .DATA.W       _Main
RXI0     .DATA.W       _Main
TXI0     .DATA.W       _Main
TEI0     .DATA.W       _Main
ERI1     .DATA.W       _Main
RXI1     .DATA.W       _Main
TXI1     .DATA.W       _Main
TEI1     .DATA.W       _Main
ADI      .DATA.W       _Main
WOVF     .DATA.W       _Main
IICI     .DATA.W       _Receive

         .SECTION      RAM,DATA,LOCATE=H'FB80
;-------------------------------------------------
; Initialization of the RAM area
;-------------------------------------------------
_TABLE   .RES.W        1                  ; H'FB80<- The place to store the received
```

```
                                        ; data
_Count    .RES.B        1               ; H'FB82<- The time period for illuminating
                                        ; the LED
_Count2   .RES.B        1               ; H'FB83<- The time period for illuminating
                                        ; the LED
_D_DATA   .RES.B        1               ; H'FB84<- Keep the digit data here.
_First    .RES.B        1               ; H'FB85<- Data for transmission 1
_Second   .RES.B        1               ; H'FB86<- Data for transmission 2

          .SECTION      PROGRAM,CODE,LOCATE=H'1000
;*********************************************************************
; Start of the main program
;*********************************************************************
_Main     MOV.W         #H'FEFE,SP      ; Set the stack pointer.
                                        ; Settings for the program to use in

;---------------------------------------------------------------------
; Settings for the program to use in displaying data on the LED
;---------------------------------------------------------------------
          MOV.B         #H'0A,R1L       ; Set the condition for clearing the counter.
          MOV.B         R1L,@_TCR

          MOV.B         #H'F0,R1L       ; Compare-match B
          MOV.B         R1L,@_TCORB

          MOV.B         #H'FF,R1L       ; Compare-match A
          MOV.B         R1L,@_TCORA

          MOV.B         #H'FF,R1L
          MOV.B         R1L,@_P1DDR     ; All pins are outputs.
          MOV.B         R1L,@_P2DDR     ; All pins are outputs.

;---------------------------------------------------------------------
; Initialization of the I2C bus interface registers
;---------------------------------------------------------------------
          MOV.B         #H'11,R1L       ; IICE = 1
          MOV.B         R1L,@_STCR      ; 0001 0001
```

```
         MOV.B         #H'EE,R1L
         MOV.B         R1L,@_SAR            ; Set the slave address.
         MOV.B         #H'C4,R1L            ; ICE = 1,IEIC = 1, Transfer clock : 400 MHz
         MOV.B         R1L,@_ICCR           ; B'1100 0100

;---------------------------------------------------------------------
; Cancellation of the interruption mask
;---------------------------------------------------------------------
     ANDC          #H'7F,CCR
;---------------------------------------------------------------------
; Swapping the data tables
;---------------------------------------------------------------------
         MOV.W         #_Table4,R0
         MOV.W         R0,@_TABLE

LOOP     JSR           @_Display            ; Jump to the routine for displaying data on
                                            ; the LED.
         BRA           LOOP

;*********************************************************************
; Subroutine for displaying data on the LED
;*********************************************************************
_Display  MOV.W        @_TABLE,R6           ; Exchanging the data tables.
         MOV.B         R1L,@_Count2         ; Count2 = 0
MORE2    MOV.B         #H'00,R1L
         MOV.B         R1L,@_Count          ; Count = 0

MORE1    MOV.W         @_TABLE,R6           ; Set the starting address of the data table.
         MOV.B         #H'08,R1L
         MOV.B         R1L,@_D_DATA         ; Set the digit data, H'01.

NEXT1    MOV.B         @_D_DATA,R1L
         NOT  R1L
         MOV.B         R1L,@_P1DR           ; Output the digit data.
         MOV.B         @R6,R1L
         MOV.B         R1L,@_P2DR           ; Output the segment data.
```

```
CMFB1   BTST    #7,@_TCSR       ; CMFB = 1?
        BEQ     CMFB1
        BCLR    #7,@_TCSR
        MOV.B   #H'FF,R1L
        MOV.B   R1L,@_P1DR      ; Output the digit data, H'FF.
CMFA1   BTST    #6,@_TCSR       ; CMFA = 1?
        BEQ     CMFA1
        BCLR    #6,@_TCSR

        MOV.B   @_D_DATA,R1L     ; Shift the digit data.
        SHLR    R1L
        MOV.B   R1L,@_D_DATA

        ADDS    #1,R6           ; Prepare the next data for the LED.
        CMP.B   #H'00,R1L
        BNE     NEXT1

        MOV.B   @_Count,R1L
        INC     R1L
        MOV.B   R1L,@_Count

        MOV.B   @_Count,R1L
        CMP.B   #H'FF,R1L
        BNE     MORE1

        MOV.B   @_Count2,R1L
        INC     R1L
        MOV.B   R1L,@_Count2
        MOV.B   @_Count2,R1L
        CMP.B   #H'02,R1L
        BNE     MORE2
        RTS
```

```
;*********************************************************************************************
; The interrupt handler for the I²C bus interface
;         Data reception and judgement
;         Exchanging the data tables
;*********************************************************************************************
_Receive   PUSH      R1
           PUSH          R4                  ; Store the contents of the registers.

           BCLR          #6,@_ICSR           ; Clear the IRIC.

           MOV.B         @_ICDR,R1L          ; Read the data (a dummy read).
           MOV.B         R1L,@_First         ; Store the data in memory.
           BSET          #0,@_ICSR           ; ACKB = 1
LOOP1      BTST          #6,@_ICSR           ; Has reception of the second byte (the data
                                             ; to distinguish the master) finished?
           BEQ           LOOP1               ;

           BCLR          #6,@_ICSR           ; Clear the IRIC.

           MOV.B         @_ICDR,R1L
           MOV.B         R1L,@_Second        ; Store the received data in memory.

           BCLR          #0,@_ICSR           ; ACKB = 0
                                             ; Set the conditions for the subsequent
                                             ; reception of data.

           MOV.B         #H'00,R1L
           MOV.B         R1L,@_ICMR          ; Set the condition that specifies 9 bits per
                                             ; 1 frame.

;---------------------------------------------------------------------
; Judgement
;---------------------------------------------------------------------
           MOV.B         @_Second,R1L        ; Read the data that distinguishes the master.

_Judge     CMP.B         #H'01,R1L           ; Judgement of the reception data
           BEQ           EXIT1
```

```
        CMP.B       #H'02,R1L
        BEQ         EXIT2
EXIT3   MOV.W       #_Table3,R4
        MOV.W       R4,@_TABLE
        BRA         Clear

EXIT1   MOV.W       #_Table1,R4
        MOV.W       R4,@_TABLE
        BRA         Clear

EXIT2   MOV.W       #_Table2,R4
        MOV.W       R4,@_TABLE
        BRA         Clear

Clear   POP         R1
        POP         R2
        POP         R4                  ; Recover the contents of the registers.

        RTE
;---------------------------------------------------------------------
; The data table for the 8-segment LED
;---------------------------------------------------------------------
_Table1 .DATA.B     H'9C                ;H'004B  LED DATA of "C"
        .DATA.B     H'CE                ;H'004C  LED DATA of "P"
        .DATA.B     H'7C                ;H'004D  LED DATA of "U"
        .DATA.B     H'60                ;H'004E  LED DATA of "1"
_Table2 .DATA.B     H'9C                ;H'004F  LED DATA of "C"
        .DATA.B     H'CE                ;H'0050  LED DATA of "P"
        .DATA.B     H'7C                ;H'0051  LED DATA of "U"
        .DATA.B     H'DA                ;H'0052  LED DATA of "2"
_Table3 .DATA.B     H'9F                ;H'0053  LED DATA of "E"
        .DATA.B     H'02                ;H'0054  LED DATA of "-"
        .DATA.B     H'02                ;H'0055  LED DATA of "-"
        .DATA.B     H'02                ;H'0056  LED DATA of "-"
_Table4 .DATA.B     H'FC                ;H'0053  LED DATA of "0"
        .DATA.B     H'FC                ;H'0054  LED DATA of "0"
        .DATA.B     H'FC                ;H'0055  LED DATA of "0"
```

```
.DATA.B       H'FC                ;H'0056  LED DATA of "0"

.END
```

Section 4 Example Applications for the H8S Series

4.1 Usage Guide to the Example Applications for the H8S Series

4.1.1 The Structure of the Example Applications for the H8S Series

The chapter, 'Example Applications for the H8S series', has the structure shown in the figure 4.1. The example applications for the H8S series product's I^2C bus interface are described in this chapter.

The H8S/2138 is used as the device.

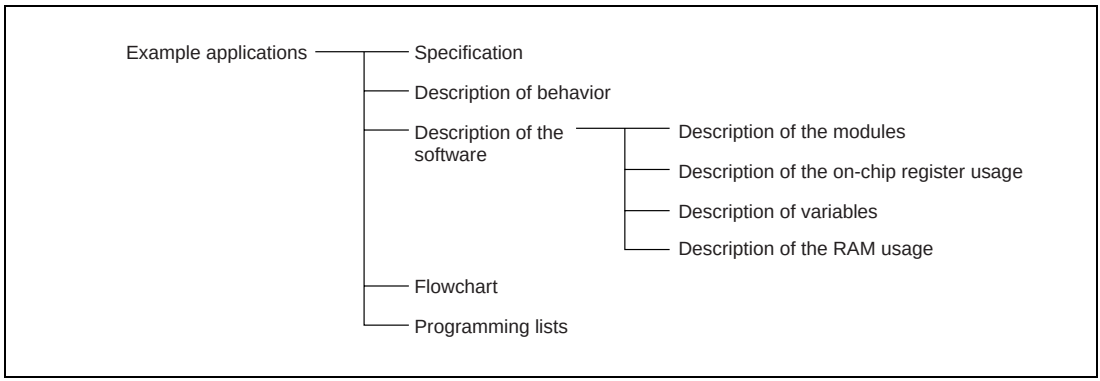


Figure 4.1 The structure of the example applications for the H8S Series

(1) Specification

Describes the system specification for these example tasks.

(2) Description of behavior

Uses timing charts to describe the behavior of these example tasks.

(3) Description of the software

(1) Description of the modules
Describes the modules of the software of this example task.

(2) Description of the on-chip register usage
Describes the settings of the I^2C bus interface in the modules and of the on-chip registers

(3) Description of the variables
Describes the variables of the software that are used in the task examples.

(4) Description of the RAM usage
Describes the label names and functions of RAM locations that are used by the modules.

(4) Flowchart

Uses flowcharts to describe the software that carries out the task examples.

(5) Program listings

Gives the program listings of the software that carries out the task examples.

4.1.2 Description of the Definition File for the Vector Table

The definition file for the vector table, in the C language, is described below. The file that defines the starting addresses of the interrupt handling routines is shown in figure 4.2. To use an interrupt handling routine, a label that gives the starting address of that routine should be written to the corresponding position in the vector table. Figure 4.2 gives an example that uses the IIC's channel-0 interrupt. The starting address (IIC0INT) is referred to by 'external reference' (refer to figure 4.2-A). The label that shows the position of the IICI0 handler should be named IIC0INT (refer to figure 4.2-B).

The label name 'IIC0INT' is referred to by 'external reference'.

```
/*******************************************************
*    H8S/2138 Series vector table
*        for mode3(normal,single-chip mode)
*******************************************************/

extern void main(void);
extern void IIC0INT(void);
const void (*vect_tbl[])(void) =
{
    main,                                   /* H'0000  Reset                */
    main,                                   /* H'0002  Reserve              */
    main,                                   /* H'0004  Reserve              */
    main,                                   /* H'0006  Reserve              */
    main,                                   /* H'0008  Reserve              */
    main,                                   /* H'000A  Reserve              */
    main,                                   /* H'000C  Direct transfer      */
    main,                                   /* H'000E  NMI                  */
    main,                                   /* H'0010  Trap                 */
    main,                                   /* H'0012  Trap                 */
    main,                                   /* H'0014  Trap                 */
```

A

The label name 'IIC0INT' is referred to by 'external reference'.

Figure 4.2 Definition file for the vector table

```
    main,                                   /* H'0016  Trap                  */
    main,                                   /* H'0018  Reserve               */
    main,                                   /* H'001A  Reserve               */
    main,                                   /* H'001C  Reserve               */
    main,                                   /* H'001E  Reserve               */
    main,                                   /* H'0020  IRQ0                  */
    main,                                   /* H'0022  IRQ1                  */
    main,                                   /* H'0024  IRQ2                  */
    main,                                   /* H'0026  IRQ3                  */
    main,                                   /* H'0028  IRQ4                  */
    main,                                   /* H'002A IRQ5                   */
    main,                                   /* H'002C IRQ6,KIN7-KIN0         */
    main,                                   /* H'002E  IRQ7                  */
    main,                                   /* H'0030  SWDTEND               */
    main,                                   /* H'0032  WOVI0                 */
    main,                                   /* H'0034  WOVI1                 */
    main,                                   /* H'0036  PC break              */
    main,                                   /* H'0038  ADI                   */
    main,                                   /* H'003A  Reserve               */
    main,                                   /* H'003C  Reserve               */
    main,                                   /* H'003E  Reserve               */
    main,                                   /* H'0040  Reserve               */
    main,                                   /* H'0042  Reserve               */
    main,                                   /* H'0044  Reserve               */
    main,                                   /* H'0046  Reserve               */
    main,                                   /* H'0048  Reserve               */
    main,                                   /* H'004A  Reserve               */
    main,                                   /* H'004C  Reserve               */
    main,                                   /* H'004E  Reserve               */
    main,                                   /* H'0050  Reserve               */
    main,                                   /* H'0052  Reserve               */
    main,                                   /* H'0054  Reserve               */
    main,                                   /* H'0056  Reserve               */
    main,                                   /* H'0058  Reserve               */
    main,                                   /* H'005A  Reserve               */
    main,                                   /* H'005C  Reserve               */
```

Figure 4.2 Definition file for the vector table (cont)

```
    main,                                   /* H'005E  Reserve                */
    main,                                   /* H'0060  ICIA                   */
    main,                                   /* H'0062  ICIB                   */
    main,                                   /* H'0064  ICIC                   */
    main,                                   /* H'0066  ICID                   */
    main,                                   /* H'0068  OCIA                   */
    main,                                   /* H'006A  OCIB                   */
    main,                                   /* H'006C  FOVI                   */
    main,                                   /* H'006E  Reserve                */
    main,                                   /* H'0070  Reserve                */
    main,                                   /* H'0072  Reserve                */
    main,                                   /* H'0074  Reserve                */
    main,                                   /* H'0076  Reserve                */
    main,                                   /* H'0078  Reserve                */
    main,                                   /* H'007A  Reserve                */
    main,                                   /* H'007C  Reserve                */
    main,                                   /* H'007E  Reserve                */
    main,                                   /* H'0080  CMIA0                  */
    main,                                   /* H'0082  CMIB0                  */
    main,                                   /* H'0084  OVI0                   */
    main,                                   /* H'0086  Reserve                */
    main,                                   /* H'0088  CMIA1                  */
    main,                                   /* H'008A  CMIB1                  */
    main,                                   /* H'008C  OVI1                   */
    main,                                   /* H'008E  Reserve                */
    main,                                   /* H'0090  CMIAY                  */
    main,                                   /* H'0092  CMIBY                  */
    main,                                   /* H'0094  OVIY                   */
    main,                                   /* H'0096  ICIX                   */
    main,                                   /* H'0098  IBF1                   */
    main,                                   /* H'009A  IBF2                   */
    main,                                   /* H'009C  Reserve                */
    main,                                   /* H'009E  Reserve                */
    main,                                   /* H'00A0  ERI0                   */
    main,                                   /* H'00A6  TEI0                   */
    main,                                   /* H'00A8  ERI1                   */
```

Figure 4.2 Definition file for the vector table (cont)

```
    main,                                   /* H'00AA  RXI1                 */
    main,                                   /* H'00AC  TXI1                 */
    main,                                   /* H'00AE  TEI1                 */
    main,                                   /* H'00B0  ERI2                 */
    main,                                   /* H'00B2  RXI2                 */
    main,                                   /* H'00B4  TXI2                 */
    IIC0INT,                                /* H'00B6  TEI2                 */
    main,                                   /* H'00B8  IICI0                */
    main,                                   /* H'00BA  DDCSWI               */
    main,                                   /* H'00BC  IICI1                */
    main,                                   /* H'00BE  Reserve              */
    main,                                   /* H'00C0  Reserve              */
    main,                                   /* H'00C2  Reserve              */
    main,                                   /* H'00C4  Reserve              */
    main,                                   /* H'00C6  Reserve              */
    main,                                   /* H'00C8  Reserve              */
    main,                                   /* H'00CA  Reserve              */
    main,                                   /* H'00CC  Reserve              */
    main,                                   /* H'00CE  Reserve              */
};
```

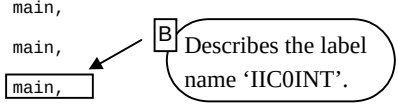


Figure 4.2 Definition file for the vector table (cont)

4.1.3 Description of the Definition File for the Registers

The definition file for the registers of H8S/2138 series products is given below.

The definition file for the registers of H8S/2138 Series products (1) <2138s.h>

```
/************************************************************************************/
/*      H8S/2138 Series Include File                                                */
/************************************************************************************/
union un_kbcomp {/* union KBCOMP  */
                unsigned char BYTE;                             /*              */
                struct {                                        /* Bit  Access  */
                       unsigned char IrE  :1;                   /*    IrE       */
                       unsigned char IrCKS:3;                   /*    IrCKS     */
                       unsigned char KBADE:1;                   /*    KBADE     */
                       unsigned char KBCH :3;                   /*    KBCH      */
                       }       BIT;                             /*              */
};                                                              /*              */
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       */
                           nsigned  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;                                 /*               */
char              wk[4];                                /*               */
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;                    /*              */
};                                                          /*              */
union un_ddcswr {                                           /* union DDCSWR */
                unsigned char BYTE;                         /*              */
                struct {                                    /*  Bit  Access */
                       unsigned char SWE:1;                 /*    SWE       */
                       unsigned char SW :1;                 /*    SW        */
                       unsigned char IE :1;                 /*    IE        */
                       unsigned char IF :1;                 /*    IF        */
                       }       BIT;                         /*              */
};                                                          /*              */
struct st_intc {                                            /* struct INTC  */
               union {                                      /* ICRA         */
                     unsigned char BYTE;                    /* Byte Access  */
                     struct {                               /* Bit  Access  */
                            unsigned char B7:1;             /*    IRQ0      */
                            unsigned char B6:1;             /*    IRQ1      */
                            unsigned char B5:1;             /*    IRQ2,IRQ3 */
                            unsigned char B4:1;             /*    IRQ4,IRQ5 */
                            unsigned char B3:1;             /*    IRQ6,IRQ7 */
                            unsigned char B2:1;             /*    DTC       */
                            unsigned char B1:1;             /*    WDT0      */
                            unsigned char B0:1;             /*    WDT1      */
                            }       BIT;                    /*              */
                     }         ICRA;                        /*              */
               union {                                      /* ICRB         */
                     unsigned char BYTE;                    /* Byte Access  */
                     struct {                               /* Bit  Access  */
                            unsigned char B7:1;             /*    A/D       */
                            unsigned char B6:1;             /*    FRT       */
                            unsigned char   :2;             /*              */
                            unsigned char B3:1;             /*    TMR0      */
                            unsigned char B2:1;             /*    TMR1      */
                            unsigned char B1:1;             /*    TMRX,Y    */
                            unsigned char B0:1;             /*    HIF       */
                            }       BIT;                    /*              */
                     }         ICRB;                        /*              */
```

```
union {                                 /* ICRC          */
      unsigned char BYTE;               /* Byte Access   */
      struct {                          /* Bit  Access   */
            unsigned char B7:1;         /*    SCI0       */
            unsigned char B6:1;         /*    SCI1       */
            unsigned char B5:1;         /*    SCI2       */
            unsigned char B4:1;         /*    IIC0       */
            unsigned char B3:1;         /*    IIC1       */
            }       BIT;                /*               */
      }             ICRC;               /*               */
union {                                 /* ISR           */
      unsigned char BYTE;               /* Byte Access   */
      struct {                          /* Bit  Access   */
            unsigned char IRQ7F:1;      /*    IRQ7F      */
            unsigned char IRQ6F:1;      /*    IRQ6F      */
            unsigned char IRQ5F:1;      /*    IRQ5F      */
            unsigned char IRQ4F:1;      /*    IRQ4F      */
            unsigned char IRQ3F:1;      /*    IRQ3F      */
            unsigned char IRQ2F:1;      /*    IRQ2F      */
            unsigned char IRQ1F:1;      /*    IRQ1F      */
            unsigned char IRQ0F:1;      /*    IRQ0F      */
            }       BIT;                /*               */
      }             ISR;                /*               */
union {                                 /* ISCR          */
      unsigned int WORD;                /*  Word Access  */
      struct {                          /*  Byte Access  */
            unsigned char H;            /*    ISCRH      */
            unsigned char L;            /*    ISCRL      */
            }       BYTE;               /*               */
      struct {                          /*  Bit  Access  */
            unsigned char IRQ7SC:2;     /*    IRQ7SC     */
            unsigned char IRQ6SC:2;     /*    IRQ6SC     */
            unsigned char IRQ5SC:2;     /*    IRQ5SC     */
            unsigned char IRQ4SC:2;     /*    IRQ4SC     */
            unsigned char IRQ3SC:2;     /*    IRQ3SC     */
            unsigned char IRQ2SC:2;     /*    IRQ2SC     */
            unsigned char IRQ1SC:2;     /*    IRQ1SC     */
```

```
                unsigned char IRQ0SC:2;             /*    IRQ0SC     */
                }       BIT;                        /*               */
        }               ISCR;                       /*               */
    char                wk1[6];                     /*               */
    union {                                         /* ABRKCR        */
        unsigned char BYTE;                         /*  Byte Access  */
        struct {                                    /*  Bit  Access  */
                unsigned char CMF:1;                /*    CMF        */
                unsigned char    :6;                /*               */
                unsigned char BIE:1;                /*    BIE        */
                }       BIT;                        /*               */
        }               ABRKCR;                     /*               */
    unsigned char       BARA;                       /* BARA          */
    unsigned char       BARB;                       /* BARB          */
    unsigned char       BARC;                       /* BARC          */
    char                wk2[202];                   /*               */
    union {                                         /* IER           */
        unsigned char BYTE;                         /*  Byte Access  */
        struct {                                    /*  Bit  Access  */
                unsigned char IRQ7E:1;              /*    IRQ7E      */
                unsigned char IRQ6E:1;              /*    IRQ6E      */
                unsigned char IRQ5E:1;              /*    IRQ5E      */
                unsigned char IRQ4E:1;              /*    IRQ4E      */
                unsigned char IRQ3E:1;              /*    IRQ3E      */
                unsigned char IRQ2E:1;              /*    IRQ2E      */
                unsigned char IRQ1E:1;              /*    IRQ1E      */
                unsigned char IRQ0E:1;              /*    IRQ0E      */
                }       BIT;                        /*               */
        }               IER;                        /*               */
    char                wk3[46];                    /*               */
    union {                                         /* KMIMR         */
        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;                        /*               */
                  }             KMIMR;                      /*               */
            char                wk4;                        /*               */
            union {                                         /* KMIMRA        */
                  unsigned char BYTE;                       /*  Byte Access  */
                  struct {                                  /*  Bit  Access  */
                        unsigned char B15:1;                /*    Bit 7      */
                        unsigned char B14:1;                /*    Bit 6      */
                        unsigned char B13:1;                /*    Bit 5      */
                        unsigned char B12:1;                /*    Bit 4      */
                        unsigned char B11:1;                /*    Bit 3      */
                        unsigned char B10:1;                /*    Bit 2      */
                        unsigned char B9 :1;                /*    Bit 1      */
                        unsigned char B8 :1;                /*    Bit 0      */
                        }       BIT;                        /*               */
                  }             KMIMRA;                     /*               */
};                                                          /*               */
struct st_dtc {                                             /* struct DTC    */
            union {                                         /* EA            */
                  unsigned char BYTE;                       /*  Byte Access  */
                  struct {                                  /*  Bit  Access  */
                        unsigned char B7:1;                 /*    IRQ0       */
                        unsigned char B6:1;                 /*    IRQ1       */
                        unsigned char B5:1;                 /*    IRQ2       */
                        unsigned char B4:1;                 /*    IRQ3       */
                        unsigned char B3:1;                 /*    A/D        */
                        unsigned char B2:1;                 /*    FRT ICIA   */
                        unsigned char B1:1;                 /*    FRT ICIB   */
                        unsigned char B0:1;                 /*    FRT OCIA   */
                        }       BIT;                        /*               */
                  }             EA;                         /*               */
            union {                                         /* EB            */
                  unsigned char BYTE;                       /*  Byte Access  */
```

```
        struct {                                /*  Bit  Access  */
               unsigned char B7:1;              /*    FRT OCIB   */
               unsigned char   :4;              /*               */
               unsigned char B2:1;              /*    TMR0 CMIA  */
               unsigned char B1:1;              /*    TMR0 CMIB  */
               unsigned char B0:1;              /*    TMR1 CMIA  */
               }       BIT;                     /*               */
        }              EB;                      /*               */
    union {                                     /* EC            */
        unsigned char BYTE;                     /*  Byte Access  */
        struct {                                /*  Bit  Access  */
               unsigned char B7:1;              /*    TMR1 CMIB  */
               unsigned char B6:1;              /*    TMRY CMIA  */
               unsigned char B5:1;              /*    TMRY CMIB  */
               unsigned char B4:1;              /*    HIF1       */
               unsigned char B3:1;              /*    HIF2       */
               unsigned char B2:1;              /*    SCIO RXI   */
               unsigned char B1:1;              /*    SCIO TXI   */
               unsigned char B0:1;              /*    SCI1 RXI   */
               }       BIT;                     /*               */
        }              EC;                      /*               */
    union {                                     /* ED            */
        unsigned char BYTE;                     /* Byte Access   */
        struct {                                /* Bit  Access   */
               unsigned char B7:1;              /*    SCI1 TXI   */
               unsigned char B6:1;              /*    SCI2 RXI   */
               unsigned char B5:1;              /*    SCI2 TXI   */
               unsigned char B4:1;              /*    IIC0       */
               unsigned char B3:1;              /*    IIC1       */
               }       BIT;                     /*               */
        }              ED;                      /*               */
    char               wk;                      /*               */
    union {                                     /* VECR          */
        unsigned char BYTE;                     /*  Byte Access  */
        struct {                                /*  Bit  Access  */
               unsigned char SWDTE:1;           /*    SWDTE      */
               unsigned char DTVEC:7;           /*    DTVEC      */
```

```
                        }       BIT;                    /*              */
                    }           VECR;                   /*              */
};                                                      /*              */
struct st_flash {                                       /* struct FLASH */
                union {                                 /* FLMCR1       */
                      unsigned char BYTE;               /*  Byte Access */
                      struct {                          /*  Bit  Access */
                            unsigned char FWE:1;        /*    FWE       */
                            unsigned char SWE:1;        /*    SWE       */
                            unsigned char    :2;        /*              */
                            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      */
                            unsigned char     :5;       /*              */
                            unsigned char ESU :1;       /*    ESU       */
                            unsigned char PSU :1;       /*    PSU       */
                            }       BIT;                /*              */
                      }             FLMCR2;             /*              */
                union {                                 /* EBR1         */
                      unsigned char BYTE;               /*  Byte Access */
                      struct {                          /*  Bit  Access */
                            unsigned char wk :6;        /*              */
                            unsigned char EB9:1;        /*    EB9       */
                            unsigned char EB8:1;        /*    EB8       */
                            }       BIT;                /*              */
                      }             EBR1;               /*              */
                union {                                 /* EBR2         */
                      unsigned char BYTE;               /*  Byte Access */
                      struct {                          /*  Bit  Access */
                            unsigned char EB7:1;        /*    EB7       */
```

```
                        unsigned char EB6:1;                /*    EB6        */
                        unsigned char EB5:1;                /*    EB5        */
                        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;                        /*               */
                    }              EBR2;                    /*               */
};                                                          /*               */
struct st_pwm {                                             /* struct PWM    */
              union {                                       /* PCSR          */
                    unsigned char BYTE;                     /*  Byte Access  */
                    struct {                                /*  Bit  Access  */
                          unsigned char wk   :5;            /*               */
                          unsigned char PWCKB:1;            /*    PWCKB      */
                          unsigned char PWCKA:1;            /*    PWCKA      */
                          }       BIT;                      /*               */
                    }              PCSR;                    /*               */
              char                  wk[79];                 /*               */
              union {                                       /* PWOER */
                    unsigned int WORD;                      /*  Word Access  */
                    struct {                                /*  Byte Access  */
                          unsigned char B;                  /*    PWOERB     */
                          unsigned char A;                  /*    PWOERA     */
                          }       BYTE;                     /*               */
                    struct {                                /*  Bit  Access  */
                          unsigned char OE15:1;             /*    OE15       */
                          unsigned char OE14:1;             /*    OE14       */
                          unsigned char OE13:1;             /*    OE13       */
                          unsigned char OE12:1;             /*    OE12       */
                          unsigned char OE11:1;             /*    OE11       */
                          unsigned char OE10:1;             /*    OE10       */
                          unsigned char OE9 :1;             /*    OE9        */
                          unsigned char OE8 :1;             /*    OE8        */
                          unsigned char OE7 :1;             /*    OE7        */
                          unsigned char OE6 :1;             /*    OE6        */
```

```
            unsigned char OE5 :1;                   /*    OE5        */
            unsigned char OE4 :1;                   /*    OE4        */
            unsigned char OE3 :1;                   /*    OE3        */
            unsigned char OE2 :1;                   /*    OE2        */
            unsigned char OE1 :1;                   /*    OE1        */
            unsigned char OE0 :1;                   /*    OE0        */
            }       BIT;                            /*               */
      }             OER;                            /*               */
union {                                             /* PWDPR         */
      unsigned int WORD;                            /*  Word Access  */
      struct {                                      /*  Byte Access  */
            unsigned char B;                        /*    PWDPRB     */
            unsigned char A;                        /*    PWDPRA     */
            }       BYTE;                           /*               */
      struct {                                      /*  Bit  Access  */
            unsigned char OS15:1;                   /*    OS15       */
            unsigned char OS14:1;                   /*    OS14       */
            unsigned char OS13:1;                   /*    OS13       */
            unsigned char OS12:1;                   /*    OS12       */
            unsigned char OS11:1;                   /*    OS11       */
            unsigned char OS10:1;                   /*    OS10       */
            unsigned char OS9 :1;                   /*    OS9        */
            unsigned char OS8 :1;                   /*    OS8        */
            unsigned char OS7 :1;                   /*    OS7        */
            unsigned char OS6 :1;                   /*    OS6        */
            unsigned char OS5 :1;                   /*    OS5        */
            unsigned char OS4 :1;                   /*    OS4        */
            unsigned char OS3 :1;                   /*    OS3        */
            unsigned char OS2 :1;                   /*    OS2        */
            unsigned char OS1 :1;                   /*    OS1        */
            unsigned char OS0 :1;                   /*    OS0        */
            }       BIT;                            /*               */
      }             DPR;                            /*               */
union {                                             /* PWSL          */
      unsigned char BYTE;                           /*  Byte Access  */
      struct {                                      /*  Bit  Access  */
            unsigned char PWCKE:1;                  /*    PWCKE      */
```

```
                        unsigned char PWCKS:1;          /*    PWCKE      */
                        unsigned char      :2;          /*               */
                        unsigned char RS   :4;          /*    RS */
                        }       BIT;                    /*               */
                  }             SL;                     /*               */
            unsigned char      DR;                      /* PWDR0-PWDR15  */
};                                                      /*               */
struct st_hif {                                         /* struct HIF    */
            union {                                     /* SYSCR2        */
                  unsigned char BYTE;                   /*  Byte Access  */
                  struct {                              /*  Bit  Access  */
                        unsigned char wk   :7;          /*               */
                        unsigned char HI12E:1;          /*    HI12E      */
                        }       BIT;                    /*               */
                  }             SYSCR2;                 /*               */
            char                 wk[108];               /*               */
            union {                                     /*  HICR         */
                  unsigned char BYTE;                   /*  Byte Access  */
                  struct {                              /*  Bit  Access  */
                        unsigned char wk    :5;         /*               */
                        unsigned char IBFIE2:1;         /*    IBFIE2     */
                        unsigned char IBFIE1:1;         /*    IBFIE1     */
                        unsigned char FGA2OE:1;         /*    FGA2OE     */
                        }       BIT;                    /*               */
                     }             HICR;                /*               */
};                                                      /*               */
struct st_hif1 {                                        /* struct HIF1   */
             unsigned char   IDR;                       /* IDR           */
             unsigned char   ODR;                       /* ODR           */
             union {                                    /* STR           */
                   unsigned char BYTE;                  /*  Byte Access  */
                   struct {                             /*  Bit  Access  */
                         unsigned char DBU7:1;          /*    DBU        */
                         unsigned char DBU6:1;          /*    DBU        */
                         unsigned char DBU5:1;          /*    DBU        */
                         unsigned char DBU4:1;          /*    DBU        */
                         unsigned char CD  :1;          /*    C/D        */
```

```
                        unsigned char DBU2:1;                   /*    DBU        */
                        unsigned char IBF :1;                   /*    IBF        */
                        unsigned char OBF :1;                   /*    OBF        */
                        }       BIT;                            /*               */
                char                    wk2[5];                 /*               */
};                                                              /*               */
union un_sbycr {                                                /* union SBYCR   */
                unsigned char BYTE;                             /* Byte Access   */
                struct {                                        /* Bit  Access   */
                       unsigned char SSBY :1;                   /*    SSBY       */
                       unsigned char STS  :3;                   /*    STS        */
                       unsigned char      :1;                   /*               */
                       unsigned char SCK  :3;                   /*    SCK        */
                       }       BIT;                             /*               */
};                                                              /*               */
union un_lpwrcr {                                               /* union LPWRCR  */
                 unsigned char BYTE;                            /* Byte Access   */
                 struct {                                       /* Bit Access    */
                        unsigned char DTON :1;                  /*    DTON       */
                        unsigned char LSON :1;                  /*    LSON       */
                        unsigned char NESEL:1;                  /*    NESEL      */
                        unsigned char EXCLE:1;                  /*    EXCLE      */
                        }       BIT;                            /*               */
};                                                              /*               */
union un_mstpcr {                                               /* union MSTPCR  */
                 unsigned int WORD;                             /* Word Access   */
                 struct {                                       /* Byte Access   */
                        unsigned char H;                        /*    MSTPCRH    */
                        unsigned char L;                        /*    MSTPCRL    */
                        }       BYTE;                           /*               */
                 struct {                                       /* Bit Access    */
                        unsigned char wk :1;                    /*               */
                        unsigned char B14:1;                    /*    DTC        */
                        unsigned char B13:1;                    /*    FRT        */
                        unsigned char B12:1;                    /*    TMR0,TMR1  */
                        unsigned char B11:1;                    /*    PWM,PWMX   */
                        unsigned char B10:1;                    /*    D/A        */
```

```
                        unsigned char B9 :1;                    /*    A/D       */
                        unsigned char B8 :1;                    /*    TMRX,TMRY */
                        unsigned char B7 :1;                    /*    SCI0      */
                        unsigned char B6 :1;                    /*    SCI1      */
                        unsigned char B5 :1;                    /*    SCI2      */
                        unsigned char B4 :1;                    /*    IIC0      */
                        unsigned char B3 :1;                    /*    IIC1      */
                        unsigned char B2 :1;                    /*    HIF       */
                        }       BIT;                            /*              */
    };                                                          /*              */
    union un_stcr {                                             /* union STCR   */
                  unsigned char BYTE;                           /* Byte Access  */
                  struct {                                      /* Bit  Access  */
                         unsigned char IICS :1;                 /*    IICS      */
                         unsigned char IICX1:1;                 /*    IICX1     */
                         unsigned char IICX0:1;                 /*    IICX0     */
                         unsigned char IICE :1;                 /*    IICE      */
                         unsigned char FLSHE:1;                 /*    FLSHE     */
                         unsigned char      :1;                 /*              */
                         unsigned char ICKS1:1;                 /*    ICKS1     */
                         unsigned char ICKS0:1;                 /*    ICKS0     */
                         }       BIT;                           /*              */
    };                                                          /*              */
    union un_syscr {                                            /* union SYSCR  */
                   unsigned char BYTE;                          /* Byte Access  */
                   struct {                                     /* Bit  Access  */
                          unsigned char CS2E :1;                /*    CS2E      */
                          unsigned char IOSE :1;                /*    IOSE      */
                          unsigned char INTM :2;                /*    INTM      */
                          unsigned char XRST :1;                /*    XRST      */
                          unsigned char NMIEG:1;                /*    NMIEG     */
                          unsigned char HIE  :1;                /*    HIE       */
                          unsigned char RAME :1;                /*    RAME      */
                          }       BIT;                          /*              */
    };                                                          /*              */
    union un_mdcr {                                             /* union MDCR   */
                  unsigned char BYTE;                           /* Byte Access  */
```

```
             struct {                                   /* Bit  Access   */
                    unsigned char EXPE:1;               /*    EXPE       */
                    unsigned char     :5;               /*               */
                    unsigned char MDS :2;               /*    MDS        */
                    }       BIT;                        /*               */
};                                                      /*               */
union st_sci {                                          /* struct SCI    */
             union {                                    /* SMR           */
                   unsigned char BYTE;                  /* Byte Access   */
                   struct {                             /* Bit  Access   */
                          unsigned char CA  :1;         /*    C/A        */
                          unsigned char CHR :1;         /*    CHR        */
                          unsigned char PE  :1;         /*    PE         */
                          unsigned char OE  :1;         /*    O/E        */
                          unsigned char STOP:1;         /*    STOP       */
                          unsigned char MP  :1;         /*    MP         */
                          unsigned char CKS :2;         /*    CKS        */
                          }       BIT;                  /*               */
                   }       SMR;                         /*               */
             unsigned char BRR;                         /* BRR           */
             union {                                    /* SCR           */
                   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;                  /*               */
                   }       SCR;                         /*               */
             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 ORER:1;                   /*    ORER       */
                        unsigned char FER :1;                   /*    FER        */
                        unsigned char PER :1;                   /*    PER        */
                        unsigned char TEND:1;                   /*    TEND       */
                        unsigned char MPB :1;                   /*    MPB        */
                        unsigned char MPBT:1;                   /*    MPBT       */
                        }       BIT;                            /*               */
                  }             SSR;                            /*               */
            unsigned char       RDR;                            /* RDR           */
            union {                                             /* SCMR          */
                  unsigned char BYTE;                           /*  Byte Access  */
                  struct {                                      /*  Bit  Access  */
                        unsigned char wk  :4;                   /*               */
                        unsigned char SDIR:1;                   /*    SDIR       */
                        unsigned char SINV:1;                   /*    SINV       */
                        unsigned char     :1;                   /*               */
                        unsigned char SMIF:1;                   /*    SMIF       */
                        }       BIT;                            /*               */
                  }             SCMR;                           /*               */
};                                                              /*               */
union st_frt {                                                  /* struct FRT    */
            union {                                             /* TIER          */
                  unsigned char BYTE;                           /*  Byte Access  */
                  struct {                                      /*  Bit  Access  */
                        unsigned char ICIAE:1;                  /*    ICIAE      */
                        unsigned char ICIBE:1;                  /*    ICIBE      */
                        unsigned char ICICE:1;                  /*    ICICE      */
                        unsigned char ICIDE:1                   /*    ICIDE      */
                        unsigned char OCIAE:1;                  /*    OCIAE      */
                        unsigned char OCIBE:1;                  /*    OCIBE      */
                        unsigned char OVIE :1;                  /*    OVIE       */
                        }       BIT;                            /*               */
                  }             TIER;                           /*               */
            union {                                             /* TCSR          */
                  unsigned char BYTE;                           /*  Byte Access  */
                  struct {                                      /*  Bit  Access  */
```

```
                unsigned char ICFA :1;                  /*    ICFA       */
                unsigned char ICFB :1;                  /*    ICFB       */
                unsigned char ICFC :2;                  /*    ICFC       */
                unsigned char ICFD :1;                  /*    ICFD       */
                unsigned char OCFA :1;                  /*    OCFA       */
                unsigned char OCFB :1;                  /*    OCFB       */
                unsigned char OVF  :1;                  /*    OVF        */
                unsigned char CCLRA:1;                  /*    CCLRA      */
                }       BIT;                            /*               */
        }               TCSR;                           /*               */
  unsigned int          FRC;                            /* FRC           */
  unsigned int          OCRA;                           /* OCRA or OCRB  */
  union {                                               /* TCR           */
        unsigned char BYTE;                             /*  Byte Access  */
        struct {                                        /*  Bit  Access  */
                unsigned char IEDGA:1;                  /*    IEDGA      */
                unsigned char IEDGB:1;                  /*    IEDGB      */
                unsigned char IEDGC:1;                  /*    IEDGC      */
                unsigned char IEDGD:1;                  /*    IEDGD      */
                unsigned char BUFEA:1;                  /*    BUFEA      */
                unsigned char BUFEB:1;                  /*    BUFEB      */
                unsigned char CKS  :2;                  /*    CKS        */
                }       BIT;                            /*               */
        }               TCR;                            /*               */
  union {                                               /* TOCR          */
        unsigned char BYTE;                             /*  Byte Access  */
        struct {                                        /*  Bit Access   */
                unsigned char ICRDMS:1;                 /*    ICRDMS     */
                unsigned char OCRAMS:1;                 /*    OCRAMS     */
                unsigned char ICRS  :1;                 /*    ICRS       */
                unsigned char OCRS  :1;                 /*    OCRS       */
                unsigned char OEA   :1;                 /*    OEA        */
                unsigned char OEB   :1;                 /*    OEB        */
                unsigned char OLVLA :1;                 /*    OLVLA      */
                unsigned char OLVLB :1;                 /*    OLVLB      */
                }       BIT;                            /*               */
        }               TOCR;                           /*               */
```

```
            unsigned int        ICRA;                   /*ICRA or OCRAR  */
            unsigned int        ICRB;                   /*ICRB or OCRAF  */
            unsigned int        ICRC;                   /*ICRC or OCRDM  */
            unsigned int        ICRD;                   /* ICRD          */
};                                                      /*               */
union un_pwmx {                                         /* struct PWMX   */
            struct {                                    /*               */
                  union {                               /* DACR          */
                  unsigned char BYTE;                   /* Byte Access   */
                  struct {                              /* Bit  Access   */
                        unsigned char TEST     :1;      /*    TEST       */
                        unsigned char PWME     :1;      /*    PWME       */
                        unsigned char char     :2;      /*               */
                        unsigned char char OEB :1;      /*    OEB        */
                        unsigned char char OEA :1;      /*    OEA        */
                        unsigned char char OS  :1;      /*    OS         */
                        unsigned char char CKS :1;      /*    CKS        */
                        }       BIT;                    /*               */
                  }             ST_DACR;                /*               */
                  char          wk[5];                  /*               */
                  union {                               /* DACNT         */
                  unsigned int WORD;                    /* Word Access   */
                  struct {                              /* Bit Access    */
                        unsigned int wk  :15;           /*               */
                        unsigned int REGS: 1;           /*    REGS       */
                        }       BIT;                    /*               */
                  }             ST_DACNT;               /*               */
            }                   REGS1;                  /*               */
            struct {                                    /*               */
                  union {                               /* DADRA         */
                        unsigned int WORD;              /* Word Access   */
                        struct {                        /* Bit  Access   */
                              unsigned int wk :14;      /*               */
                              unsigned int CFS: 1;      /*    CFS        */
                              }       BIT;              /*               */
                  }                   ST_DADRA;         /*               */
            char                      wk[4];            /*               */
```

```
              union {                                   /* DADRB         */
                    unsigned int WORD;                  /* Word Access   */
                    struct {                            /* Bit  Access   */
                          unsigned int wk  :14;         /*               */
                          unsigned int CFS : 1;         /*    CFS        */
                          unsigned int REGS: 1;         /*    REGS       */
                          }       BIT;                  /*               */
                    }             ST_DADRB;             /*               */
              }                   REGSO;                /*               */
};                                                      /*               */
struct st_p1 {                                          /* struct P1     */
            union {                                     /* P1PCR         */
                  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;                    /*               */
                  }             PCR;                    /*               */
            char                wk1[3];                 /*               */
            unsigned char       DDR;                    /* P1DDR         */
            char                wk2;                    /*               */
            union {                                     /* P1DR          */
                  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;                        /*               */
                  }             DR;                         /*               */
};                                                          /*               */
struct st_p3 {                                              /* struct P3     */
            union {                                         /* P3PCR         */
                  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;                        /*               */
                  }             PCR;                        /*               */
            char                wk1[5];                     /*               */
            unsigned char       DDR;                        /* P3DDR         */
            char                wk2;                        /*               */
            union {                                         /* P3DR          */
                  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;                        /*               */
                  }             DR;                         /*               */
};                                                          /*               */
struct st_p4 {                                              /* struct P4     */
            unsigned char DDR;                              /* P4DDR         */
```

```
            char            wk;                         /*              */
            union {                                     /* P4DR         */
            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;                         /*              */
            }              DR;                          /*              */
};                                                      /*              */
struct st_p5 {                                          /* struct P5    */
            unsigned char DDR;                          /* P5DDR        */
            char            wk;                         /*              */
            union {                                     /* P5DR         */
                  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;                   /*              */
                  }              DR;                    /*              */
};                                                      /*              */
struct st_p6 {                                          /* struct P6    */
            unsigned char DDR;                          /* P6DDR        */
            char            wk1;                        /*              */
            union {                                     /* P6DR         */
                  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;                        /*               */
                  }             DR;                         /*               */
            char                wk2[54];                    /*               */
            union {                                         /* P6PCR         */
                  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;                        /*               */
                  }             PCR;                        /*               */
};                                                          /*               */
struct st_p7 {                                              /* struct P7     */
            union {                                         /* P7PIN         */
                  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;                        /*               */
                  }             PIN;                        /*               */
};                                                          /*               */
```

```
struct st_p8 {                                          /* struct P8     */
            unsigned char DDR;                          /* P8DDR         */
            char            wk;                         /*               */
            union {                                     /* P8DR          */
                  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      */
                        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;                    /*               */
                  }             DR;                     /*               */
};                                                      /*               */
struct st_p9 {                                          /* struct P9     */
            unsigned char DDR;                          /* P9DDR         */
            union {                                     /* P9DR          */
                  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;                    /*               */
                  }             DR;                     /*               */
};                                                      /*               */
struct st_bsc {                                         /* struct BSC    */
             union {                                    /* BCR           */
                   unsigned char BYTE;                  /*  Byte Access  */
                   struct {                             /*  Bit  Access  */
```

```
                        unsigned char ICIS1 :1;             /*    ICIS1      */
                        unsigned char ICIS0 :1;             /*    ICIS0      */
                        unsigned char BRSTRM:1;             /*    BRSTRM     */
                        unsigned char BRSTS1:1;             /*    BRSTS1     */
                        unsigned char BRSTS0:1;             /*    BRSTS0     */
                        unsigned char       :1;             /*               */
                        unsigned char IOS   :2;             /*    IOS        */
                        }       BIT;                        /*               */
                  }             BCR;                        /*               */
            union {                                         /* WSCR          */
                  unsigned char BYTE;                       /*  Byte Access  */
                  struct {                                  /*  Bit  Access  */
                        unsigned char RAMS:1;               /*    RAMS       */
                        unsigned char RAM0:1;               /*    RAM0       */
                        unsigned char ABW :1;               /*    ABW        */
                        unsigned char AST :1;               /*    AST        */
                        unsigned char WMS :2;               /*    WMS        */
                        unsigned char WC  :2;               /*    WC         */
                        }       BIT;                        /*               */
                  }             WSCR;                       /*               */
};                                                          /*               */
struct st_tmr {                                             /* struct TMR    */
            union {                                         /* TCR0          */
                  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;                        /*               */
                  }             TCR0;                       /*               */
            union {                                         /* TCR1          */
                  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;                    /*               */
                  }             TCR1;                   /*               */
            union {                                     /* TCSR0         */
                  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 ADTE:1;           /*    ADTE       */
                        unsigned char OS  :4;           /*    OS         */
                        }       BIT;                    /*               */
                  }             TCSR0;                  /*               */
            union {                                     /* TCSR1         */
                  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;                    /*               */
                  }             TCSR1;                  /*               */
            unsigned int        TCORA;                  /* TCORA         */
            unsigned int        TCORB;                  /* TCORB         */
            unsigned int        TCNT;                   /* TCNT          */
};                                                      /*               */
struct st_tmr0 {                                        /* struct TMR0   */
             union {                                    /* TCR           */
                   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;                    /*              */
                  }             TCR;                    /*              */
            char                wk;                     /*              */
            union {                                     /* TCSR         */
                  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 ADTE:1;           /*    ADTE      */
                        unsigned char OS  :4;           /*    OS        */
                        }       BIT;                    /*              */
                  }             TCSR;                   /*              */
            char                wk2;                    /*              */
            unsigned char       TCORA;                  /* TCORA        */
            char                wk3;                    /*              */
            unsigned char       TCORB;                  /* TCORB        */
            char                wk4;                    /*              */
            unsigned char       TCNT;                   /* TCNT         */
};                                                      /*              */
struct st_tmr1 {                                        /* struct TMR1  */
            union {                                     /* TCR          */
                  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;                    /*              */
                  }             TCR;                    /*              */
            char                wk1;                    /*              */
            union {                                     /* TCSR         */
                  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;                    /*               */
                  }             TCSR;                   /*               */
            char                wk2;                    /*               */
            unsigned char       TCORA;                  /* TCORA         */
            char                wk3;                    /*               */
            unsigned char       TCORB;                  /* TCORB         */
            char                wk4;                    /*               */
            unsigned char       TCNT;                   /* TCNT          */
};                                                      /*               */
struct st_tmrx {                                        /* struct TMRX   */
            union {                                     /* TCR           */
                  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;                    /*               */
                  }             TCR;                    /*               */
            union {                                     /* TCSR          */
                  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 ICF :1;           /*    ICF        */
                        unsigned char OS  :4;           /*    OS         */
                        }       BIT;                    /*               */
                  }             TCSR;                   /*               */
            unsigned char       TICRR;                  /* TICRR         */
            unsigned char       TICRF;                  /* TICRF         */
            unsigned char       TCNT;                   /* TCNT          */
```

```
                unsigned char        TCORC;                  /* TCORC         */
                unsigned char        TCORA;                  /* TCORA         */
                unsigned char        TCORB;                  /* TCORB         */
  };                                                         /*               */
  struct st_tmry {                                           /* struct TMRY   */
                union {                                      /* TCR           */
                      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;                     /*               */
                      }             TCR;                     /*               */
                union {                                      /* TCSR          */
                      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 ICIE:1;            /*    ICIE       */
                            unsigned char OS  :4;            /*    OS         */
                            }       BIT;                     /*               */
                      }             TCSR;                    /*               */
                unsigned char        TCORA;                  /* TCORA         */
                unsigned char        TCORB;                  /* TCORB         */
                unsigned char        TCNT;                   /* TCNT          */
                union {                                      /* TISR          */
                      unsigned char BYTE;                    /*  Byte Access  */
                      struct {                               /*  Bit  Access  */
                            unsigned char wk:7;              /*               */
                            unsigned char IS:1;              /*    IS         */
                            }       BIT;                     /*               */
                      }             TISR;                    /*               */
  };                                                         /*               */
  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 TRGS:2;          /*    TRGS        */
                        }       BIT;                   /*                */
                  }             CR;                    /*                */
};                                                     /*                */
struct st_da {                                         /* struct D/A    */
            unsigned char      DR0;                    /* DADR0         */
            unsigned char      DR1;                    /* DADR1         */
            union {                                    /* DACR          */
                  unsigned char BYTE;                  /*  Byte Access  */
                  struct {                             /*  Bit  Access  */
                        unsigned char DA0E1:1;         /*    DA0E1       */
                        unsigned char DA0E0:1;         /*    DA0E0       */
                        unsigned char DAE  :1;         /*    DAE         */
                        }       BIT;                   /*                */
                  }             CR;                    /*                */
};                                                     /*                */
struct st_tc {                                         /* struct TC     */
            union {                                    /* TCONRI        */
                  unsigned char BYTE;                  /*  Byte Access  */
```

```
        struct {                                /*  Bit  Access  */
               unsigned char SIMOD:2;           /*    SIMOD      */
               unsigned char SCONE:1;           /*    SCONE      */
               unsigned char ICST :1;           /*    ICST       */
               unsigned char HFINV:1;           /*    HFINV      */
               unsigned char VFINV:1;           /*    VFINV      */
               unsigned char HIINV:1;           /*    HIINV      */
               unsigned char VIINV:1;           /*    VIINV      */
               }       BIT;                     /*               */
        }              TCONRI;                  /*               */
  union {                                       /* TCONR0        */
        unsigned char BYTE;                     /*  Byte Access  */
        struct {                                /*  Bit  Access  */
               unsigned char HOE   :1;          /*    HOE        */
               unsigned char VOE   :1;          /*    VOE        */
               unsigned char CLOE  :1;          /*    CLOE       */
               unsigned char CBOE  :1;          /*    CBOE       */
               unsigned char HOINV :1;          /*    HOINV      */
               unsigned char VOINV :1;          /*    VOINV      */
               unsigned char CLOINV:1;          /*    CLOINV     */
               unsigned char CBOINV:1;          /*    CBOINV     */
               }       BIT;                     /*               */
        }              TCONR0;                  /*               */
  union {                                       /* TCONRS        */
        unsigned char BYTE;                     /*  Byte Access  */
        struct {                                /*  Bit  Access  */
               unsigned char TMRXY :1;          /*    TMRXY      */
               unsigned char ISGENE:1;          /*    ISGENE     */
               unsigned char HOMOD :2;          /*    HOMOD      */
               unsigned char VOMOD :2;          /*    VOMOD      */
               unsigned char CLMOD :2;          /*    CLMOD      */
               }       BIT;                     /*               */
        }              TCONRS;                  /*               */
  union {                                       /* SEDGR         */
        unsigned char BYTE;                     /*  Byte Access  */
        struct {                                /*  Bit  Access  */
               unsigned char VEDG :1;           /*    VEDG       */
```

```
                        unsigned char HEDG :1;                  /*    HEDG       */
                        unsigned char CEDG :1;                  /*    CEDG       */
                        unsigned char HFEDG:1;                  /*    HFEDG      */
                        unsigned char VFEDG:1;                  /*    VFEDG      */
                        unsigned char PREQF:1;                  /*    PREQF      */
                        unsigned char IHI  :1;                  /*    IHI        */
                        unsigned char IVI  :1;                  /*    IVI        */
                        }       BIT;                            /*               */
                    }           SEDGR;                          /*               */
};                                                              /*               */
#define KBCOMP   (*(volatile union  un_kbcomp*)0xFFFEE4)        /* KBCOMP Address */
#define IIC0     (*(volatile struct st_iic0  *)0xFFFFD8)        /* IIC0 Address   */
#define IIC1     (*(volatile struct st_iic1  *)0XFFFF88)        /* IIC1 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 DDCSWR   (*(volatile union  un_ddcswr*)0xFFFEE6)        /* DDCSWR Address */
#define INTC     (*(volatile struct st_intc  *)0xFFFEE8)        /* INTC Address   */
#define DTC      (*(volatile struct st_dtc   *)0xFFFEEE)        /* DTC  Address   */
#define FLASH    (*(volatile struct st_flash *)0xFFFF80)        /* FLASH Address  */
#define PWM      (*(volatile struct st_pwm   *)0xFFFF82)        /* PWM   Address  */
#define HIF      (*(volatile struct st_hif   *)0xFFFF83)        /* HIF  Address   */
#define HIF1     (*(volatile struct st_hif1  *)0xFFFFF4)        /* HIF1 Address   */
#define HIF2     (*(volatile struct st_hif2  *)0xFFFFFC)        /* HIF1 Address   */
#define SBYCR    (*(volatile union  un_sbycr *)0xFFFF84)        /* SBYCR Address  */
#define LPWRCR   (*(volatile union  un_lpwrcr*)0xFFFF85)        /* LPWRCR Address */
#define MSTPCR   (*(volatile union  un_mstpcr*)0xFFFF86)        /* MSTPCR Address */
#define STCR     (*(volatile union  un_stcr  *)0xFFFFC3)        /* STCR Address   */
#define SYSCR    (*(volatile union  un_syscr *)0xFFFFC4)        /* SYSCR Address  */
#define MDCR     (*(volatile union  un_mdcr  *)0xFFFFC5)        /* MDCR Address   */
#define SCI0     (*(volatile struct st_sci0  *)0xFFFFD8)        /* SCI0 Address   */
#define SCI1     (*(volatile struct st_sci1  *)0xFFFF88)        /* SCI1 Address   */
#define SCI2     (*(volatile struct st_sci2  *)0xFFFFA0)        /* SCI2 Address   */
#define FRT      (*(volatile struct st_frt   *)0xFFFF90)        /* FRT  Address   */
#define OCRB     OCRA                                           /* OCRB  Change   */
#define OCRAR    ICRA                                           /* OCRAR Change   */
```

```
#define OCRAF    ICRB                                          /* OCRAF Change   */
#define OCRDM    ICRC                                          /* OCRDM Change   */
#define PWMX     (*(volatile union  un_pwmx  *)0xFFFFA0)       /* PWMX Address   */
#define DACR     REGS1.ST_DACR                                 /* DACR  Change   */
#define DACNT    REGS1.ST_DACNT                                /* DACNT Change   */
#define DADRA    REGS0.ST_DADRA                                /* DADRA Change   */
#define DADRB    REGS0.ST_DADRB                                /* DADRB Change   */
#define P1       (*(volatile struct st_p1    *)0xFFFFAC)       /* P1   Address   */
#define P2       (*(volatile struct st_p2    *)0xFFFFAD)       /* P2   Address   */
#define P3       (*(volatile struct st_p3    *)0xFFFFAE)       /* P3   Address   */
#define P4       (*(volatile struct st_p4    *)0xFFFFB5)       /* P4   Address   */
#define P5       (*(volatile struct st_p5    *)0xFFFFB8)       /* P5   Address   */
#define P6       (*(volatile struct st_p6    *)0xFFFFB9)       /* P6   Address   */
#define P7       (*(volatile struct st_p7    *)0xFFFFBE)       /* P7   Address   */
#define P8       (*(volatile struct st_p8    *)0xFFFFBD)       /* P8   Address   */
#define P9       (*(volatile struct st_p9    *)0xFFFFC0)       /* P9   Address   */
#define BSC      (*(volatile struct st_bsc   *)0xFFFFC6)       /* BSC  Address   */
#define TMR      (*(volatile struct st_tmr   *)0xFFFFC8)       /* TMR  Address   */
#define TMR0     (*(volatile struct st_tmr0  *)0xFFFFC8)       /* TMR0 Address   */
#define TMR1     (*(volatile struct st_tmr1  *)0xFFFFC9)       /* TMR1 Address   */
#define TMRX     (*(volatile struct st_tmrx  *)0xFFFFF0)       /* TMRX Address   */
#define TMRY     (*(volatile struct st_tmry  *)0xFFFFF0)       /* TMRY Address   */
#define AD       (*(volatile struct st_ad    *)0xFFFFE0)       /* A/D  Address   */
#define DA       (*(volatile struct st_da    *)0xFFFFF8)       /* D/A  Address   */
#define TC       (*(volatile struct st_tc    *)0xFFFFFC)       /* TC   Address   */
#define st_hif2 st_hif1                                  /*Change Struct HIF2      */
#define st_p2   st_p1                                    /*Change Struct P2->P1    */
```

4.1.4 Description of the Inclusion of Assembler Files in C Language Programs

The technique of including assembler files in C language programs enables us, within a C-language program, to carry out such processes as initializing the contents of the stack by using assembly language. This technique is used in the program listings of the example applications.

The C-compiler (CH38.EXE) is unable to directly generate object files from assembly language. Assembling an assembly-language file, therefore, must generate the object file. The assembly-language file is generated by using the assembler (ASM38.EXE) with the correct code option. The file's name is "sub-file name.src".

The code option must be specified as “-c=a” to generate the object file for the CH38.EXE. Refer to the manual of the compiler for more details.

4.1.5 Description of the Linkage of Files

Figure 4.3 shows the submit-file used in the linkage process. The definition file for the vector table, definition file for the registers, and each task file is linked according to the information in the submit-file. Figure 4.3 shows an example of a submit-file.

```
input SMRxd, 2138vec ……[1]….
lib c : ¥ch38¥lib¥c8s26n.lib ……[2]…
output SMRxd ……[3]…….
print SMRxd ……[4]…….
start VECT(00000), P(01000), Bramerea(0E100) ……[5]
exit
```

[1]: The object file versions of the definition file for the vector table (2138vec.obj) and task files (SMRxd.obj) are selected as the objects of the linkage.
[2]: Specifies the library (c8s26n.lib) for the H8S/2600 in its normal mode.
[3]: Specifies the object file's name (the output file is called SMRxd.abs).
[4]: Specifies the map file's name (the output file is called SMRxd.map).
[5]: Specifies the starting addresses (in this example, the vector (VECT) is allocated from H'0000, program (P) from H'1000, and data region that has not been initialized (Bramerea) from H'E100, respectively in this example).

Figure 4.3 A submit-file

4.2 Single-Master Transmission

4.2.1 Specification

- Writes 10 bytes of data to the EEPROM (HN58X2408), using channel 0 of the I²C bus interface for the H8S/2138.
- The data is written to the memory area in the address range from H'00 to H'09 in the connected EEPROM that has a slave address of [1010000].
- The data written is [H'01, H'02, H'03, H'04, H'05, H'06, H'07, H'08, H'09, and H'0A].
- The device that is connected to the I²C bus of this system has a single-master configuration. Along with the one master device (H8S/2138), there is one slave device (EEPROM).
- The frequency of the transfer clock is 100 kHz.
- Figure 4.4 illustrates the connection of the H8S/2138 with the EEPROM.

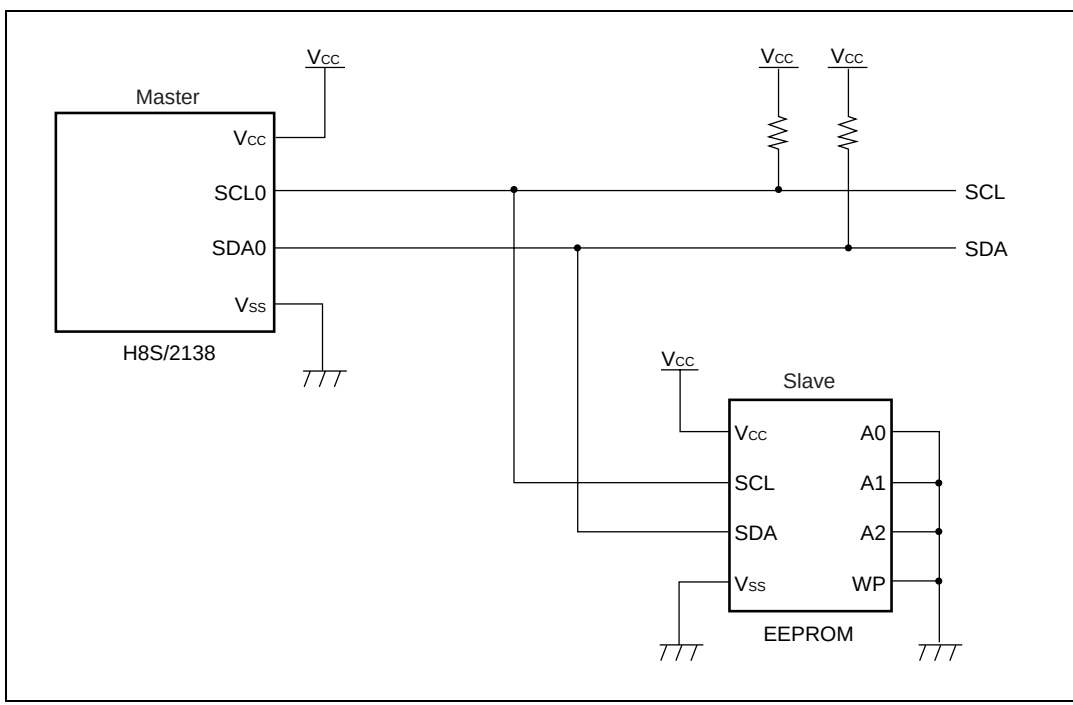


Figure 4.4 Example of the connection of the H8S/2138 with the EEPROM

- Figure 4.5 shows the I²C bus format used in the task example.

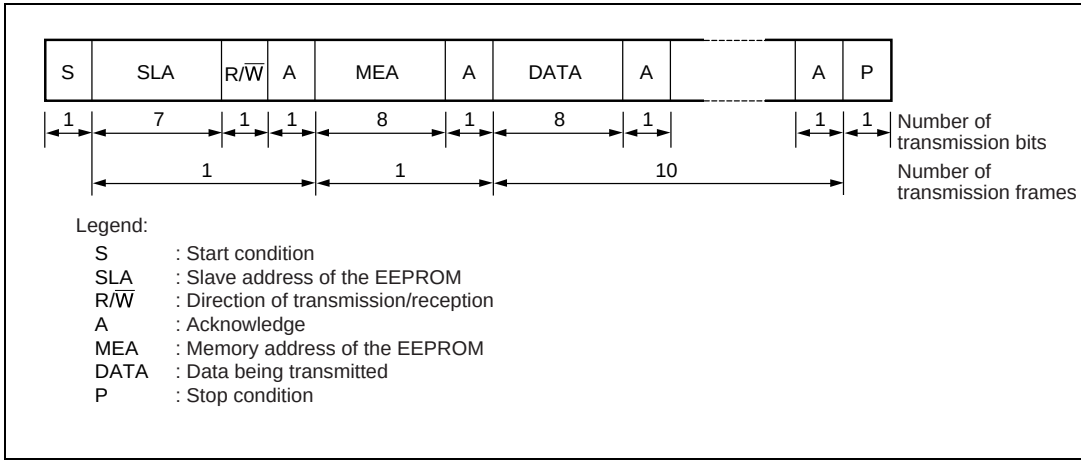
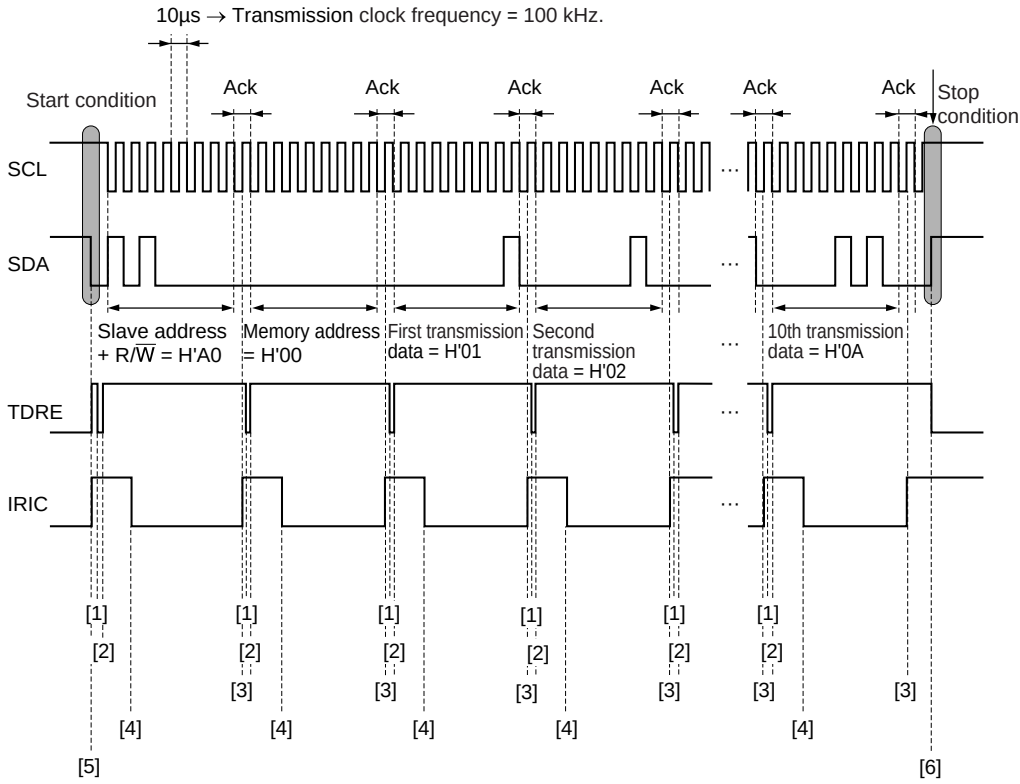


Figure 4.5 Transfer format used in the task example

4.2.2 Description of the Operation

Figure 4.6 illustrates the principle of operation of single-master transmission.



	Software processing	Hardware processing
[1]	Writes data for transmission to ICDR0	TDRE=0 (writes data to ICDRT with TRS=1)
[2]	No action	TDRE=1 (transmits data from ICDRT to ICDRS)
[3]	No action	IRIC=1 (ends data transmission (at rising of ninth clock of transmission clocks))
[4]	Clears IRIC to 0 to judge transmission end	No action
[5]	Sets start conditions (BBSY=1, SCP=0)	IRIC=1 and TDRE=1 (detects start conditions from the bus line state)
[6]	Issues stop conditions (BBSY=0, SCP=0)	TDRE=0 (detects the stop conditions from the bus line state)

Figure 4.6 Single-Master Transmission Operation Principle

4.2.3 Description of the Software

(1) Description of the Module

Table 4.1 describes the modules of this example task.

Table 4.1 Description of the modules

Module name	Label name	Functions
Main routine	main	Sets the stack pointer, and the MCU mode. Enables interrupts.
Initial setting	Intialize	Initial setting for the IIC0.
Single-master transmission	mst_trs	Uses single-master transmission to transmit 10-bytes of data to the EEPROM.
Setting the start condition	set_start	Sets the start condition.
Issuing the stop condition	set_stop	Issues the stop condition.
Transmission of the slave address + W	trs_slvadr_a0	Transmits the slave address of the EEPROM + W data (H'A0).
Transmission of the memory address of the EEPROM	trs_memadr	Transmits the memory address data of the EEPROM (H'00).

(2) Description of the on-chip registers to be used

Table 4.2 describes the on-chip registers that are used in this example task.

Table 4.2 Description of the on-chip registers

Registers		Functions	Addresses	Settings
ICDR0		Stores the data for transmission.	H'FFDE	—
SAR0	FS	Sets the FSX bit in the SARX0, the SW bit in the DDCSWR, and the transfer format.	H'FFDF bit0	0
SARX0	FSX	Sets the FS bit in the SAR0, the SW bit in the DDCSWR, and the transfer format.	H'FFDE bit0	1
ICMR0	MLS	Sets the data transfer as in the MSB-first mode.	H'FFDF bit7	0
	WAIT	Sets the continuous transfer of the data and acknowledge.	H'FFDF bit6	0
	CKS2 to CKS0	Set the frequency of the transfer clock to 100 kHz by the combination of the values in bits CKS2 to CKS0 and the IICX0 bit in the STCR.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set the number of bits per frame for the subsequent transfer of data in the I²C bus format to nine.	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Selects the access control for the registers ICMR0, ICDR0/SAR and SARX. Selects the activation (SCL0/SDA0 have port functions) or non-activation of the I²C bus interface (the SCL/SDA pins are in the bus-driven state).	H'FFD8 bit7	0/1
	IEIC	Disables the generation of interrupt requests by the I²C bus interface.	H'FFD8 bit6	0
	MST	Uses the I²C bus interface in the master mode.	H'FFD8 bit5	1
ICCR0	TRS	Uses the I²C bus interface in the transmission mode.	H'FFD8 bit4	1
	ACKE	Ceases the continuous transfer if the acknowledge bit equals 1.	H'FFD8 bit3	1
	BBSY	Determines whether or not the I²C bus is occupied. Uses the combination of the bits BBSY and SCP to issue the start or stop condition.	H'FFD8 bit2	0/1
	IRIC	Detects the start condition. Judges the end of data transmission. Detects the condition that acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Uses the combination of the bits SCP and BBSY to issue the start or stop condition.	H'FFD8 bit0	0

Table 4.2 Descriptions of Registers (cont)

Registers		Functions	Addresses	Settings
ICSR0	ACKB	Stores the acknowledge data transmitted from the EEPROM.	H'FFD9 bit0	—
STCR	IICX0	Sets the combination of values in the IICX0 bit and the bits CKS2 to CKS0 of the ICMR0 to make the frequency of the transfer clock 100 kHz.	H'FFC3 bit5	1
	IICE	Enables access to the data register and control registers of the I^2C bus interface by the CPU.	H'FFC3 bit4	1
	FLSHE	Sets the control registers for the flash memory to their non-selected state.	H'FFC3 bit3	0
DDCSWR	SWE	Inhibits automatic switching from format-less to I^2C bus format for IIC channel 0.	H'FEE6 bit7	0
	SW	Uses IIC channel 0 in the I^2C bus format.	H'FEE6 bit6	0
	IE	Inhibits an interrupt in automatic format switching.	H'FEE6 bit5	0
	CLR3 to CLR0	Control initialization of the internal state of IIC0.	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels module stop mode of SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels module stop mode of IIC channel 0.	H'FF87 bit4	0
SCR0	CKE1, 0	Set the P52/SCK0/SCL0 pin to an I/O port.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets SCI0 operating mode to asynchronous mode.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set interrupt control mode of the interrupt controller to control by bit 1.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set MCU operating mode to mode 3 by latching the input level of pins MD1 and MD0.	H'FFC5 bit1, 0	MDS1=1 MDS0=1

(3) Descriptions of variables

Table 4.3 shows the descriptions of variables in this task example.

Table 4.3 Descriptions of Variables

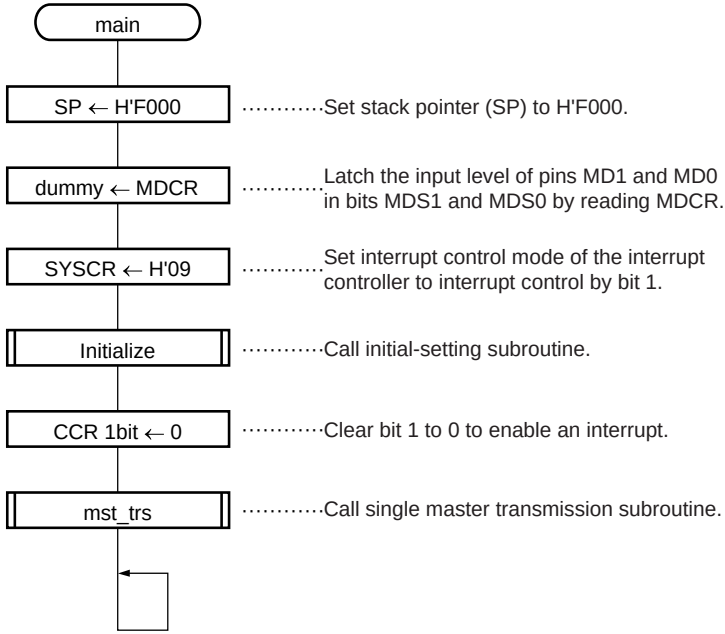
Variable	Function	Data Length	Initial Value	Used Module Name
dt_trs[0]	First-byte transmission data	1 byte	H'01	mst_trs
dt_trs[1]	Second-byte transmission data	1 byte	H'02	mst_trs
dt_trs[2]	Third-byte transmission data	1 byte	H'03	mst_trs
dt_trs[3]	Fourth-byte transmission data	1 byte	H'04	mst_trs
dt_trs[4]	Fifth-byte transmission data	1 byte	H'05	mst_trs
dt_trs[5]	Sixth-byte transmission data	1 byte	H'06	mst_trs
dt_trs[6]	Seventh-byte transmission data	1 byte	H'07	mst_trs
dt_trs[7]	Eighth-byte transmission data	1 byte	H'08	mst_trs
dt_trs[8]	Ninth-byte transmission data	1 byte	H'09	mst_trs
dt_trs[9]	Tenth-byte transmission data	1 byte	H'0A	mst_trs
i	Transmission-data counter	1 byte	H'00	mst_trs
dummy	MDCR read value	1 byte	—	main

(4) Used RAM descriptions

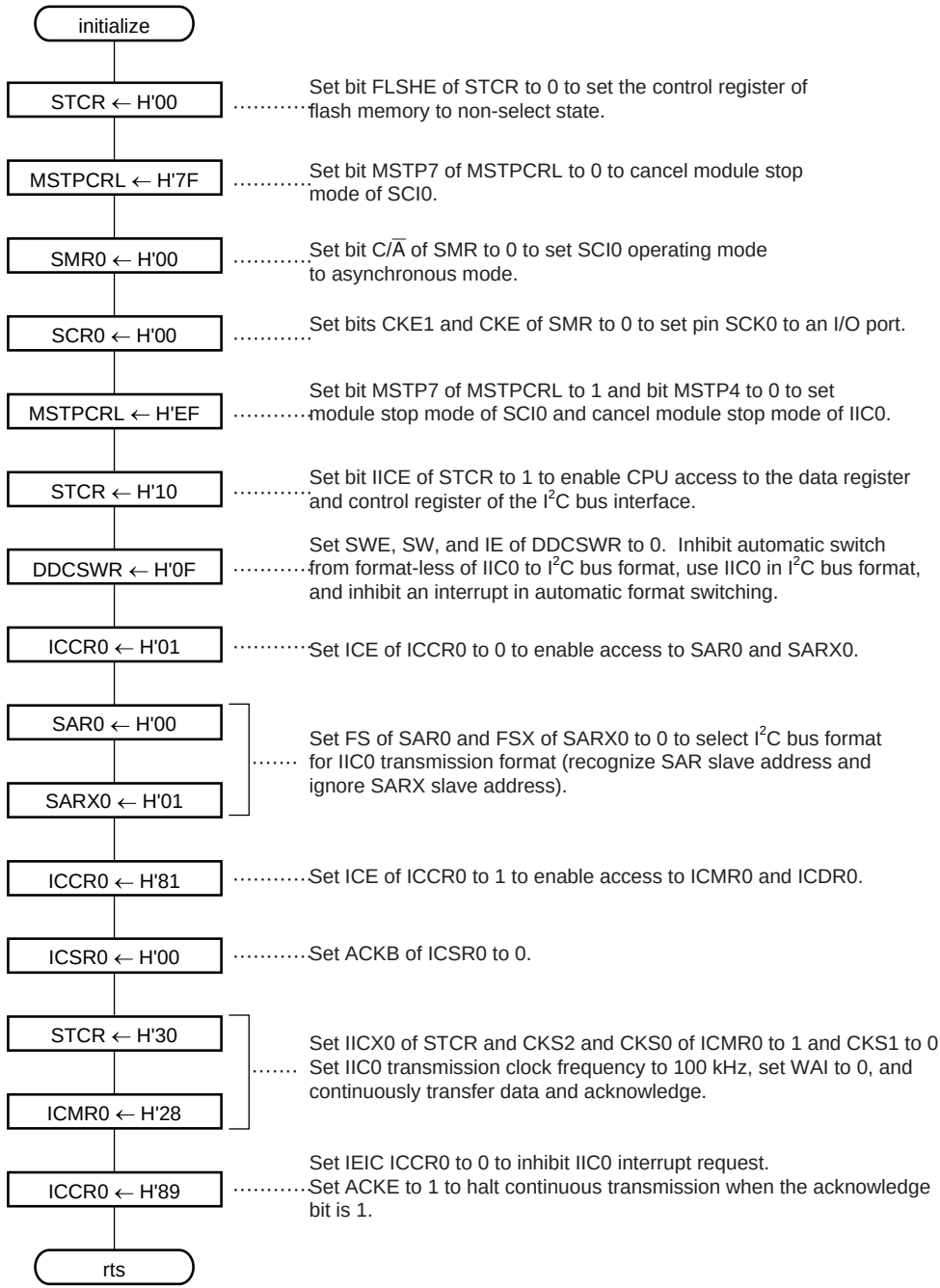
RAM for other than variables is not used in this task example.

4.2.4 Flowchart

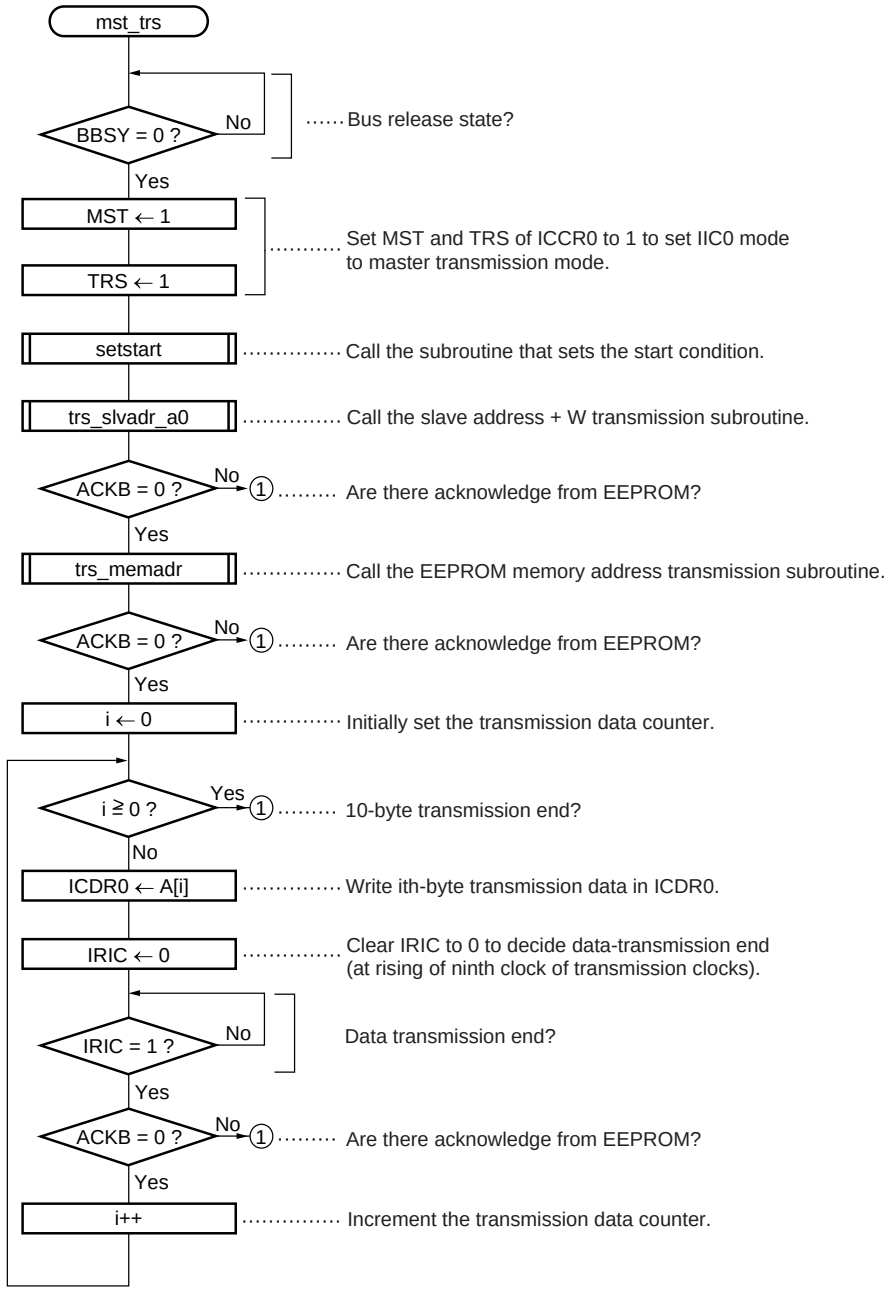
(1) Main routine

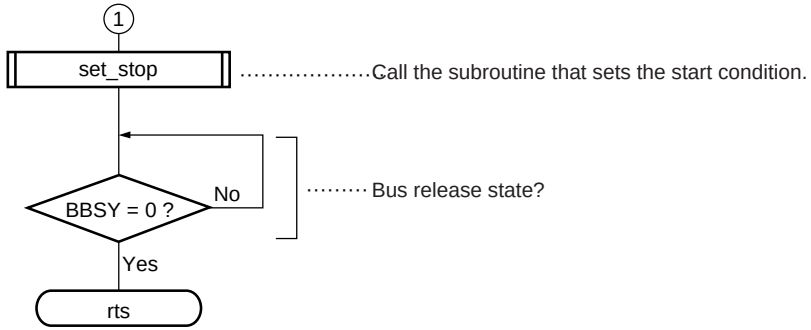


(2) Initial-setting subroutine

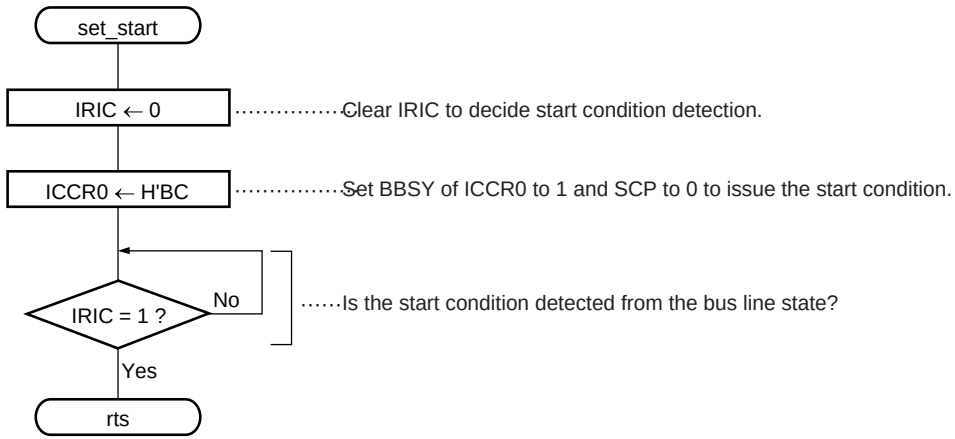


(3) Single-master transmission subroutine

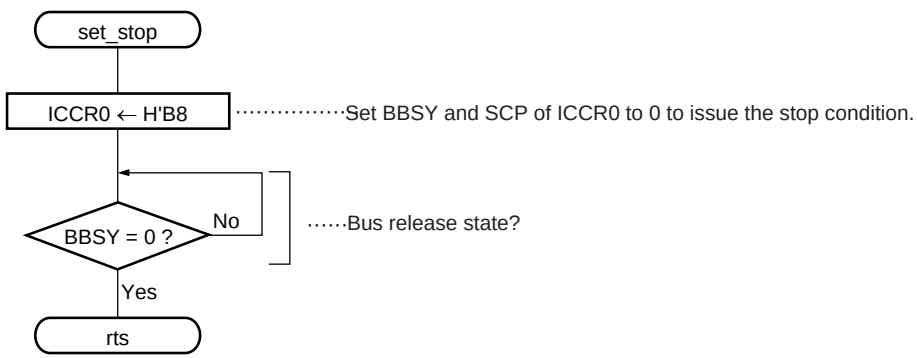




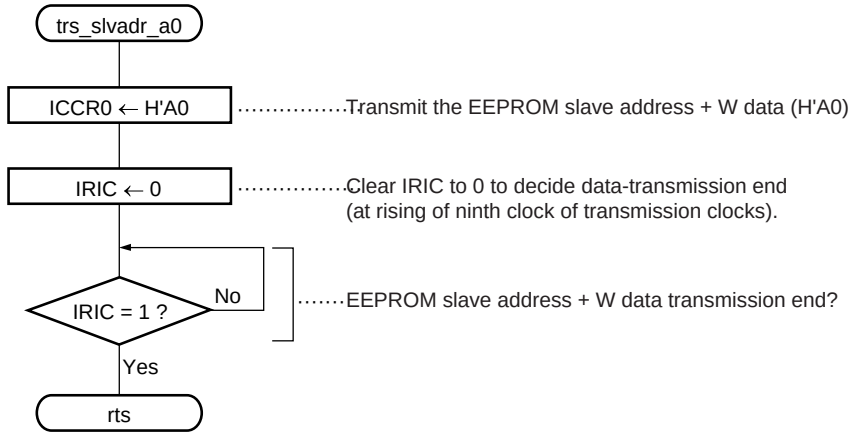
(4) Subroutine that sets the start condition



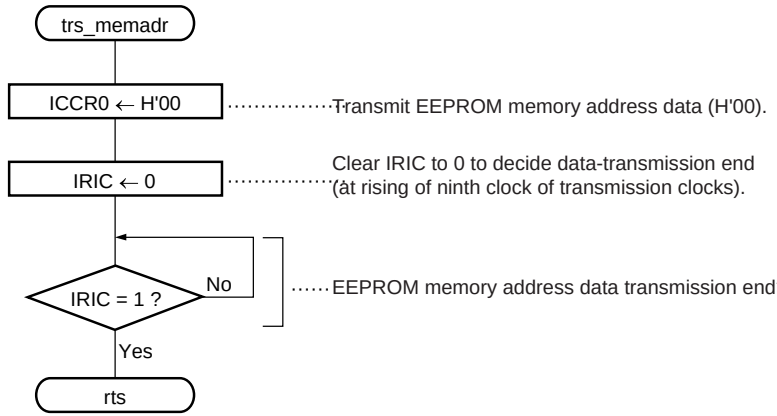
(5) Subroutine that sets the stop condition



(6) Slave address + W transmission subroutine



(7) EEPROM memory address transmission subroutine



4.2.5 Program List

```
/*****************************************************
*  H8S/2138 IIC bus application note                 *
*    1.Single master transmit to EEPROM              *
*                       File name   :  SMTxd.c        *
*                       Fai         :  20MHz          *
*                       Mode        :  3              *
*****************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*****************************************************
*  Prototype                                         *
*****************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* IIC0 initialize */
void mst_trs(void);                                 /* Master transmit to EEPROM */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_memadr(void);                              /* EEPROM memory address data transmit */

/*****************************************************
*  Data table                                        *
*****************************************************/
const unsigned char dt_trs[10] =                    /* Transmit data (10 byte) */
{
    0x01,                                           /*  1st transmit data */
    0x02,                                           /*  2nd transmit data */
    0x03,                                           /*  3rd tranmist data */
    0x04,                                           /*  4th tranmist data */
    0x05,                                           /*  5th tranmist data */
    0x06,                                           /*  6th tranmist data */
    0x07,                                           /*  7th tranmist data */
    0x08,                                           /*  8th tranmist data */
```

```
    0x09,                                   /*  9th tranmist data */
    0x0a                                    /* 10th tranmist data */
};

/*****************************************************
*  main : Main routine                                *
*****************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                 ;Stack pointer initialize
#pragma endasm
{
    unsigned char dummy;
    dummy = MDCR.BYTE;                      /* MCU mode set */

    SYSCR.BYTE = 0x09;                      /* Interrupt control mode set */
    initialize();                           /* Initialize */
    set_imask_ccr(0);                       /* Interrupt enable */
    mst_trs();                              /* Master transmit to EPROM */
    while(1);                               /* End */
}

/*****************************************************
*  initialize : IIC0 Initialize                       *
*****************************************************/
void initialize(void)
{
    STCR.BYTE = 0x00;                       /* FLSHE = 0 */
    MSTPCR.BYTE.L = 0x7f;                   /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                   /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    MSTPCR.BYTE.L = 0xef;                   /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                       /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                     /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                  /* ICE = 0 */
    IIC0.SAR.BYTE = 0x00;                   /* FS = 0 */
    IIC0.SARX.BYTE = 0x01;                  /* FSX = 1 */
```

```
    IIC0.ICCR.BYTE = 0x81;                          /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                          /* ACKB = 0 */
    STCR.BYTE = 0x30;                               /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                          /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                          /* IEIC = 0, ACKE = 1 */
}

/*************************************************
*  mst_trs : Master transmit to EEPROM            *
*************************************************/
void mst_trs(void)
{
    unsigned char i;                                /* Tranmit data counter */

    while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */

    IIC0.ICCR.BIT.MST = 1;                          /* Master transmit mode set */
    IIC0.ICCR.BIT.TRS = 1;                          /* MST = 1, TRS = 1 */
    set_start();                                    /* Start condition set */
    trs_slvadr_a0();                                /* Slave address + W data transmit */

    if(IIC0.ICSR.BIT.ACKB == 0)
    {
        trs_memadr();                               /* EEPROM memory address data transmit */

        if(IIC0.ICSR.BIT.ACKB == 0)
        {
            for(i=0; i<10; i++)
            {
                IIC0.ICDR = dt_trs[i];              /* Transmit data write */
                IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);     /* Transmit end (IRIC=1) ? */
                if(IIC0.ICSR.BIT.ACKB == 1)         /* ACKB = 0 ? */
                {
                    break;                          /* ACKB = 1 */
                }
            }
```

```
        }
    }
    set_stop();                                 /* Stop condition set */
}

/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
void set_start(void)
{
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    IIC0.ICCR.BYTE = 0xbc;                      /* Start condition set (BBSY=1,SCP=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Start condition set (IRIC=1) ? */
}

/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
    IIC0.ICCR.BYTE = 0xb8;                      /* Stop condition set (BBSY=0,SCP=0) */
    while(IIC0.ICCR.BIT.BBSY == 1);             /* Bus empty (BBSY=0) ? */
}

/*****************************************************
*  trs_slvadr_a0 : Slave addres + W data transmit    *
*****************************************************/
void trs_slvadr_a0(void)
{
    IIC0.ICDR = 0xa0;                           /* Slave address + W data(H'A0) write */
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Transmit end (IRIC=1) ? */
}
```

```
/*****************************************************
*  trs_memadr : EEPROM memory address data transmit  *
*****************************************************/
void trs_memadr(void)
{
     IIC0.ICDR = 0x00;                              /* EEPROM memory address data(H'00) write */
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                /* Transmit end (IRIC=1) ? */
}
```

4.3 Single-Master Reception

4.3.1 Specifications

- The I²C bus interface of channel 0 in H8S/2138 is used to read 10-byte data from EEPROM (HN58X2408).
- The slave address of EEPROM to be connected is “1010000”, and data is read from H'00 to H'09 of EEPROM memory addresses.
- Read data is stored in H'E100 to H'E1009 of RAM.
- Devices connected to the I²C bus of this system consist of a master device (H8S/2138) and a slave device (EEPROM) (single-master configuration).
- The frequency of a transmission clock is 100 kHz.
- Figure 4.7 shows the connection example of H8S/2138 and EEPROM.

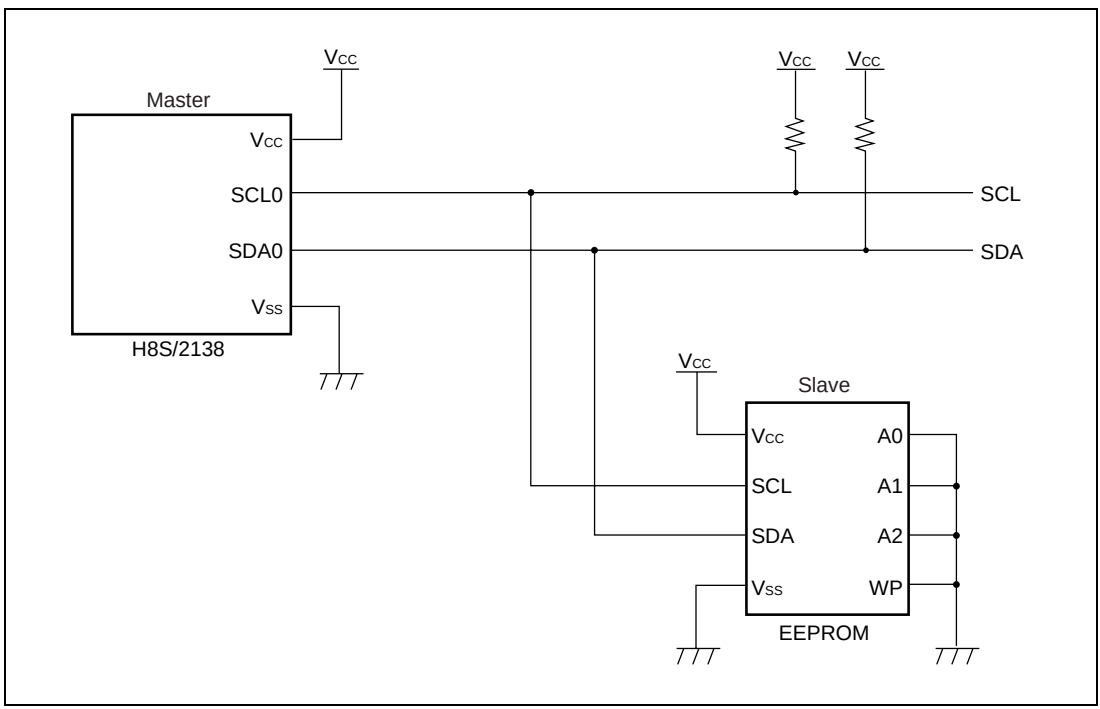
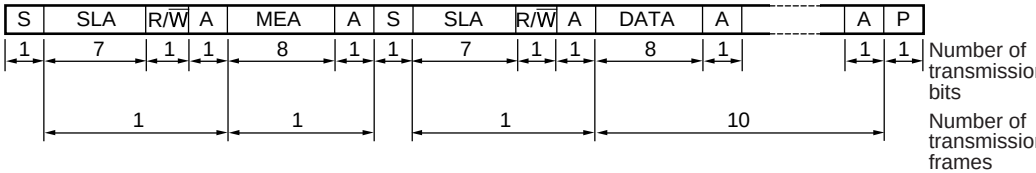


Figure 4.7 Connection Example of H8S/2138 and EEPROM

- Figure 4.8 shows the I²C bus format used in this task example.

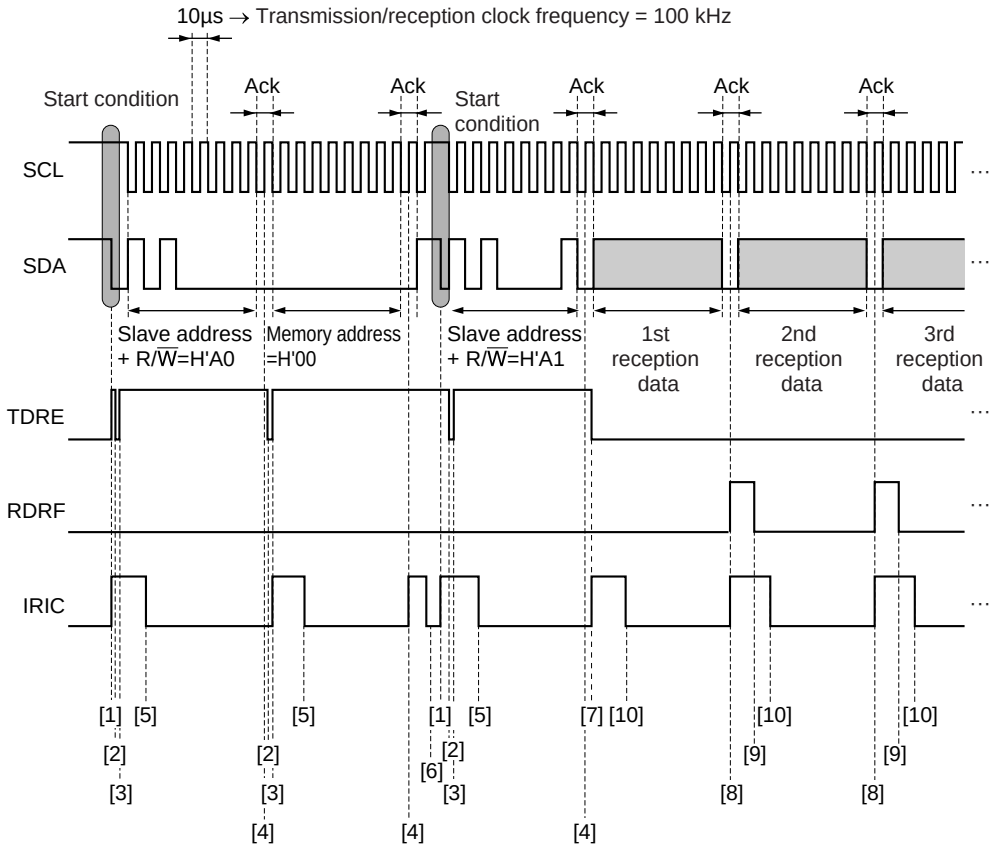


Legend:
- S : Start condition
- SLA : EEPROM slave address
- R/$\overline{W}$: Transmission/reception direction
- A : Acknowledge
- MEA : EEPROM memory address
- DATA : Reception data
- P : Stop condition

Figure 4.8 Transmission Format Used in this Task Example

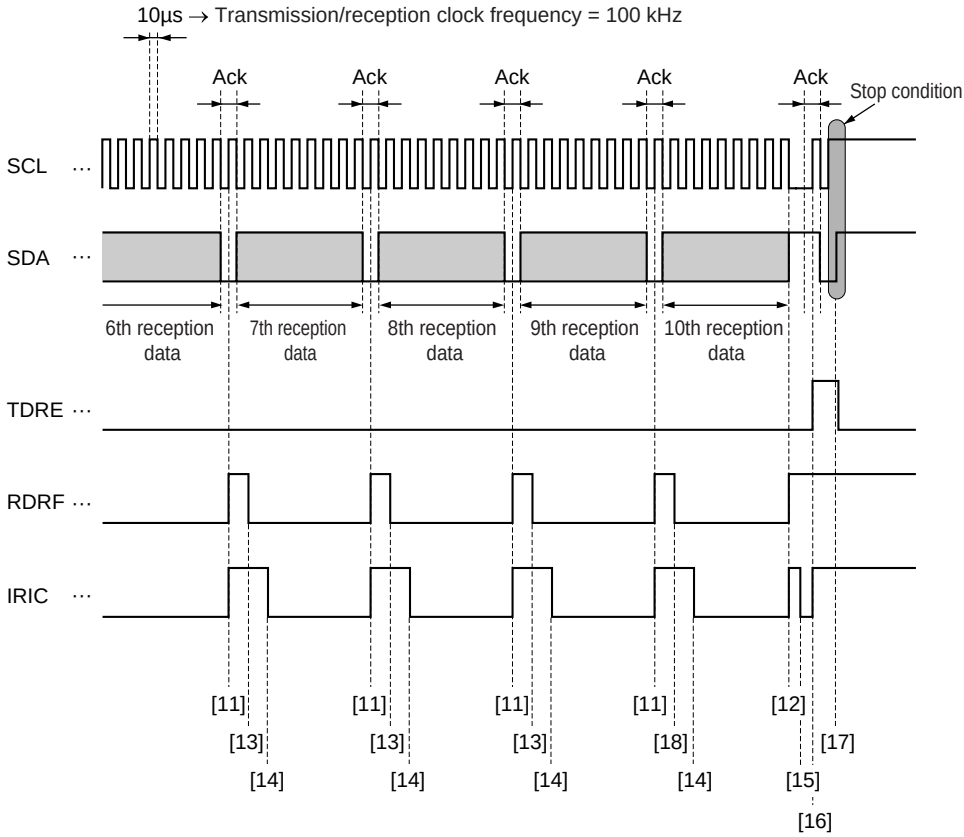
4.3.2 Operation Descriptions

Figures 4.9 and 4.10 show the operation principle.



	Software Processing	Hardware Processing
[1]	Issues start conditions (BBSY = 1, SCP = 0)	IRIC = 1, TDRE = 1 (detects start conditions from the bus line state)
[2]	Writes transmission data to ICDR0	TDRE = 0 (writes data to ICDRT when TRS = 1)
[3]	No processing	TDRE = 1 (transmits data to ICDRS from ICDRT)
[4]	No processing	IRIC = 1 (ends data transmission (at rising of 9th transmission clock))
[5]	Clears IRIC to 0 to judge transmission end	No processing
[6]	Clears IRIC to 0 to judge start condition detection	No processing
[7]	Sets master reception mode (MST = 1, TRS = 0)	TDRE = 0 (when TRS = 0)
[8]	No processing	IRIC = 1 (ends data reception (at rising of 9th reception clock))
[9]	Reads reception data from ICDR0	RDRF = 0 (reads reception data of ICDRR in reception mode)
[10]	Clears IRIC to 0 to judge reception end	No processing

Figure 4.9 Single-Master Reception Operation Principle (1)



	Software Processing	Hardware Processing
[11]	No processing	IRIC = 1, RDRF = 1 (WAIT = 0) (ends data reception (at rising of 9th reception clock))
[12]	No processing	IRIC = 1, RDRF = 1 (WAIT = 1) (ends data reception (at rising of 8th reception clock))
[13]	Reads reception data from ICDR0	RDRF = 0 (reads reception data of ICDRR in reception mode)
[14]	Clears IRIC to 0 to judge reception end	No processing
[15]	Clears IRIC to 0 to judge output end of the 9th reception clock	Starts output of 9th reception clock
[16]	Sets master transmission mode (MST = 1, TRS = 0)	IRIC = 1 (at rising of 9th reception clock) TDRE = 1 (when TRS = 0 is switched to TRS = 1 after start condition detection)
[17]	Issues stop conditions (BBSY = 0, SCP = 0)	TDRE = 0 (detects stop conditions from the bus line state after stop condition issue)

Figure 4.10 Single-Master Reception Operation Principle (2)

4.3.3 Software Descriptions

(1) Descriptions of modules

Table 4.4 shows the descriptions of modules in this task example.

Table 4.4 Descriptions of Modules

Module Name	Label Name	Function
Main routine	main	Sets stack pointer, sets MCU mode, and enables an interrupt.
Initial setting	intialize	Initially sets IIC0 and RAM area to be used.
Single master reception	mst_rec	Receives 10-byte data from EEPROM by single master reception.
Start condition issue	set_start	Issues start conditions.
Stop condition issue	set_stop	Issues stop conditions.
Slave address + W transmission	trs_slvadr_a0	Transmits slave address + W data (H'A0) of EEPROM.
Slave address + R transmission	trs_slvadr_a1	Transmits slave address + R data (H'A1) of EEPROM.
EEPROM memory address transmission	trs_memadr	Transmits memory address data (H'00) of EEPROM.
Data reception	rec_data	Receives 10-byte data.

(2) Descriptions of internal registers

Table 4.5 shows the descriptions of internal registers to be used in this task example.

Table 4.5 Descriptions of Registers

Register		Function	Address	Set Value
ICDR0		Stores reception data.	H'FFDE	—
SAR0	FS	Sets transmission format by using bit FSX of SAR0 and bit SW of DDCSWR.	H'FFDF bit0	0
SARX0	FSX	Sets transmission format by using bit FS of SAR0 and bit SW of DDCSWR.	H'FFDE bit0	1
ICMR0	MLS	Sets data transmission by MSB-first.	H'FFDF bit7	0
	WAIT	Sets whether waits are inserted between data and acknowledge.	H'FFDF bit6	0/1
	CKS2 to CKS0	Set transmission clock frequency to 100 kHz by using bit IICX0 of STCR.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set 9 bits/frame to the number of bits of data to be transmitted next in I^2C bus format.	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Selects access control of registers ICMR0, ICDR0/SAR, and SARX, and I^2C bus interface operation (port function for pin SCL0/SDA0)/non-operation (bus drive state for pin SCL/SDA).	H'FFD8 bit7	0/1
	IEIC	Inhibits I^2C bus interface interrupt requests.	H'FFD8 bit6	0
	MST	Uses the I^2C bus interface in master mode.	H'FFD8 bit5	1
	TRS	Sets transmission/reception mode of the I^2C bus interface.	H'FFD8 bit4	1/0
	ACKE	Halts continuous transmission when the acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms that the I^2C bus is occupied or released, and issues start and stop conditions by using bit SCP.	H'FFD8 bit2	0/1
	IRIC	Detects start conditions, decides data transmission end, and detects acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Issues start and stop conditions by using bit BBSY.	H'FFD8 bit0	0
ICSR0	ACKB	Stores acknowledge received from EEPROM at transmission, and sets acknowledge to be transmitted to EEPROM at reception.	H'FFD9 bit0	—

Table 4.5 Descriptions of Registers (cont)

Register		Function	Address	Set Value
STCR	IICX0	Sets transmission clock frequency to 100 kHz by using CKS2 to CKS0 of ICMR0.	H'FFC3 bit5	1
	IICE	Enables CPU access to the data register and control register of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets a non-select state to the control register of flash memory.	H'FFC3 bit3	0
DDCSWR	SWE	Inhibits automatic switching from format-less to I^2C bus format for IIC channel 0.	H'FEE6 bit7	0
	SW	Uses IIC channel 0 in the I^2C bus format.	H'FEE6 bit6	0
	IE	Inhibits an interrupt in automatic format switching.	H'FEE6 bit5	0
	CLR3 to CLR0	Control initialization of the internal state of IIC0.	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels module stop mode of SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels module stop mode of IIC channel 0.	H'FF87 bit4	0
SCR0	CKE1, 0	Set the P52/SCK0/SCL0 pin to an I/O port.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets SCI0 operating mode to asynchronous mode.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set interrupt control mode of the interrupt controller to control by bit 1.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set MCU operating mode to mode 3 by latching the input level of pins MD1 and MD0.	H'FFC5 bit1, 0	MDS1=1 MDS0=1

(3) Descriptions of variables

Table 4.6 shows the descriptions of variables in this task example.

Table 4.6 Descriptions of Variables

Variable	Function	Data Length	Initial Value	Used Module Name
dummy	MDCR read value	1 byte	—	Main
i	Transmission-data counter	1 byte	H'00	initialize rec_data

(4) Used RAM descriptions

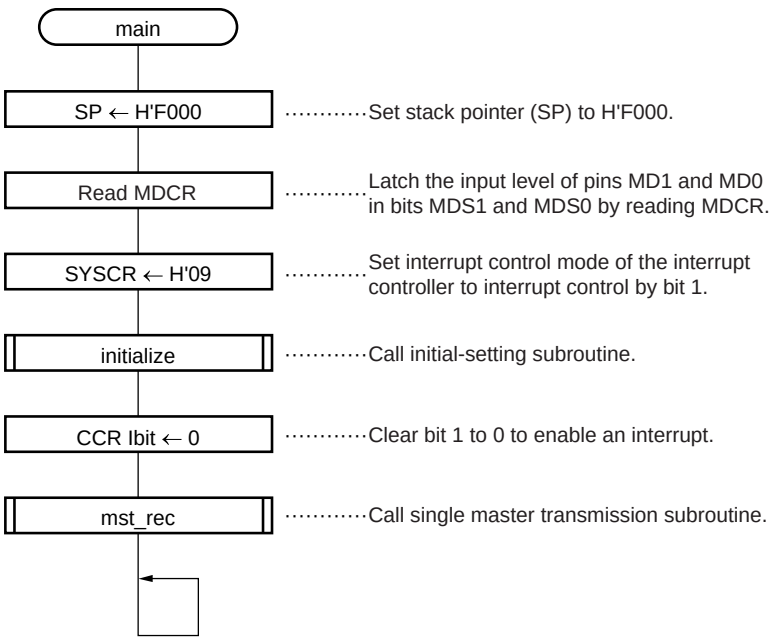
Table 4.7 shows the descriptions of used RAM in this task example.

Table 4.7 Descriptions of Used RAM

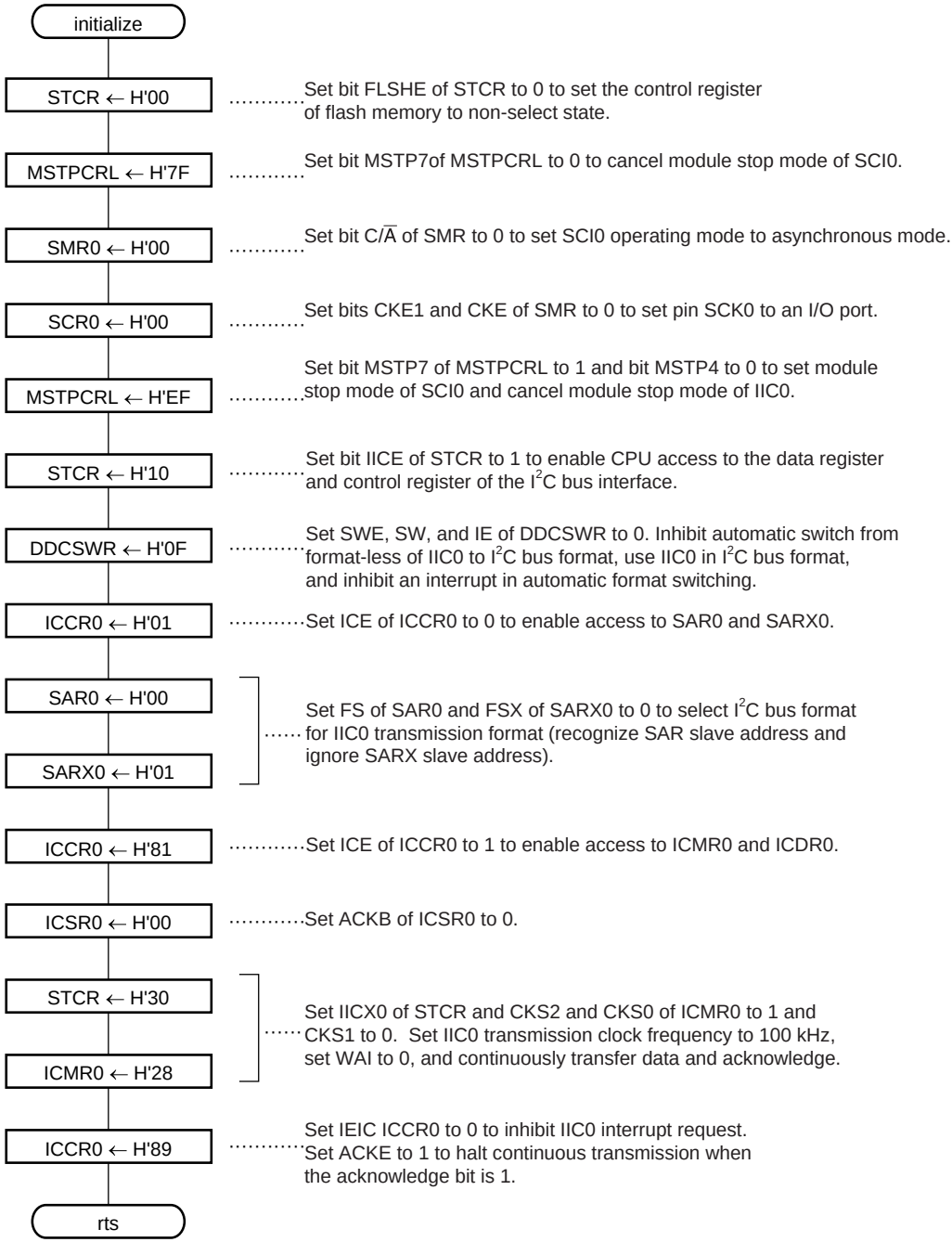
Label	Function	Data Length	Address	Used Module Name
dt_rec[i]	Stores received data	10 bytes	H'E100 to H'E109	initialize rec_data

4.3.4 Flowchart

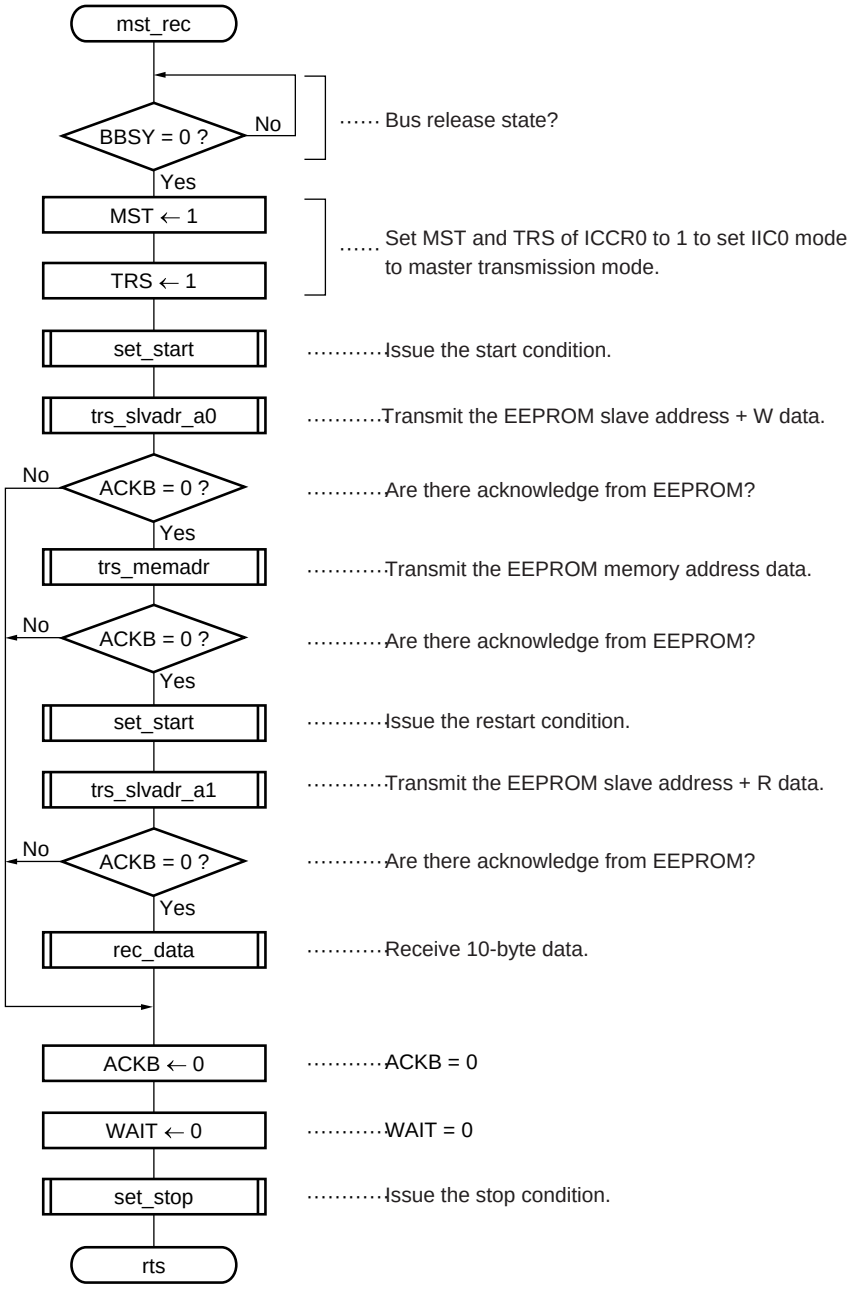
(1) Main routine



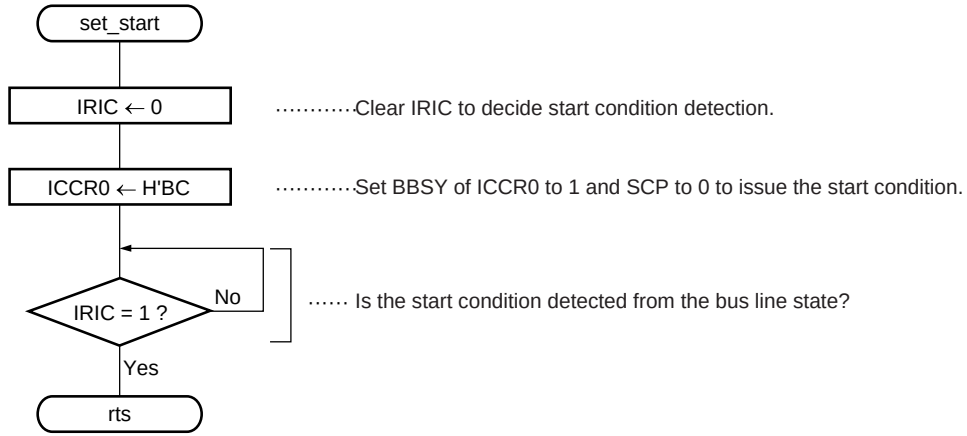
(2) Initial-setting subroutine



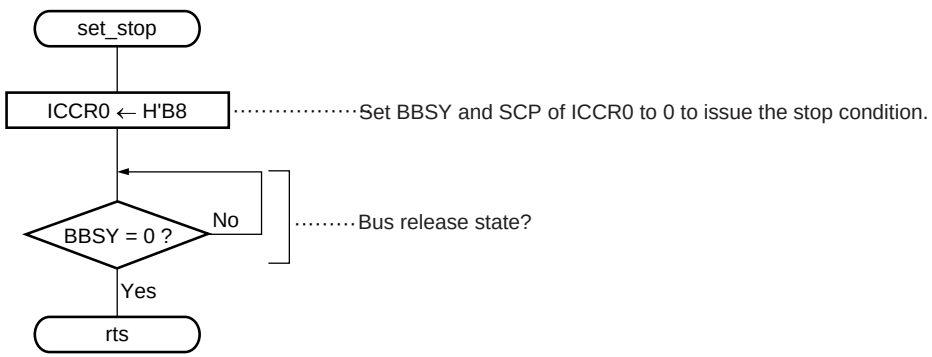
(3) Single master reception subroutine



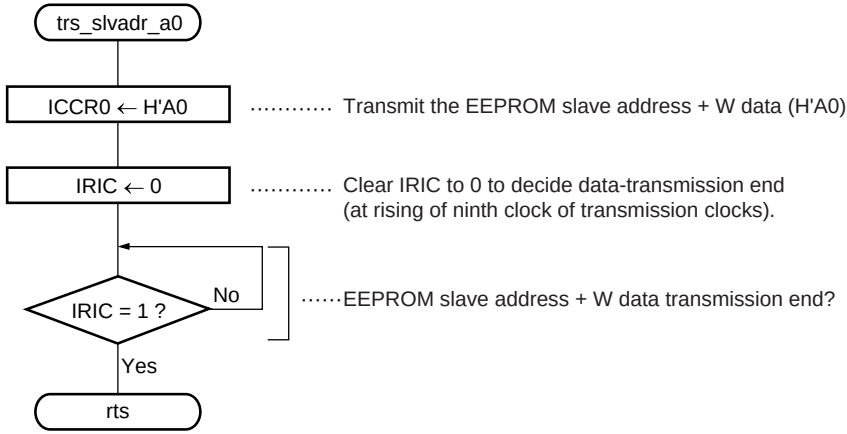
(4) Subroutine that sets the start condition



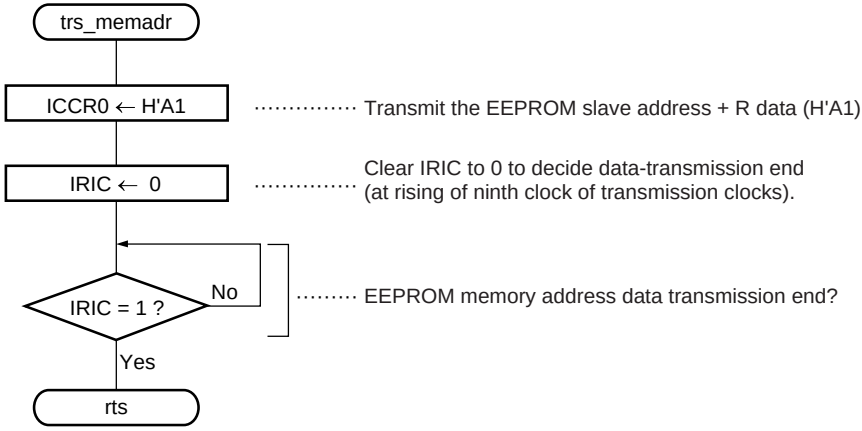
(5) Subroutine that sets the stop condition



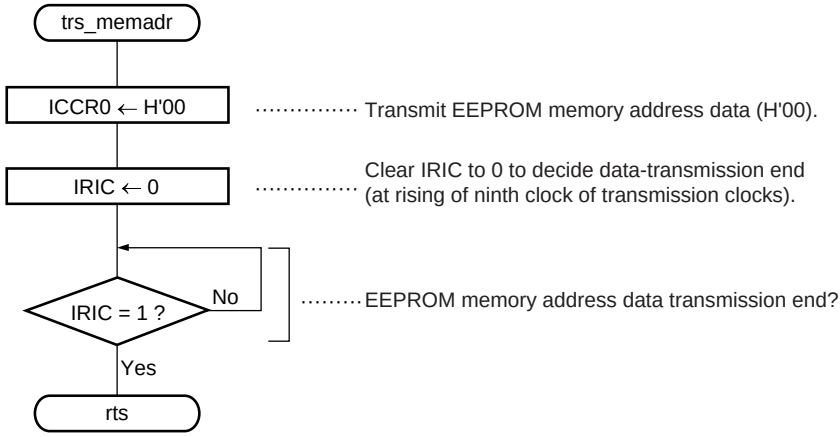
(6) Slave address + W transmission subroutine



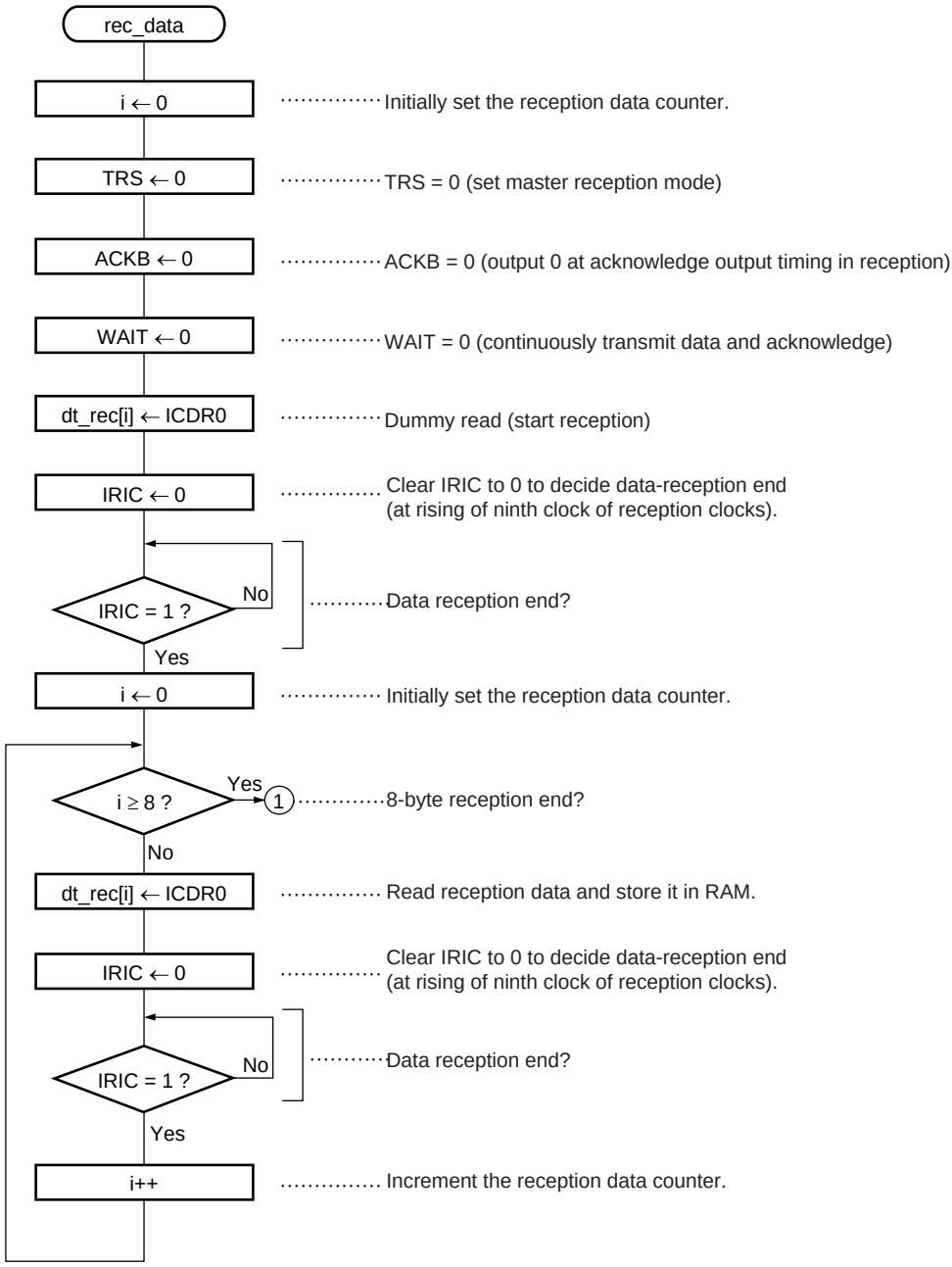
(7) Slave address + R transmission subroutine

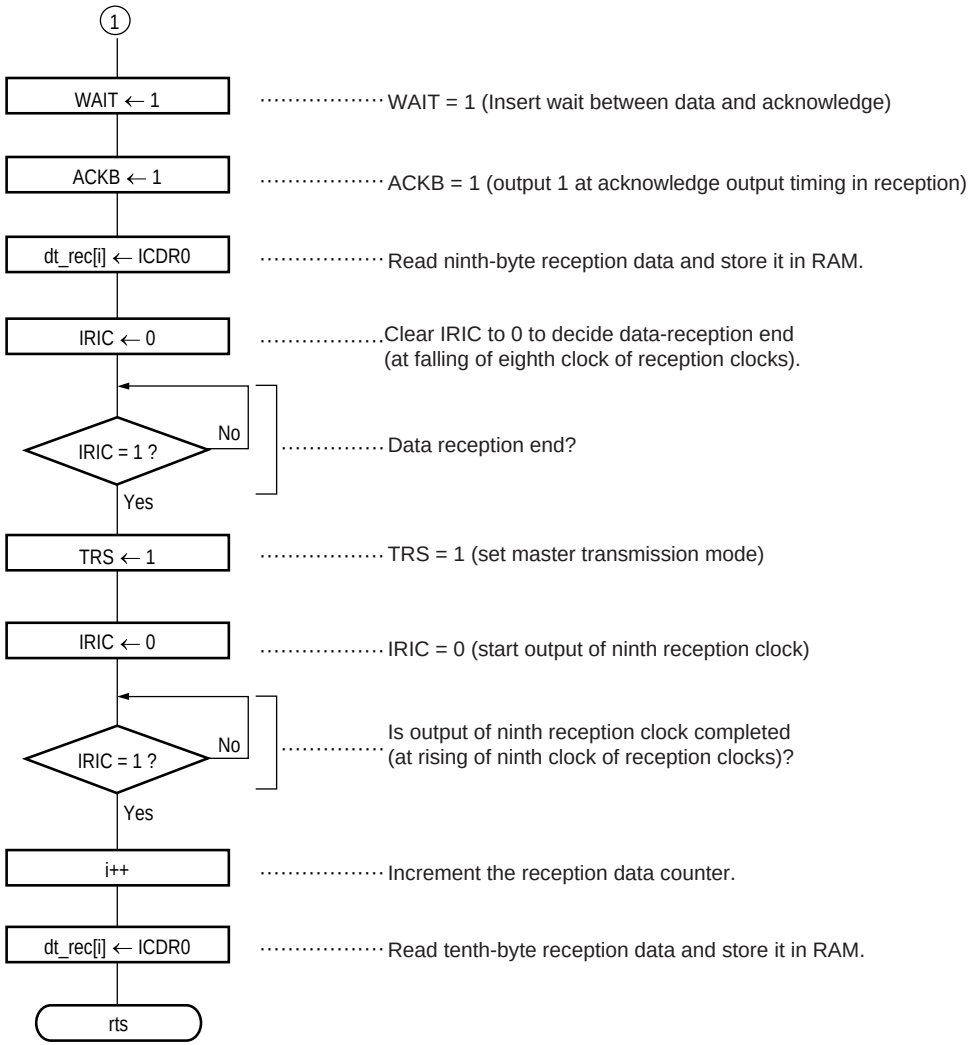


(8) EEPROM memory address transmission subroutine



(9) Data reception subroutine




1
WAIT ← 1
WAIT = 1 (Insert wait between data and acknowledge)
ACKB ← 1
ACKB = 1 (output 1 at acknowledge output timing in reception)
dt_rec[i] ← ICDR0
Read ninth-byte reception data and store it in RAM.
IRIC ← 0
Clear IRIC to 0 to decide data-reception end
(at falling of eighth clock of reception clocks).
IRIC = 1 ?
No
Yes
Data reception end?
TRS ← 1
TRS = 1 (set master transmission mode)
IRIC ← 0
IRIC = 0 (start output of ninth reception clock)
IRIC = 1 ?
No
Yes
Is output of ninth reception clock completed
(at rising of ninth clock of reception clocks)?
i++
Increment the reception data counter.
dt_rec[i] ← ICDR0
Read tenth-byte reception data and store it in RAM.
rts

4.3.5 Program List

```
/******************************************************
*  H8S/2138 IIC bus application note                  *
*    2.Single master receive from EEPROM              *
*                       File name   :  SMRxd.c        *
*                       Fai         :  20MHz          *
*                       Mode        :  3              *
******************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/******************************************************
*  Prototype                                          *
******************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* RAM & IIC0 initialize */
void mst_rec(void);                                 /* Matser receive from EEPROM */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_slvadr_a1(void);                           /* Slave address + R data transmit */
void trs_memadr(void);                              /* EEPROM memory address data transmit */
void rec_data(void);                                /* 10-byte data receive */

/******************************************************
*  RAM allocation                                     *
******************************************************/
#pragma section ramarea
unsigned char dt_rec[10];                           /* Receive data store area */

/******************************************************
*  main : Main routine                                *
******************************************************/
#pragma section
void main(void)
```

```
#pragma asm
mov.l   #h'f000,sp                          /* Stack pointer initialize */
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                     /* MCU mode set */

     SYSCR.BYTE = 0x09;                     /* Interrupt control mode set */
     initialize();                          /* Initialize */
     set_imask_ccr(0);                      /* Interrupt enable */
     mst_rec();                             /* Master receive from EPROM */
     while(1);                              /* End */
}

/*****************************************************
*  initialize : RAM & IIC0 Initialize                *
*****************************************************/
void initialize(void)
{
     unsigned char i=0;

     for(i=0; i<10; i++)                    /* Receive data store area initialize */
     {
          dt_rec[i]  = 0x00;
     }
                                            /* IIC0 module initialize */
     STCR.BYTE = 0x00;                      /* FLSHE = 0 */
     MSTPCR.BYTE.L = 0x7f;                  /* SCI0 module stop mode reset */
     SCI0.SMR.BYTE = 0x00;                  /* SCL0 pin function set */
     SCI0.SCR.BYTE = 0x00;
     MSTPCR.BYTE.L = 0xef;                  /* IIC0 module stop mode reset */
     STCR.BYTE = 0x10;                      /* IICE = 1 */
     DDCSWR.BYTE = 0x0f;                    /* IIC bus format initialize */
     IIC0.ICCR.BYTE = 0x01;                 /* ICE = 0 */
     IIC0.SAR.BYTE = 0x00;                  /* FS = 0 */
     IIC0.SARX.BYTE = 0x01;                 /* FSX = 1 */
     IIC0.ICCR.BYTE = 0x81;                 /* ICE = 1 */
```

```
    IIC0.ICSR.BYTE = 0x00;                          /* ACKB = 0 */
    STCR.BYTE = 0x30;                               /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                          /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                          /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  mst_rec : Master receive from EEPROM              *
*****************************************************/
void mst_rec(void)
{
    while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
    IIC0.ICCR.BIT.MST = 1;                          /* Mster transmit mode set */
    IIC0.ICCR.BIT.TRS = 1;                          /* MST = 1, TRS = 1 */
    set_start();                                    /* Start condition set */
    trs_slvadr_a0();                                /* EEPROM slave address + W data transmit */

    if(IIC0.ICSR.BIT.ACKB == 0)                     /* ACKB = 0 ? */
    {
        trs_memadr();                               /* EEPROM memory address data transmit */

        if(IIC0.ICSR.BIT.ACKB == 0)                 /* ACKB = 0 ? */
        {
            set_start();                            /* Re-start condition set */
            trs_slvadr_a1();                        /* EEPROM slave address + R data transmit */

            if(IIC0.ICSR.BIT.ACKB == 0)             /* ACKB = 0 ? */
            {
                rec_data();                         /* Data recieve */
            }
        }
    }
    set_stop();
}
/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
```

```
void set_start(void)
{
     IIC0.ICCR.BIT.IRIC = 0;                          /* IRIC = 0 */
     IIC0.ICCR.BYTE = 0xbc;                           /* Start condition set (BBSY=1,SCP=0) */
     while(IIC0.ICCR.BIT.IRIC == 0);                  /* Start condition set (IRIC=1) ? */
}

/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
     IIC0.ICCR.BYTE = 0xb8;                           /* Stop condition set (BBSY=0,SCP=0) */
     while(IIC0.ICCR.BIT.BBSY == 1);                  /* Bus empty (BBSY=0) ? */
}

/*****************************************************
*  trs_slvadr_a0 : Slave address + W data transmit   *
*****************************************************/
void trs_slvadr_a0(void)
{
     IIC0.ICDR = 0xa0;                                /* Slave address + W data(H'A0) write */
     IIC0.ICCR.BIT.IRIC = 0;                          /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                  /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_slvadr_a1 : Slave address + R data transmit   *
*****************************************************/
void trs_slvadr_a1(void)
{
     IIC0.ICDR = 0xa1;                                /* Slave address + R data(H'A1) write */
     IIC0.ICCR.BIT.IRIC = 0;                          /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                  /* Transmit end (IRIC=1) ? */
}
```

```
/*****************************************************
*  trs_memadr : EEPROM memory address data transmit  *
*****************************************************/
void trs_memadr(void)
{
     IIC0.ICDR = 0x00;                              /* EEPROM memory address data(H'00) write */
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  rec_data : 10-byte data receive                  *
*****************************************************/
void rec_data(void)
{
     unsigned char i=0;                             /* Receive data counter initialize */

     IIC0.ICCR.BIT.TRS = 0;                         /* Master transmit mode set (MST=1,TRS=0) */
     IIC0.ICSR.BIT.ACKB = 0;                        /* ACKB = 0 */
     IIC0.ICMR.BIT.WAIT = 0;                        /* WAIT = 0 */

     dt_rec[i] = IIC0.ICDR;                         /* Dummy read */
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                /* receive end (IRIC=1) ? */

     for(i=0; i<8; i++)                             /* 1st to 8th data receive */
     {
          dt_rec[i] = IIC0.ICDR;                    /* Receive data read */
          IIC0.ICCR.BIT.IRIC = 0;                   /* IRIC = 0 */
          while(IIC0.ICCR.BIT.IRIC == 0);           /* Receive end ? */
     }

     IIC0.ICMR.BIT.WAIT = 1;                        /* WAIT = 1 */
     IIC0.ICSR.BIT.ACKB = 1;                        /* ACKB = 1 */
     dt_rec[i] = IIC0.ICDR;                         /* 9th receive data read */
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                /* Receive end (IRIC=1) ? */
```

```
    IIC0.ICCR.BIT.TRS = 1;                          /* Master transmit mode set (MST=1,TRS=1) */
    IIC0.ICCR.BIT.IRIC = 0;                         /* 9th clock transmit (IRIC=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* 9th clock transmit end (IRIC=1) ? */

    dt_rec[++i] = IIC0.ICDR;                        /* 10th (last) receive data read */

    IIC0.ICSR.BIT.ACKB = 0;                         /* ACKB = 0 */
    IIC0.ICMR.BIT.WAIT = 0;                         /* WAIT = 0 */
}
```

4.4 One-Byte Data Transmission by Single-Master Transmission

4.4.1 Specifications

- The I²C bus interface of channel 0 in H8S/2138 is used to write 1-byte data to EEPROM (HN58X2408).
- The slave address of EEPROM to be connected is “1010000”, and data is written to H'00 of EEPROM memory addresses.
- Devices connected to the I²C bus of this system consist of a master device (H8S/2138) and a slave device (EEPROM) (single master configuration).
- The frequency of a transmission clock is 100 kHz.
- Figure 4.11 shows the connection example of H8S/2138 and EEPROM.

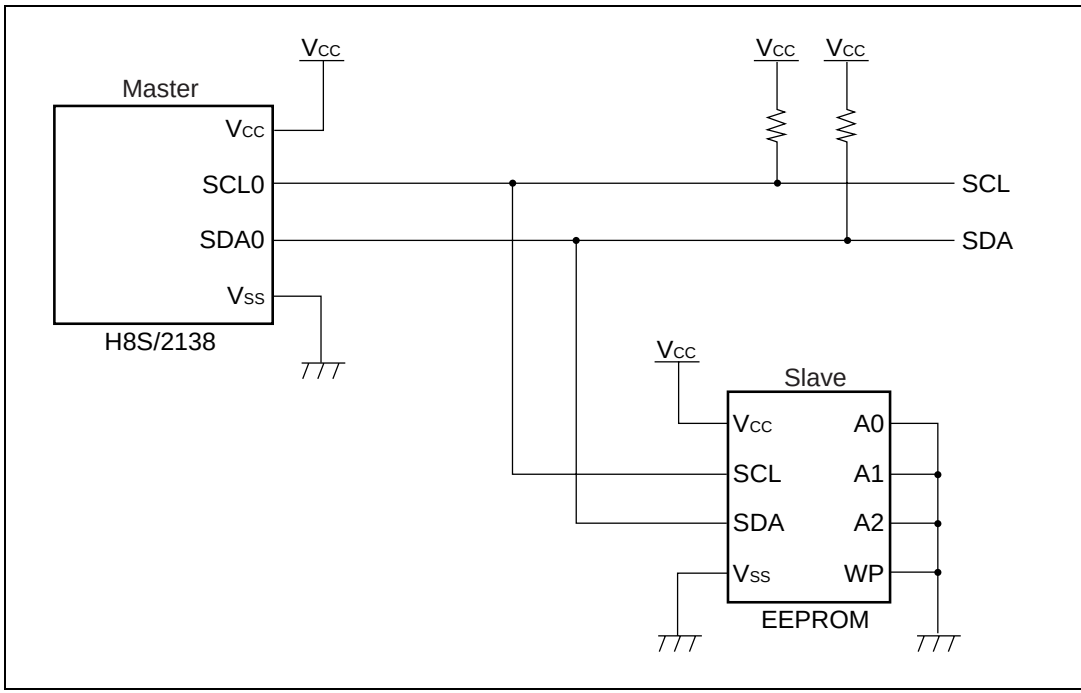
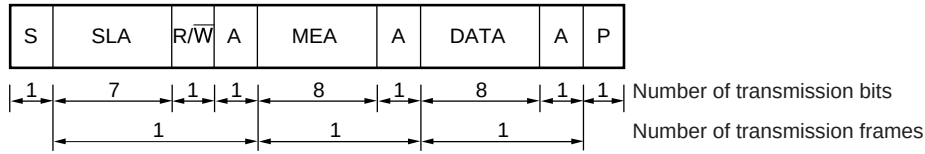


Figure 4.11 Connection Example of H8S/2138 and EEPROM

- Figure 4.12 shows the I²C bus format used in this task example.

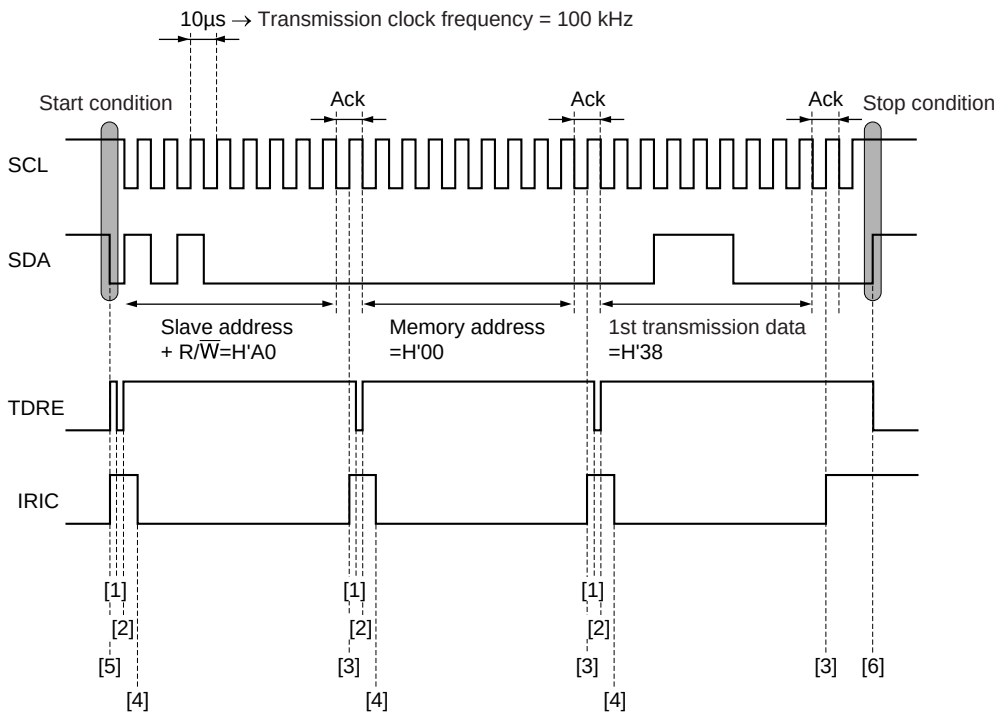


Legend:
- S : Start condition
- SLA : EEPROM slave address
- R/$\overline{W}$: Transmission/reception direction
- A : Acknowledge
- MEA : EEPROM memory address
- DATA : Transmission data
- P : Stop condition

Figure 4.12 Transmission Format Used in this Task Example

4.4.2 Operation Descriptions

Figure 4.13 shows an operation principle.



	Software Processing	Hardware Processing
[1]	Writes transmission data to ICDR0	TDRE = 0 (writes data to ICDRT when TRS = 1)
[2]	No processing	TDRE = 1 (transmits data to ICDRS from ICDRT)
[3]	No processing	IRIC = 1 (ends data transmission (at rising of 9th reception clock))
[4]	Clears IRIC to 0 to judge transmission end	No processing
[5]	Issues start conditions (BBSY = 1, SCP = 0)	IRIC = 1, TDRE = 1 (detects start conditions from the bus line state)
[6]	Issues stop conditions (BBSY = 0, SCP = 0)	TDRE = 0 (detects stop conditions from the bus line state)

Figure 4.13 One-byte Data Transmission Operation Principle by Single Master Transmission

4.4.3 Software Descriptions

(1) Descriptions of modules

Table 4.8 shows the descriptions of modules in this task example.

Table 4.8 Descriptions of Modules

Module Name	Label Name	Function
Main routine	main	Sets stack pointer, sets MCU mode, and enables an interrupt.
Initial setting	Intialize	Initially sets IIC0.
Single master transmission	mst_trs	Transmits 1-byte data to EEPROM by single master transmission.
Start condition issue	set_start	Issues start conditions.
Stop condition issue	set_stop	Issues stop conditions.
Slave address + W transmission	trs_slvadr_a0	Transmits slave address + W data (H'A0) of EEPROM.
EEPROM memory address transmission	trs_memadr	Transmits memory address data (H'00) of EEPROM.

(2) Descriptions of internal registers

Table 4.9 shows the descriptions of internal registers to be used in this task example.

Table 4.9 Descriptions of Registers

Register		Function	Address	Set Value
ICDR0		Stores transmission data.	H'FFDE	—
SAR0	FS	Sets transmission format by using bit FSX of SAR0 and bit SW of DDCSWR.	H'FFDF bit0	0
SARX0	FSX	Sets transmission format by using bit FS of SAR0 and bit SW of DDCSWR.	H'FFDE bit0	1
ICMR0	MLS	Sets data transmission by MSB-first.	H'FFDF bit7	0
	WAIT	Sets continuous transmission of data and acknowledge.	H'FFDF bit6	0
	CKS2 to CKS0	Set transmission clock frequency to 100 kHz by using bit IICX0 of STCR.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set 9 bits/frame to the number of bits of data to be transmitted next in I^2C bus format.	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Selects access control of registers ICMR0, ICDR0/SAR, and SARX, and I^2C bus interface operation (port function for pin SCL0/SDA0)/non-operation (bus drive state for pin SCL/SDA).	H'FFD8 bit7	0/1
	IEIC	Inhibits I^2C bus interface interrupt requests.	H'FFD8 bit6	0
	MST	Uses the I^2C bus interface in master mode.	H'FFD8 bit5	1
ICCR0	TRS	Sets transmission mode of the I^2C bus interface.	H'FFD8 bit4	1/0
	ACKE	Halts continuous transmission when the acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms that the I^2C bus is occupied or released, and issues start and stop conditions by using bit SCP.	H'FFD8 bit2	0/1
	IRIC	Detects start conditions, decides data transmission end, and detects acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Issues start and stop conditions by using bit BBSY.	H'FFD8 bit0	0
ICSR0	ACKB	Stores acknowledge transmitted from EEPROM.	H'FFD9 bit0	—
STCR	IICX0	Sets transmission clock frequency to 100 kHz by using CKS2 to CKS0 of ICMR0.	H'FFC3 bit5	1
	IICE	Enables CPU access to the data register and control register of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets a non-select state to the control register of flash memory.	H'FFC3 bit3	0

Table 4.9 Descriptions of Registers (cont)

Register		Function	Address	Set Value
DDCSWR	SWE	Inhibits automatic switching from format-less to I²C bus format for IIC channel 0.	H'FEE6 bit7	0
	SW	Uses IIC channel 0 in the I²C bus format.	H'FEE6 bit6	0
	IE	Inhibits an interrupt in automatic format switching.	H'FEE6 bit5	0
	CLR3 to CLR0	Control initialization of the internal state of IIC0.	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels module stop mode of SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels module stop mode of IIC channel 0.	H'FF87 bit4	0
SCR0	CKE1,0	Sets the P52/SCK0/SCL0 pin to an I/O port.	H'FFDA bit1,0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets SCI0 operating mode to asynchronous mode.	H'FFD8 bit7	0
SYSCR	INTM1,0	Set interrupt control mode of the interrupt controller to control by bit 1.	H'FFC4 bit5,4	INTM1=0 INTM0=0
MDCR	MDS1,0	Set MCU operating mode to mode 3 by latching the input level of pins MD1 and MD0.	H'FFC5 bit1,0	MDS1=1 MDS0=1

(3) Descriptions of variables

Table 4.10 shows the descriptions of variables in this task example.

Table 4.10 Descriptions of Variables

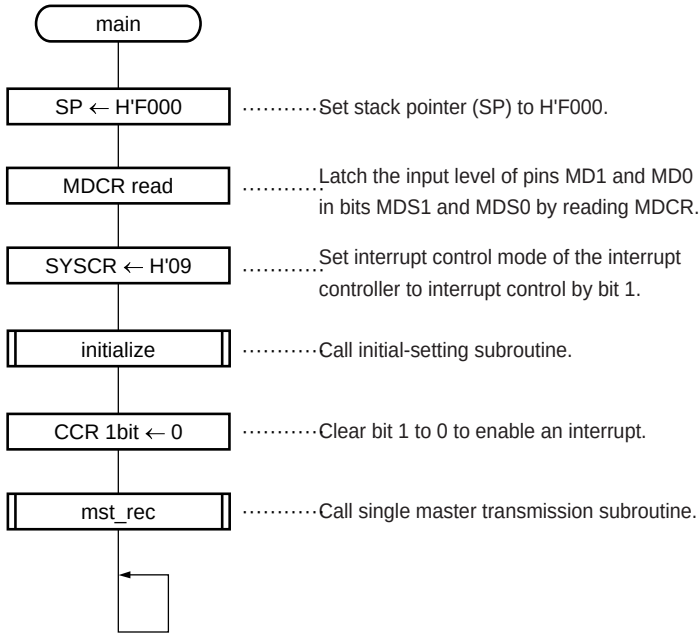
Variable	Function	Data Length	Initial Value	Used Module Name
dt_trs	One-byte transmission data	1 byte	H'38	mst_trs
dummy	MDCR read value	1 byte	—	main

(4) Used RAM descriptions

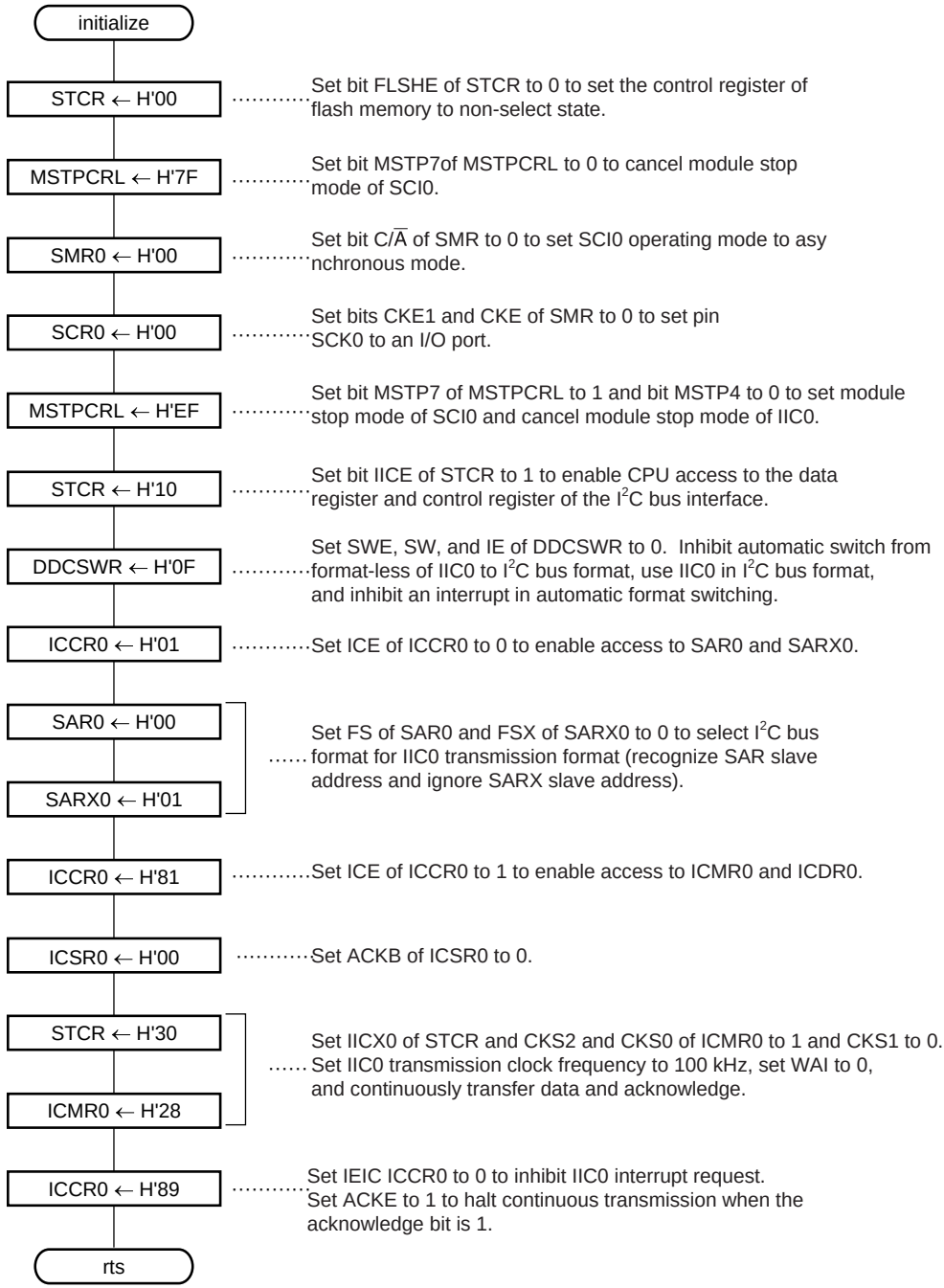
RAM for other than variables is not used in this task example.

4.4.4 Flowchart

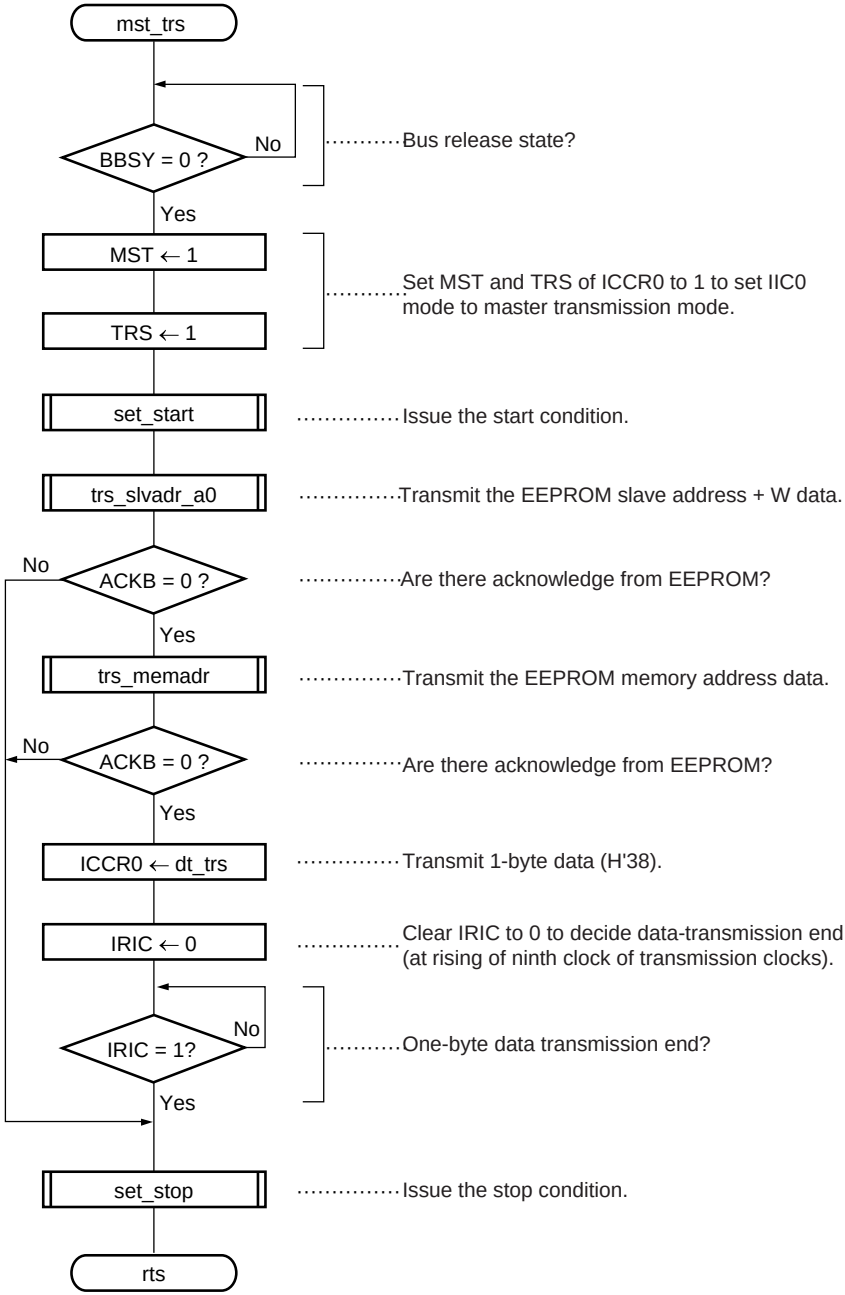
(1) Main routine



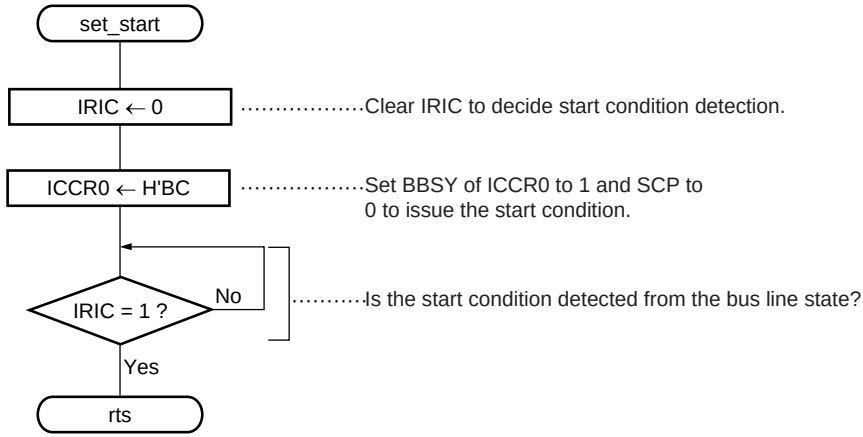
(2) Initial-setting subroutine



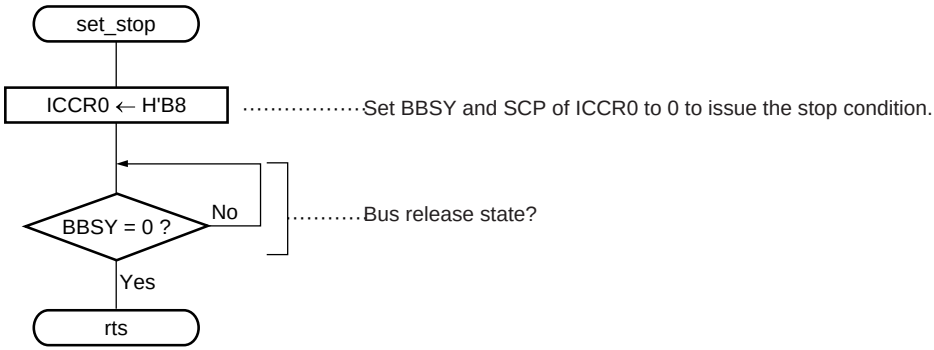
(3) Single master transmission subroutine



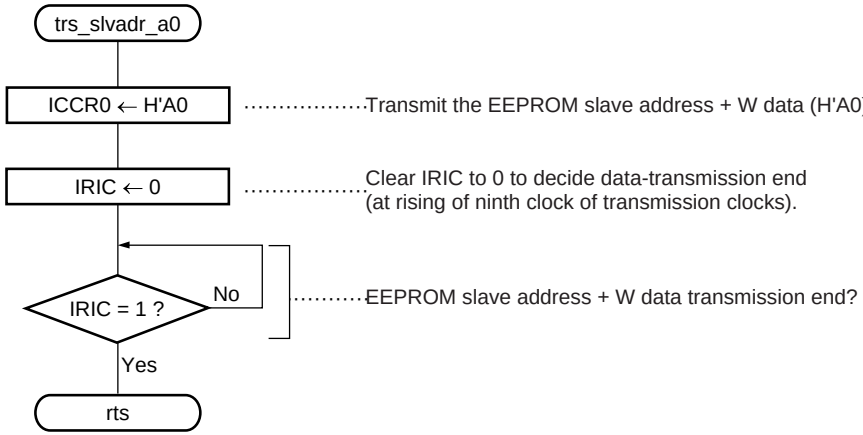
(4) Subroutine that sets the start condition



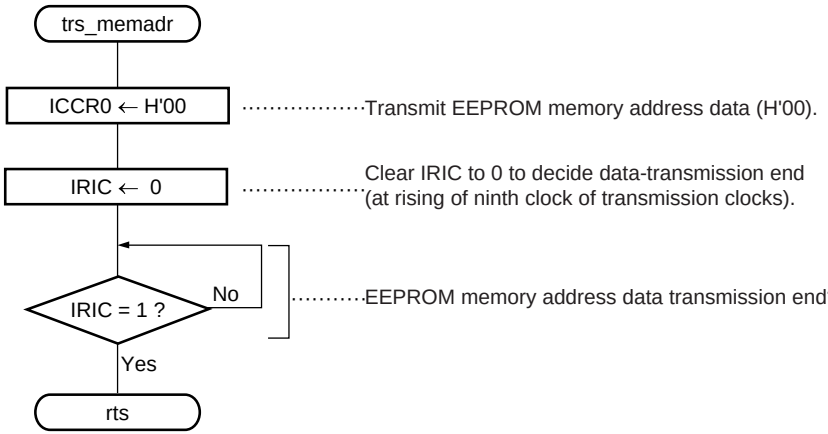
(5) Subroutine that sets the stop condition



(6) Slave address + W transmission subroutine



(7) EEPROM memory address transmission subroutine



4.4.5 Program List

```
/*****************************************************
*  H8S/2138 IIC bus application note                 *
*    3.Single master transmit 1byte data to EEPROM   *
*                       File name   :  BYTxd.c        *
*                       Fai         :  20MHz          *
*                       Mode        :  3              *
*****************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*****************************************************
*  Prototype                                         *
*****************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* IIC0 initialize */
void mst_trs(void);                                 /* Master transmit to EEPROM */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_memadr(void);                              /* EEPROM memory address data transmit */

unsigned char dt_trs = 0x38;                        /* Transmit data (1byte) */

/*****************************************************
*  main : Main routine                               *
*****************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                         ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                             /* MCU mode set */
```

```
    SYSCR.BYTE = 0x09;                          /* Interrupt control mode set */
    initialize();                               /* Initialize */
    set_imask_ccr(0);                           /* Interrupt enable */
    mst_trs();                                  /* Master transmit to EPROM */
    while(1);                                   /* End */
}

/*****************************************************
*  initialize : IIC0 initialize                      *
*****************************************************/
void initialize(void)
{
    STCR.BYTE = 0x00;                           /* FLSHE = 0 */
    MSTPCR.BYTE.L = 0x7f;                       /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                       /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    MSTPCR.BYTE.L = 0xef;                       /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                           /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                         /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                      /* ICE = 0 */
    IIC0.SAR.BYTE = 0x00;                       /* FS = 0 */
    IIC0.SARX.BYTE = 0x01;                      /* FSX = 1 */
    IIC0.ICCR.BYTE = 0x81;                      /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                      /* ACKB = 0 */
    STCR.BYTE = 0x30;                           /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                      /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                      /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  mst_trs : Master transmit to EEPROM               *
*****************************************************/
void mst_trs(void)
{
    while(IIC0.ICCR.BIT.BBSY == 1);             /* Bus empty (BBSY=0) ? */
    IIC0.ICCR.BIT.MST = 1;                      /* Master transmit mode set */
```

```
    IIC0.ICCR.BIT.TRS = 1;                          /* MST = 1, TRS = 1 */
    set_start();                                    /* Start condition set */
    trs_slvadr_a0();                                /* Slave address + W data transmit */

    if (IIC0.ICSR.BIT.ACKB == 0)
    {
        trs_memadr();                               /* EEPROM memory address data transmit */

        if (IIC0.ICSR.BIT.ACKB == 0)
        {
            IIC0.ICDR = dt_trs;                     /* 1 byte data transmit */
            IIC0.ICCR.BIT.IRIC = 0;                 /* IRIC = 0 */
            while(IIC0.ICCR.BIT.IRIC == 0);         /* transmit end (IRIC=1) ? */
        }
    }
    set_stop();                                     /* Stop condition set */
}
/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
void set_start(void)
{
    IIC0.ICCR.BIT.IRIC = 0;       /* IRIC = 0 */
    IIC0.ICCR.BYTE = 0xbc;                          /* Start condition set (BBSY=1,SCP=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Start condition set (IRIC=1) ? */
}

/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
    IIC0.ICCR.BYTE = 0xb8;                          /* Stop condition set (BBSY=0,SCP=0) */
    while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
}
```

```
/*****************************************************
*  trs_slvadr_a0 : Slave address + W data transmit   *
*****************************************************/
void trs_slvadr_a0(void)
{
     IIC0.ICDR = 0xa0;                              /* Slave address + W data(H'A0) write */
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_memadr : EEPROM memory address data transmit  *
*****************************************************/
void trs_memadr(void)
{
     IIC0.ICDR = 0x00;                              /* EEPROM memory address data(H'00) write */
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                /* Transmit end (IRIC=1) ? */
}
```

4.5 One-Byte Data Reception by Single-Master Reception

4.5.1 Specifications

- One-byte data is read from EEPROM (HN58X2408) using channel 0 of the I²C bus interface in the H8S/2138.
- The slave address of EEPROM to be connected is 1010000, and data in address H'00 of the EEPROM memory address is read.
- Data to be read is stored at address H'E100 in RAM.
- The device connected to the I²C bus in this system is a single–master configuration—one master device (H8S/2138) and one slave device (EEPROM).
- The transfer clock frequency is 100 kHz.
- Figure 4.14 shows an example of the H8S/2138 and EEPROM connection.

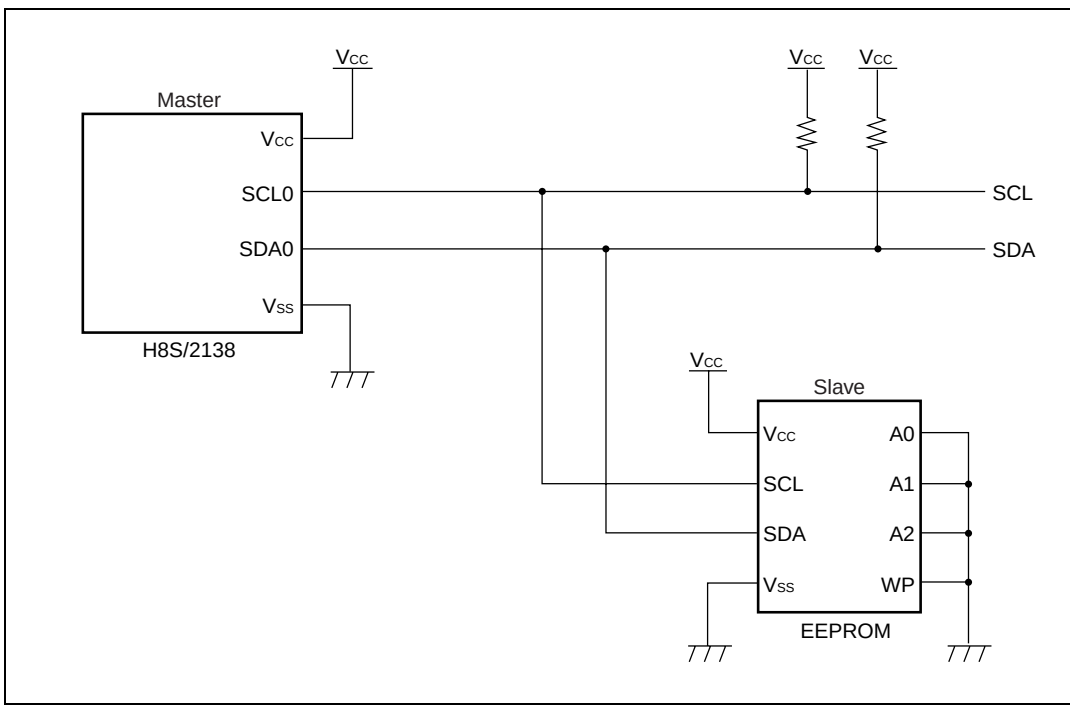


Figure 4.14 Example of H8S/2138 and EEPROM Connection

- Figure 4.15 shows the I²C bus format used in this task example.

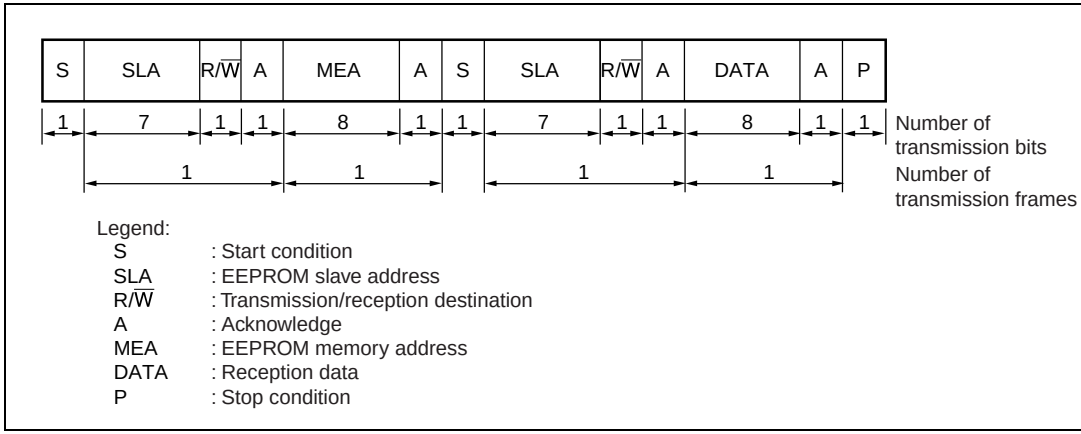
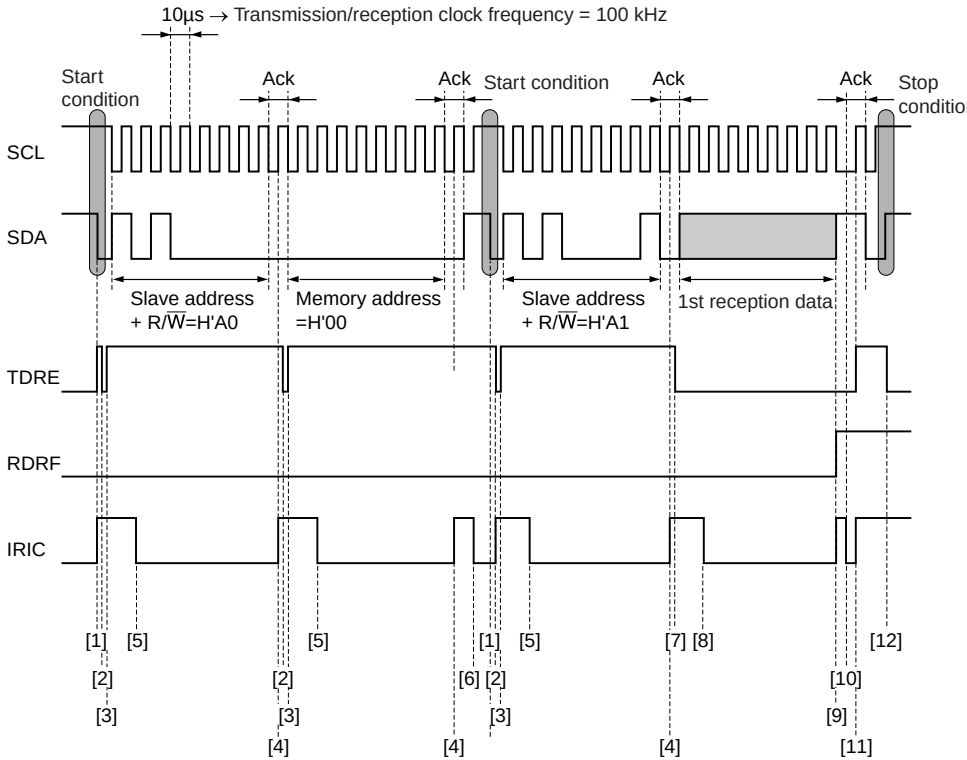


Figure 4.15 Transfer Format Used in This Task Example

4.5.2 Operation Description

Figure 4.16 shows the operation principle.



	Software processing	Hardware processing
[1]	Issues start conditions (BBSY = 1, SCP = 0)	IRIC = 1, TDRE = 1 (detects start condition from the bus line state)
[2]	Writes transfer data to ICDR0	TDRE = 0 (writes data to ICDRT while TRS = 1)
[3]	No processing	TDRE = 1 (transmits data from ICDRT to ICDRS)
[4]	No processing	IRIC = 1 (ends data transmission (at rising of 9th transmission clock))
[5]	Clears IRIC to 0 to judge transfer end	No processing
[6]	Clears IRIC to 0 to judge detection of start conditions	No processing
[7]	Sets in master reception mode (MST = 1, TRS = 0)	TDRE = 0 (when TRS = 0)
[8]	Clears IRIC to 0 to judge reception end	No processing
[9]	No processing	IRIC = 1, RDRF = 1 (WAIT = 1) (ends data reception (at falling of 8th reception clock))
[10]	Clears IRIC to 0 to judge output end of 9th reception clock	Starts output at 9th reception clock
[11]	Sets in master transfer mode (MST = 1, TRS = 1)	IRIC = 1 (at rising of 9th reception clock) TDRE = 1 (after detecting start conditions, when switching from TRS = 0 to TRS = 1)
[12]	Issues stop conditions (BBSY = 0, SCP = 0)	TDRE = 0 (after issuing stop condition, detects stop conditions from the bus line state)

Figure 4.16 Principle of Reception Operation in One-Byte Data by Single-Master Reception

4.5.3 Software Description

(1) Module Description

Table 4.11 describes the module in this task example.

Table 4.11 Module Description

Module Name	Label Name	Function
Main routine	main	Sets stack pointer and MCU mode, and enables interrupts.
Initial setting	initialize	Initial settings of using RAM area and IIC0.
Single-master reception	mst_rec	Receives one-byte data from EEPROM by single-master reception.
Start condition issuance	set_start	Issues start condition.
Stop condition issuance	set_stop	Issues stop condition.
Slave address + W transmission	trs_slvadr_a0	Transmits slave address + W data (H'A0) in EEPROM.
Slave address + R transmission	trs_slvadr_a1	Transmits slave address + R data (H'A1) in EEPROM.
EEPROM memory address transmission	trs_memadr	Transmits memory address data (H'00) in EEPROM.
Data reception	rec_data	Receives one-byte data.

(2) On-Chip Register Description

Table 4.12 describes the on-chip register in this task example.

Table 4.12 On-Chip Register Description

Register		Function	Address	Setting Value
ICDR0		Stores transmission/reception data.	H'FFDE	—
SAR0	FS	Sets transfer format with the FSX bit in SARX0 and the SW bit in DDCSWR.	H'FFDF bit0	0
SARX0	FSX	Sets transfer format with the FS bit in SAR0 and the SW bit in DDCSWR.	H'FFDE bit0	1
ICMR0	MLS	Sets data transfer by MSB first.	H'FFDF bit7	0
	WAIT	Sets whether wait is input or not between data and acknowledge bit.	H'FFDF bit6	0/1
	CKS2 to CKS0	Set transfer clock frequency to 100 kHz in conjunction with the IICX0 bit in STCR.	H'FFDF Bit5 to Bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set number of data bits to be transferred next to 9 bits/frame by the I^2C bus format.	H'FFDF Bit2 to Bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Controls access to ICMR0, ICDR0/SAR, SARX, and selects the I^2C bus interface to operate (SCL0 and SDA0 pins function as port) or not to operate (SCL/SDA pins are in the bus drive state).	H'FFD8 bit7	0/1
	IEIC	Disables an interrupt request of the I^2C bus interface.	H'FFD8 bit6	0
	MST	Uses the I^2C bus interface in master mode.	H'FFD8 bit5	1
	TRS	Sets transmission/reception mode in the I^2C bus interface.	H'FFD8 bit4	0/1
	ACKE	Suspends continuous transfer when an acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms the I^2C bus is occupied or released, and issues start or stop condition in conjunction with the SCP bit.	H'FFD8 bit2	0/1
	IRIC	Detects start condition, judges end of data transfer, and detects an acknowledge bit = 1.	H'FFD8 bit1	0/1
	SCP	Issues start or stop condition in conjunction with the BBSY bit.	H'FFD8 bit0	0
ICSR0	ACKB	Stores an acknowledge bit received from EEPROM in transmitting. Sets an acknowledge bit to be transferred to EEPROM in reception.	H'FFD9 bit0	-

Table 4.12 On-chip Register Description (cont)

Register		Function	Address	Setting Value
STCR	IICX0	Sets the transfer clock frequency to 100 kHz in conjunction with CKS2 to CKS0 bits in ICMR0.	H'FFC3 bit5	1
	IICE	Enables access to CPU by the data and control registers of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets the control register in flash memory to be in non-selectable state.	H'FFC3 bit3	0
DDCSWR	SWE	Disables automatic switching from formatless of channel 0 in IIC to the I^2C bus format.	H'FEE6 bit7	0
	SW	Uses channel 0 in IIC in the I^2C bus format.	H'FEE6 bit6	0
	IE	Disables interrupts when format is switched automatically.	H'FEE6 bit5	0
	CLR3 to CLR0	Control initialization of an internal state in IIC0.	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels module stop mode in channel 0 in SCI.	H'FF87 bit7	0
	MSTP4	Cancels module stop mode in channel 0 in IIC.	H'FF87 bit4	0
SCR0	CKE1,0	Set P52/SCK0/SCL0 pin as I/O port.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets operating mode in SCI0 to synchronous mode.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set interrupt control mode in interrupt controller to be controlled by the 1 bit.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set MCU operating mode to mode 3 by latching input levels of MD1 and MD0 pins.	H'FFC5 bit1, 0	MDS1=1 MDS0=1

(3) Variable Description

Table 4.13 describes the variable in this task example.

Table 4.13 Variable Description

Variable	Function	Data Length	Initial Value	Module in Use
dummy	MDCR read value	1 byte	—	main

(4) Using RAM Description

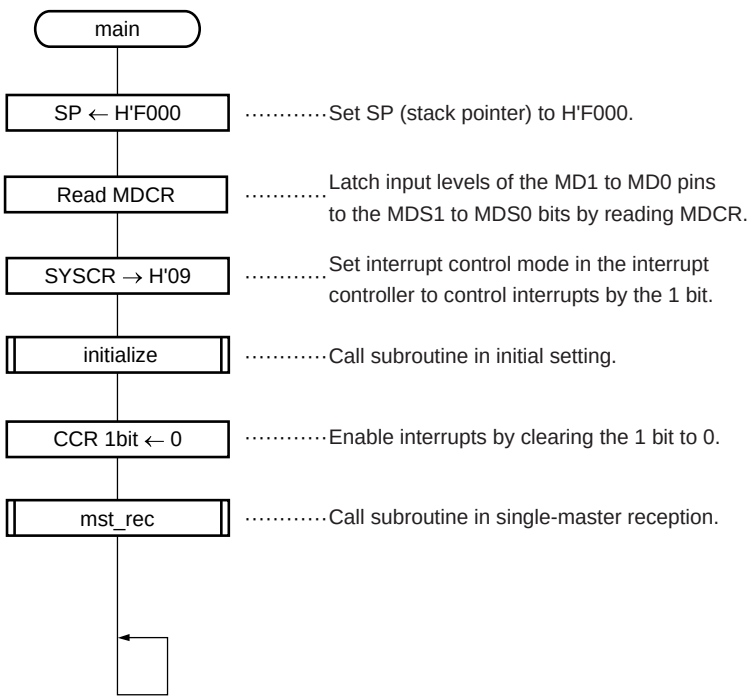
Table 4.14 describes the RAM used in this task example.

Table 4.14 Description of RAM Used

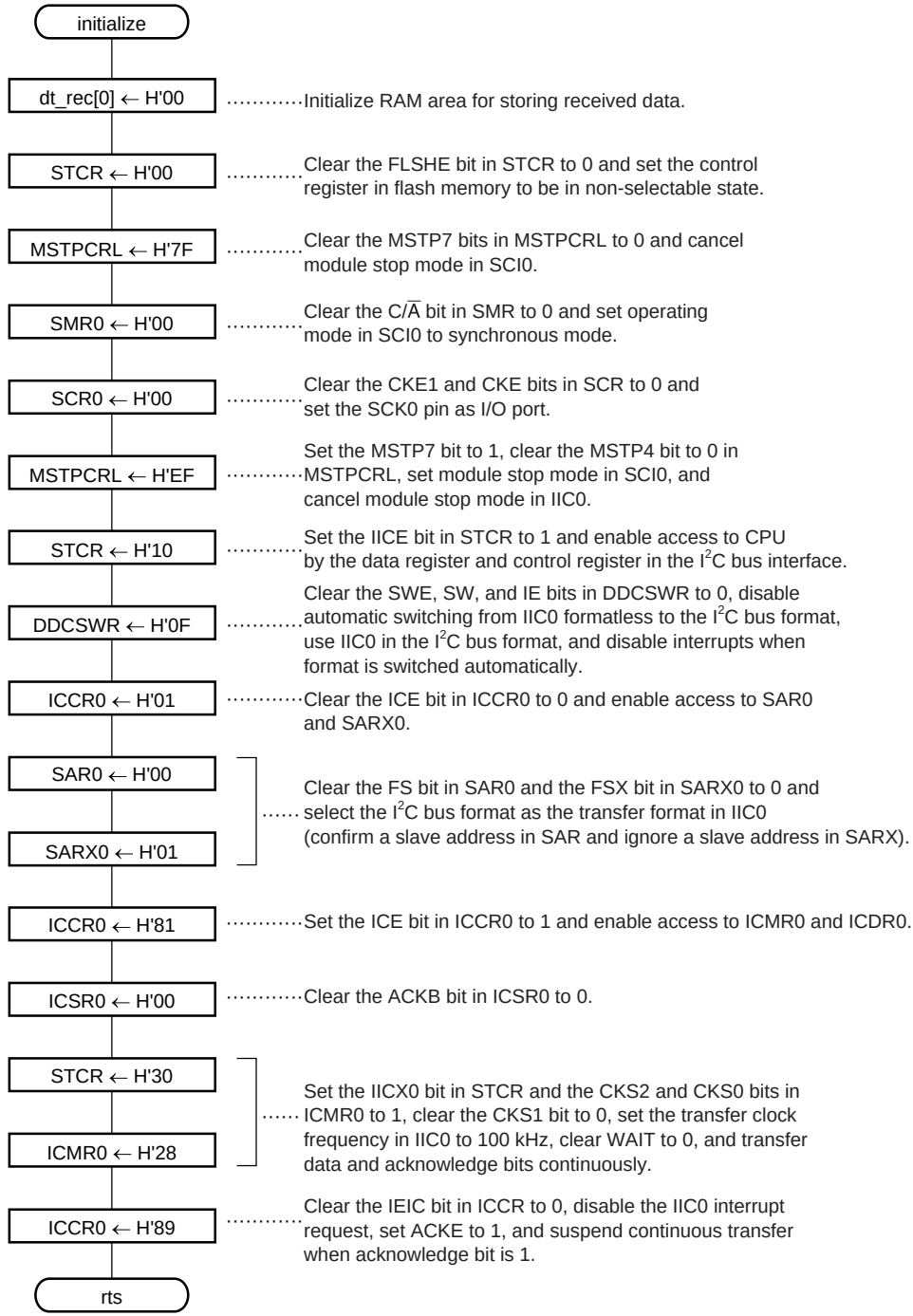
Label	Function	Data Length	Address	Module in Use
dt_rec[0]	Stores received data.	1 byte	H'E100	Initialize rec_data

4.5.4 Flowchart

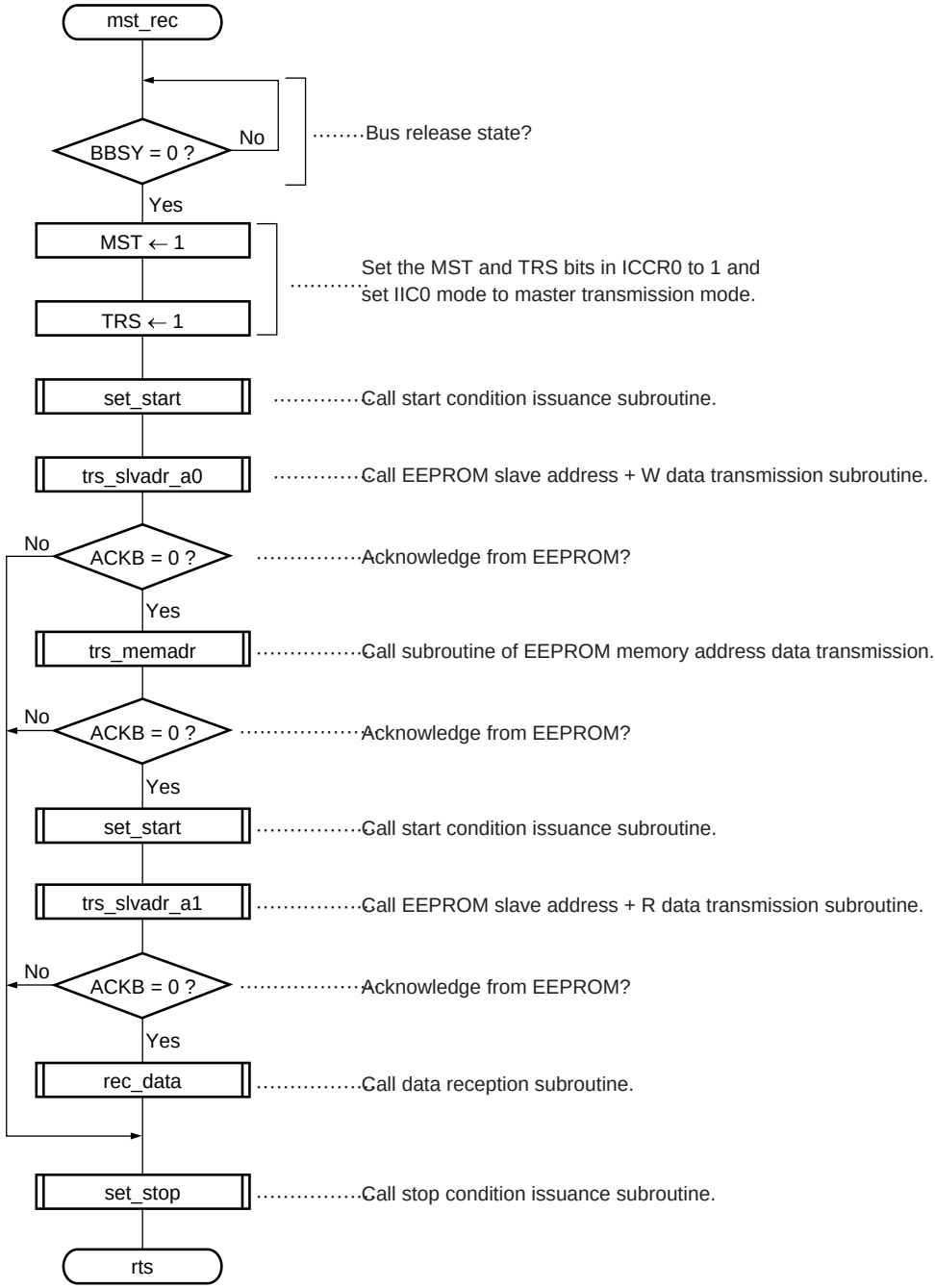
(1) Main Routine



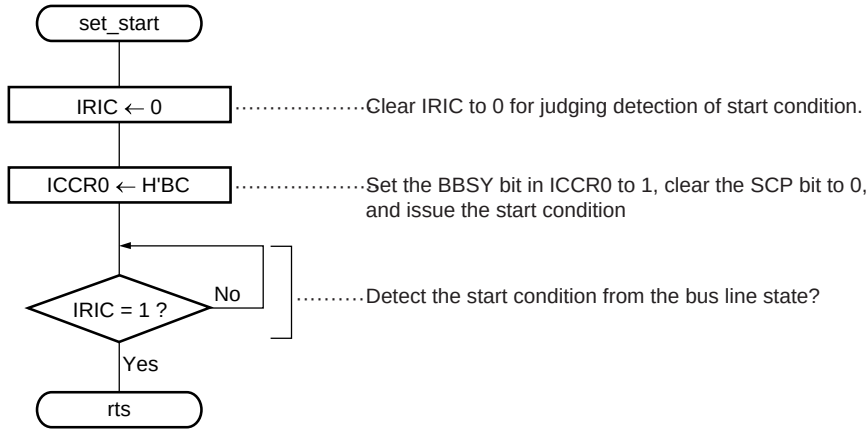
(2) Initial Setting Subroutine



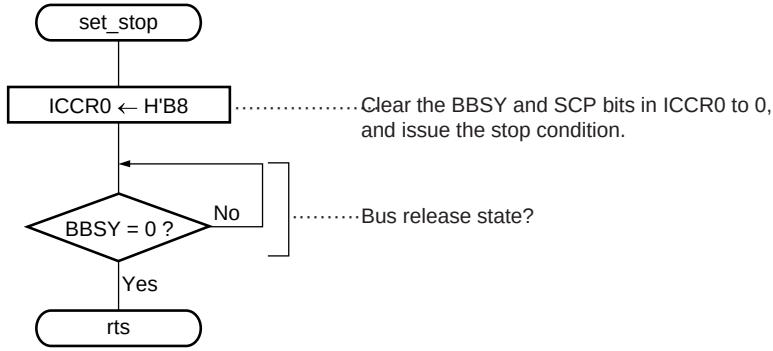
(3) Single-Master Reception Subroutine



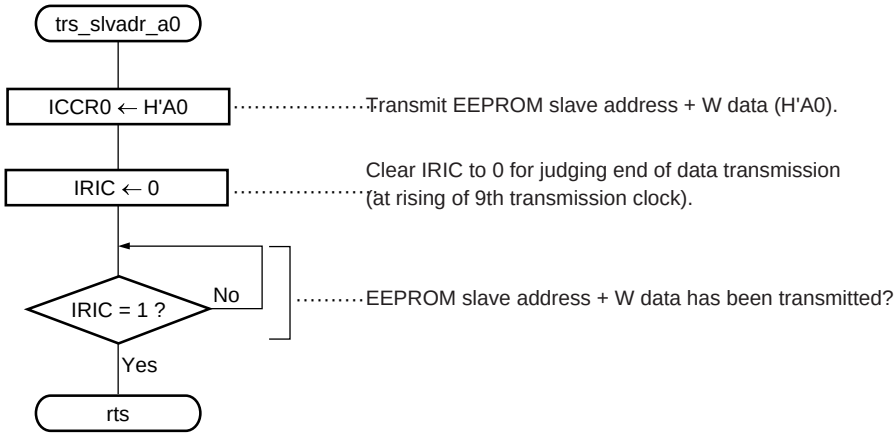
(4) Start Condition Issuance Subroutine



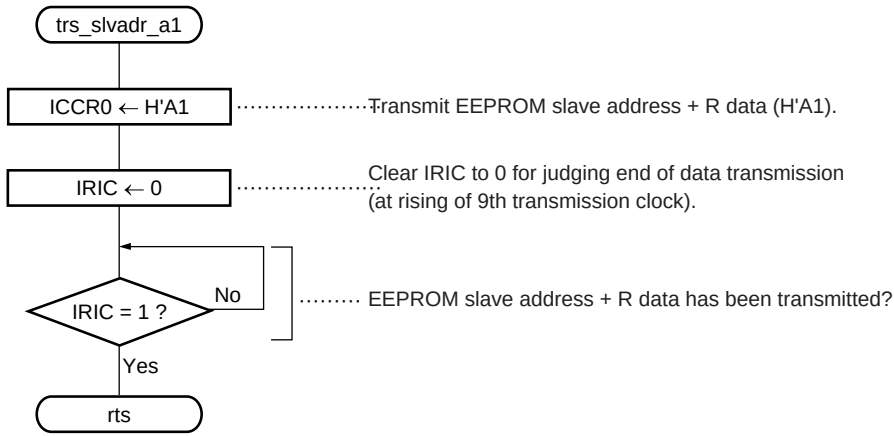
(5) Stop Condition Issuance Subroutine



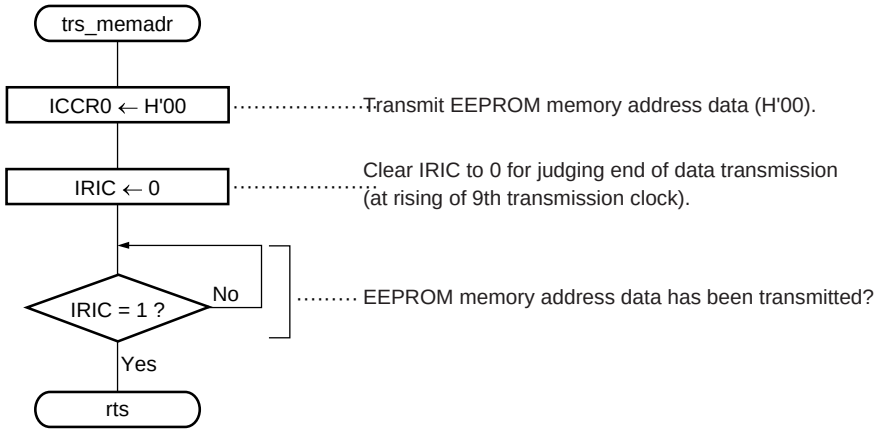
(6) Slave Address + W Transmission Subroutine



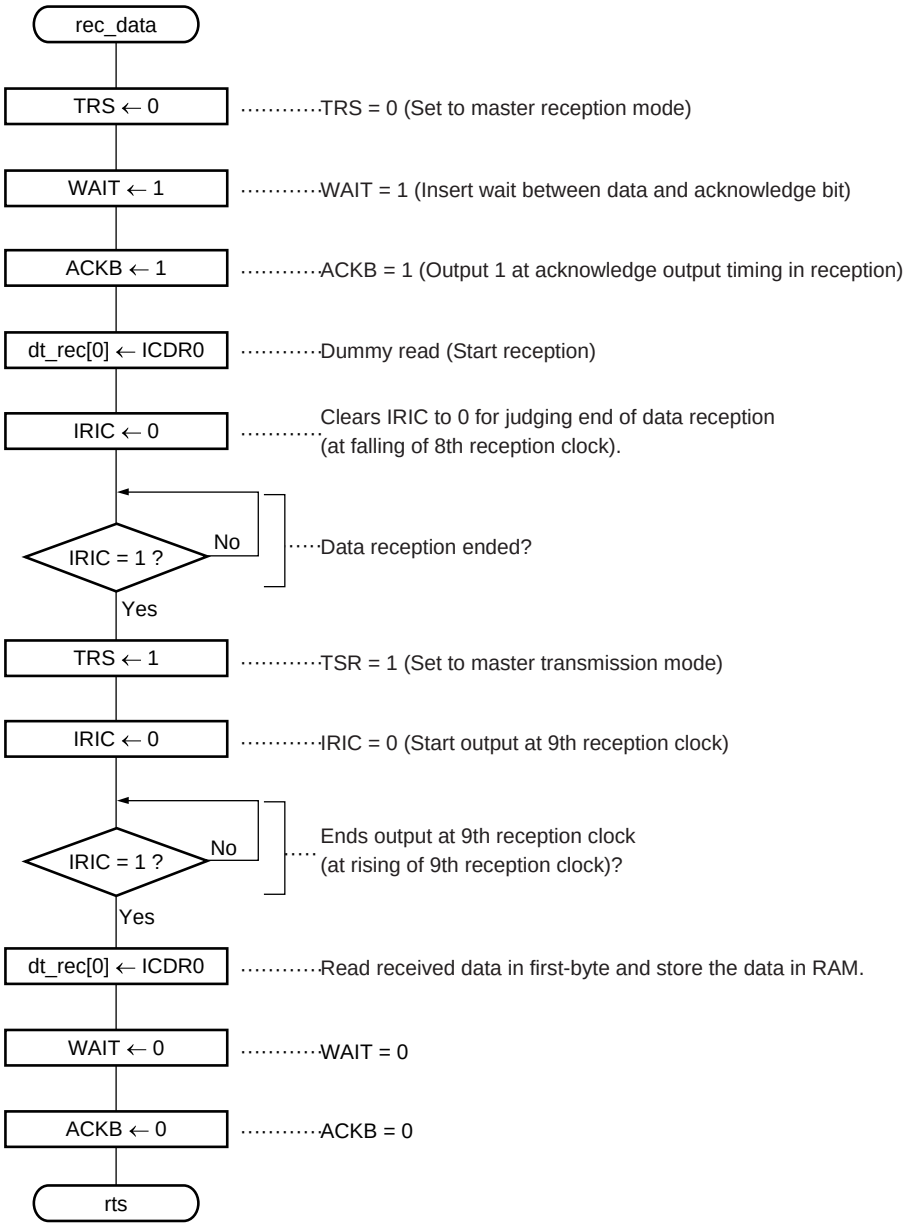
(7) Slave Address + R Transmission Subroutine



(8) Subroutine of EEPROM Memory Address Transmission



(9) Data Reception Subroutine



4.5.5 Program List

```
/*******************************************************
*  H8S/2138 IIC bus application note                   *
*    4.Single master receive 1byte data from EEPROM    *
*                       File name   :   BYRxd.c         *
*                       Fai         :   20MHz           *
*                       Mode        :   3               *
*******************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*******************************************************
*  Prototype                                           *
*******************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* RAM & IIC0 initialize */
void mst_rec(void);                                 /* Master receive from EEPROM */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_slvadr_a1(void);                           /* Slave address + R data transmit */
void trs_memadr(void);                              /* EEPROM memory address data transmit */
void rec_data(void);                                /* 1-byte data receive */

/*******************************************************
*  RAM allocation                                      *
*******************************************************/
#pragma section ramerea
unsigned char dt_rec[1];                            /* Receive data store area */

/*******************************************************
*  main : Main routine                                 *
*******************************************************/
#pragma section
void main(void)
```

```
#pragma asm
        mov.l   #h'f000,sp                      ;Stack pointer initialize
#pragma endasm
{
    unsigned char dummy;
    dummy = MDCR.BYTE;                          /* MCU mode set */

    SYSCR.BYTE = 0x09;                          /* Interrupt control mode set */
    initialize();                               /* Initialize */
    set_imask_ccr(0);                           /* Interrupt enable */
    mst_rec();                                  /* Master receive from EPROM */
    while(1);                                   /* End */
}

/*************************************************
*  initialize : RAM & IIC0 Initialize           *
*************************************************/
void initialize(void)
{
    dt_rec[0] = 0x00;                           /* Receive data store area initialize */

    STCR.BYTE = 0x00;                           /* FLSHE = 0 */
    MSTPCR.BYTE.L = 0x7f;                       /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                       /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    MSTPCR.BYTE.L = 0xef;                       /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                           /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                         /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                      /* ICE = 0 */
    IIC0.SAR.BYTE = 0x00;                       /* FS = 0 */
    IIC0.SARX.BYTE = 0x01;                      /* FSX = 1 */
    IIC0.ICCR.BYTE = 0x81;                      /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                      /* ACKB = 0 */
    STCR.BYTE = 0x30;                           /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                      /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                      /* IEIC = 0, ACKE = 1 */
}
```

```
/*****************************************************
*  mst_rec : Master receive from EEPROM              *
*****************************************************/
void mst_rec(void)
{
     while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */

     IIC0.ICCR.BIT.MST = 1;                          /* Master transmit mode set */
     IIC0.ICCR.BIT.TRS = 1;                          /* MST = 1, TRS = 1 */
     set_start();                                    /* Start condition set */
     trs_slvadr_a0();                                /* Slave address + W data transmit */

     if(IIC0.ICSR.BIT.ACKB == 0)                     /* ACKB = 0 ? */
     {
          trs_memadr();                              /* EEPROM memory address data transmit */
          if(IIC0.ICSR.BIT.ACKB == 0)                /* ACKB = 0 ? */
          {
               set_start();                          /* Re-start condition set */
               trs_slvadr_a1();                      /* Slave address + R data transmit */
               if(IIC0.ICSR.BIT.ACKB == 0)           /* ACKB = 0 ? */
               {
                    rec_data();                      /* 1-byte data receive */
               }
          }
     }
     set_stop();                                     /* Stop condition set */
}

/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
void set_start(void)
{
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     IIC0.ICCR.BYTE = 0xbc;                          /* Start condition set (BBSY=1,SCP=0) */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Start condition set (IRIC=1) ? */
```

```
}

/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
     IIC0.ICCR.BYTE = 0xb8;                          /* Stop condition set (BBSY=0,SCP=0) */
     while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
}

/*****************************************************
*  trs_slvadr_a0 : Slave addres + W data transmit    *
*****************************************************/
void trs_slvadr_a0(void)
{
     IIC0.ICDR = 0xa0;                               /* Slave address + W data(H'A0) write */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_slvadr_a1 : Slave addres + R data transmit    *
*****************************************************/
void trs_slvadr_a1(void)
{
     IIC0.ICDR = 0xa1;                               /* Slave address + R data(H'A1) write */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_memadr : EEPROM memory address data transmit  *
*****************************************************/
void trs_memadr(void)
{
```

```
    IIC0.ICDR = 0x00;                           /* EEPROM memory address data(H'00) write */
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  rec_data : 1-byte data receive                    *
*****************************************************/
void rec_data(void)
{
    IIC0.ICCR.BIT.TRS = 0;                      /* Master receive mode set (MST=1,TRS=0) */
    IIC0.ICMR.BIT.WAIT = 1;                     /* WAIT = 1 */
    IIC0.ICSR.BIT.ACKB = 1;                     /* ACKB = 1 */

    dt_rec[0] = IIC0.ICDR;                      /* Dummy read (Receive start) */
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Receive end (IRIC=1) ? */

    IIC0.ICCR.BIT.TRS = 1;                      /* Master transmit mode set (MST=1,TRS=1) */
    IIC0.ICCR.BIT.IRIC = 0;                     /* 9th clock transmit start (IRIC=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* 9th clock transmit end (IRIC=1) ? */

    dt_rec[0] = IIC0.ICDR;                      /* 1-byte receive data read */

    IIC0.ICMR.BIT.WAIT = 0;                     /* WAIT = 0 */
    IIC0.ICSR.BIT.ACKB = 0;                     /* ACKB = 0 */
}
```

4.6 Single-Master Transmission by DTC

4.6.1 Specifications

- 10-byte data is written to EEPROM (HN58X2408) using channel 0 of the I²C bus interface in the H8S/2138 and the data transfer controller (DTC).
- The slave address of EEPROM to be connected is 1010000, and data is written to addresses H'00 to H'09 of the EEPROM memory.
- 10-byte data to be written is stored at addresses H'E102 to H'E10B in RAM.
- The device connected to the I²C bus in this system is a single–master configuration—one master device (H8S/2138) and one slave device (EEPROM).
- The transfer clock frequency is 100 kHz.
- Figure 4.17 shows an example of the H8S/2138 and EEPROM connection.

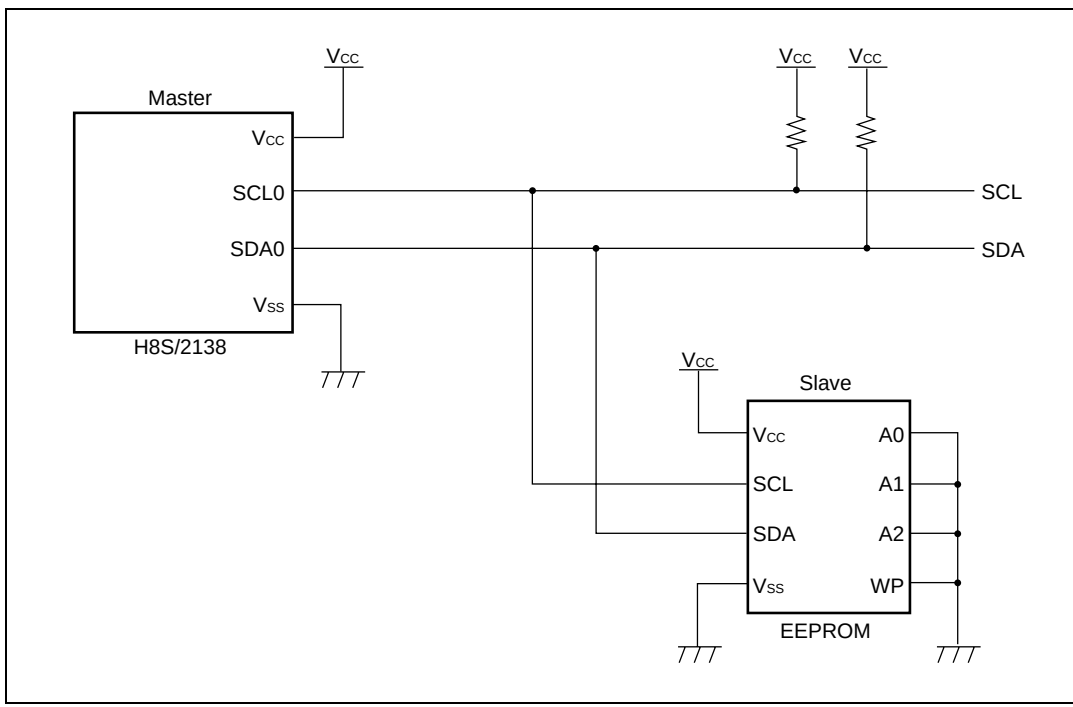


Figure 4.17 Example of H8S/2138 and EEPROM Connection

- Figure 4.18 shows the I²C bus format used in this task example.

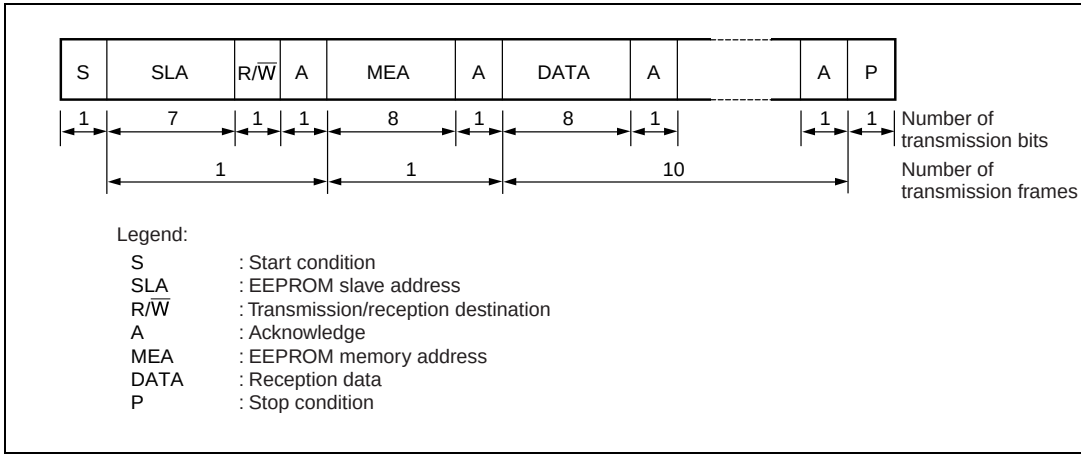


Figure 4.18 Transfer Format Used in This Task Example

- An example usage of the data transfer controller (DTC) in the H8S/2138 series used in this task example is described below.

(a) The DTC is activated by an interrupt request of channel 0 in the I^2C bus interface (IICI0) and transmission data is transferred.

(b) Normal mode is used for the DTC transfer mode.

(c) Figure 4.19 shows a block diagram of the DTC used in this task example.

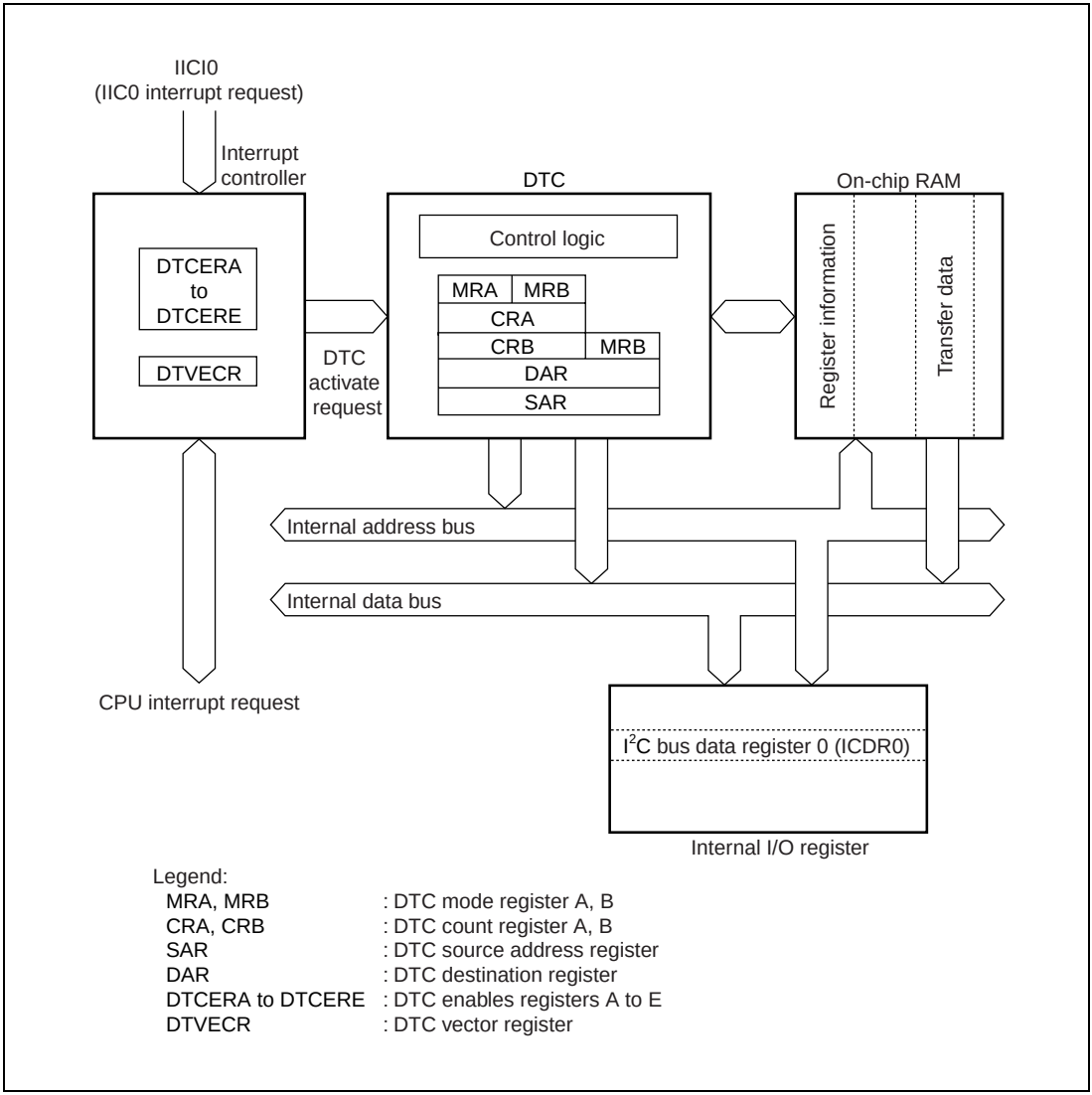


Figure 4.19 Block Diagram of DTC in This Task Example

(d) Figure 4.20 shows the location of transfer data on on-chip RAM.

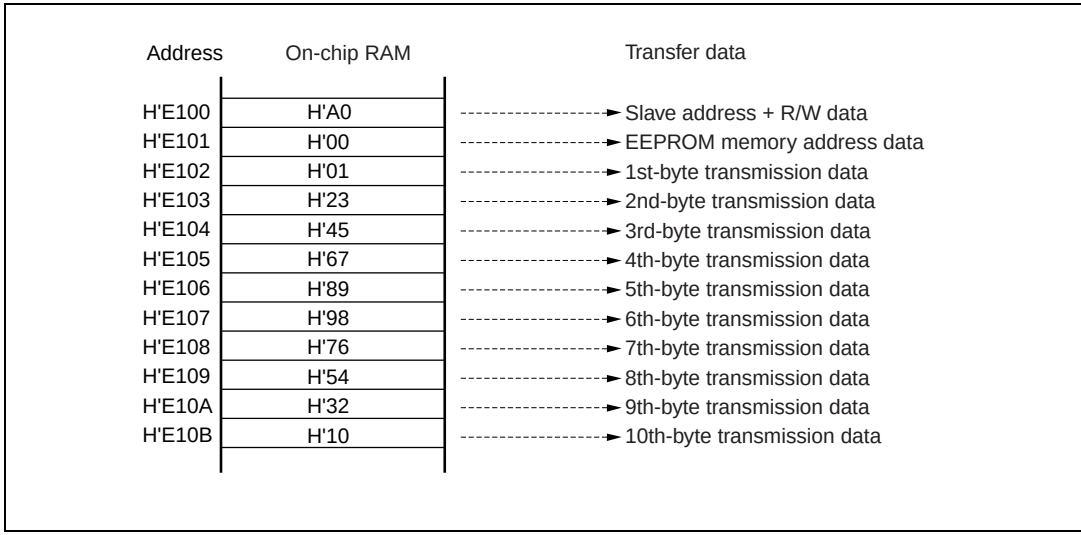


Figure 4.20 Location of Transfer Data on On-Chip RAM

(e) Figure 4.21 shows the location of DTC vector table and register information on the on-chip RAM in this task example. DTC register information is provided from address H'EC00 to the MRA, SAR, MRB, DAR, CRA, and CRB registers in that order.

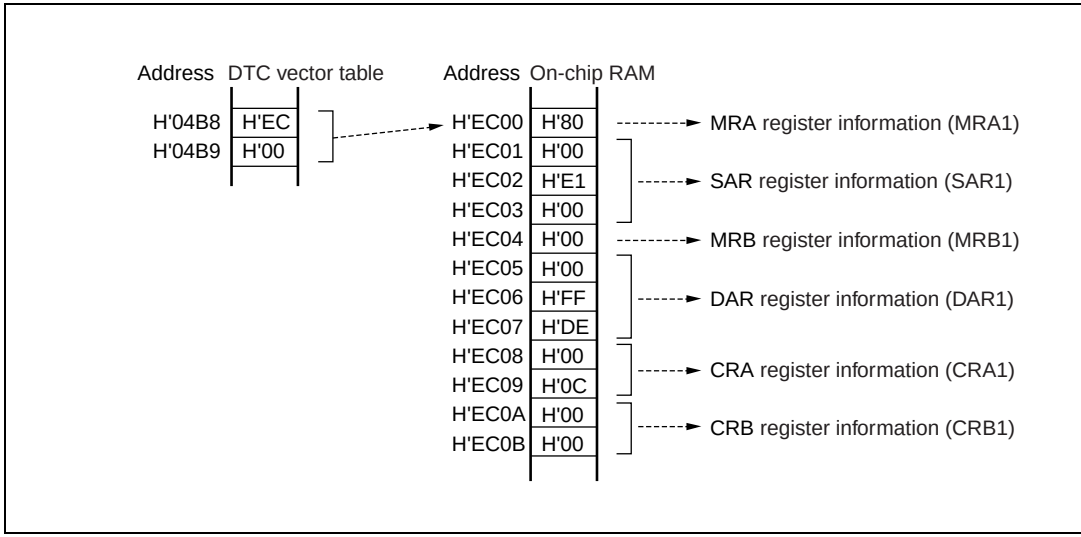


Figure 4.21 Location of DTC Vector Table and Register Information on On-Chip RAM

(f) Table 4.15 describes the register of the DTC used in this task example.

Table 4.15 DTC Register Description

Register		Function
MRA		DTC mode register A
		Controls DTC operating mode.
	SM1, 0	Source address mode 1, 0
	(bit7, 6)	Specify whether SAR is incremented, decremented, or fixed after data transfer is performed.
		When SM1 =0 and SM0 = *, SAR is fixed (*: 0 or 1)
		When SM1 =1 and SM0 = 0, SAR is incremented after transfer (when Sz = 0: + 1, when Sz = 1: + 2)
		When SM1 =1 and SM0 = 1, SAR is decremented after transfer (when Sz = 0: – 1, when Sz = 1: – 2)
	DM1, 0	Destination address mode 1, 0
	(bit5, 4)	Specify whether DAR is incremented, decremented, or fixed after data transfer is performed.
		When DM1 =0 and DM0 = *, DAR is fixed (*: 0 or 1)
		When DM1 =1 and DM0 = 0, DAR is incremented after transfer (when Sz = 0: + 1, when Sz = 1: + 2)
		When DM1 =1 and DM0 = 1, DAR is decremented after transfer (when Sz = 0: – 1, when Sz = 1: – 2)
	MD1, 0	DTC mode 1, 0
	(bit3, 2)	Specify DTC transfer mode.
		When MD1 = 0 and MD0 = 0, normal mode
		When MD1 = 0 and MD0 = 1, repeat mode
		When MD1 = 1 and MD0 = 0, block transfer mode
		When MD1 = 1 and MD0 = 1, setting prohibited
	DTS	DTC transfer mode select
	(bit1)	Specifies either source side or destination side becomes repeat area or block area in repeat mode or block.
		When DTS = 0, destination side becomes repeat area or block area.
		When DTS = 1, source side becomes repeat area or block area.
	Sz	DTC data transfer size
	(bit0)	Specifies data size in data transfer.

Table 4.15 DTC Register Description (cont)

Register		Function
MRB		DTC mode register B
		Controls DTC mode.
	CHEN	DTC chain transfer enable
	(bit7)	Specifies chain transfer
		When CHEN = 0, DTC data transfer is ended.
		When CHEN = 1, DTC chain transfer.
MRB	DISEL	DTC interrupt select
	(bit6)	Specifies an interrupt request to CPU is disabled or enabled after one data transfer is performed.
		When DISEL = 0, an interrupt to CPU is disabled if transfer counter is not 0 after the DTC data transfer is ended.
		When DISEL = 1, an interrupt to CPU is enabled after the DTC data transfer is ended.
SAR		DTC source address register
		Specifies the transfer source address of data to be transferred by the DTC.
DAR		DTC destination address register
		Specifies the transfer destination address of data to be transferred by the DTC.
CRA		DTC transfer count register A
		Specifies the number of data transfers by the DTC.
CRB		DTC transfer count register B
		Specifies the number of block data transfers by the DTC in block transfer mode.
DTVECR(H'FEF3)		DTC vector register
		Sets the DTC activation to be enabled or disabled by software and sets the vector address for the software activation interrupt.
	SWDTE	DTC software activation enable
	(bit7)	Sets the DTC software activation to be enabled or disabled.
		When SWDTE = 0, the DTC software activation is disabled.
		When SWDTE = 1, the DTC software activation is enabled.
	DTVEC 6-0	DTC software activation vectors 6 to 0
	(bit6-0)	Set the vector address for the DTC software activation.

Table 4.15 DTC Register Description (cont)

Register		Function
DTCERD(H'FEF1)		DTC enable register
		Controls the enabling or disabling of DTC activation by each interrupt source.
	DTCED4	DTC activation enable D4
	(bit4)	When DTCED4 = 0, the DTC activation is disabled by the IICI0 interrupt.
		When DTCED4 = 2, the DTC activation is enabled by the IICI0 interrupt.

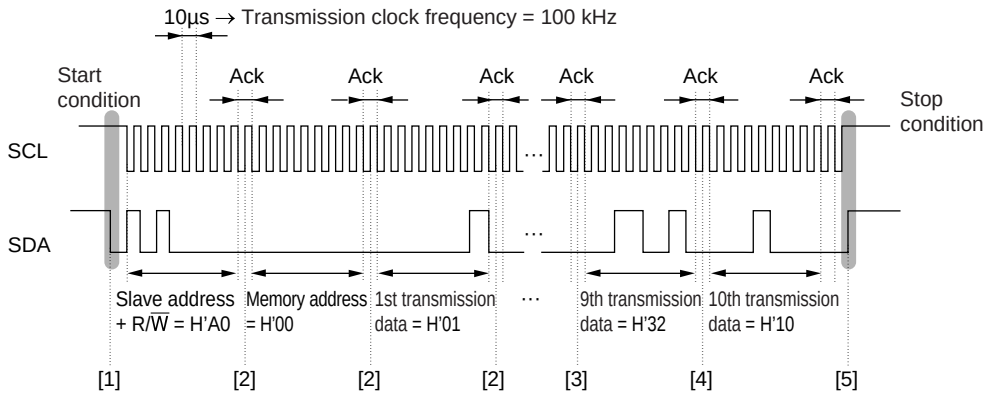
(g) The I^2C bus format provides for selection of the slave device and transfer direction by means of the slave address and the R/$\overline{W}$ bit, confirmation of reception with the acknowledge bit, indication of the last frame, and so on. Therefore, continuous data transfer using the DTC must be carried out in conjunction with CPU processing by means of interrupts. Table 4.16 shows an example of processing using the DTC in master transmission mode in this task example.

Table 4.16 Operation Example by DTC (master transmission mode)

Item	Master Transmission Mode
Slave address + R/$\overline{W}$ bit transmission	Transmission by DTC (ICDR write)
Dummy data read	—
Actual data transmission	Transmission by DTC (ICDR write)
Dummy data (H'FF) write	—
Last frame processing	Not necessary
Transfer request processing after last frame processing	1st time: Clearing by CPU 2nd time: End condition issuance by CPU
Setting of number of DTC transfer data frames	Transmission: Actual data count + 1(+ 1 equivalent to slave address + R/W bits)

4.6.2 Operation Description

Figure 4.22 shows the operation principle.



	CPU processing	DTC processing
[1]	Issues start condition (BBSY = 1, SCP = 0)	No processing
[2]	No processing	Writes transmission data to ICDR and clears IRIC to 0
[3]	Clears IRIC to 0 (1st IICI0 interrupt processing)	Writes transmission data to ICDR
[4]	Clears IRIC to 0 (2nd IICI0 interrupt processing)	No processing
[5]	Issues stop condition (BBSY = 0, SCP = 0)	No processing

Figure 4.22 Principle of Transmission Operation in Single Master by DTC

4.6.3 Software Description

(1) Module Description

Table 4.17 describes the module in this task example.

Table 4.17 Module Description

Module Name	Label Name	Function
Main routine	main	Sets the stack pointer and MCU mode, and enables interrupts.
Initial setting	initialize	Initial settings of using RAM area, IIC0 and the DTC
Transmission setup	trs_stup	Sets master transmission mode and issues start condition.
IIC0 interrupt processing	iici0	Clears IRIC and issues stop condition.

(2) On-Chip Register Description

Table 4.18 shows an on-chip register description in this task example.

Table 4.18 On-Chip Register Description

Register		Function	Address	Setting Value
ICDR0		Stores transmission/reception data.	H'FFDE	—
SAR0	FS	Sets transfer format with the FSX bit in SARX0 and the SW bit in DDCSWR.	H'FFDF bit0	0
SARX0	FSX	Sets transfer format with the FS bit in SAR0 and the SW bit in DDCSWR.	H'FFDE bit0	1
ICMR0	MLS	Sets data transfer by MSB first.	H'FFDF bit7	0
	WAIT	Sets whether wait is input or not between data and acknowledge bit.	H'FFDF bit6	0
	CKS2 to CKS0	Set transfer clock frequency to 100 kHz in conjunction with the IICX0 bit in STCR.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set number of data bits to be transferred next to 9 bits/frame by the I²C bus format.	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Controls access to ICMR0, ICDR0/SAR, SARX, and selects the I²C bus interface to operate (SCL0 and SDA0 pins function as port) or not to operate (SCL/SDA pins are in the bus drive state).	H'FFD8 bit7	0/1
	IEIC	Disables an interrupt request of the I²C bus interface.	H'FFD8 bit6	0
	MST	Uses the I²C bus interface in master mode.	H'FFD8 bit5	1
	TRS	Sets transmission/reception mode in the I²C bus interface.	H'FFD8 bit4	1
	ACKE	Suspends continuous transfer when an acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms the I²C bus is occupied or released, and issues start or stop condition in conjunction with the SCP bit.	H'FFD8 bit2	0/1

Table 4.18 On-Chip Register Description (cont)

Register		Function	Address	Setting Value
ICCR0	IRIC	Detects start condition, judges end of data transfer, and detects an acknowledge bit = 1.	H'FFD8 bit1	0/1
	SCP	Issues start or stop condition in conjunction with the BBSY bit.	H'FFD8 bit0	0
ICSR0	ACKB	Stores an acknowledge bit received from EEPROM in transmitting. Sets an acknowledge bit to be transferred to EEPROM in reception.	H'FFD9 bit0	-
STCR	IICX0	Sets the transfer clock frequency to 100 kHz in conjunction with CKS2 to CKS0 bits in ICMR0.	H'FFC3 bit5	1
	IICE	Enables access to CPU by the data and control registers of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets the control register in flash memory to be in non-selectable state.	H'FFC3 bit3	0
DDCSWR	SWE	Disables automatic switching from formatless of channel 0 in IIC to the I^2C bus format.	H'FEE6 bit7	0
	SW	Uses channel 0 in IIC in the I^2C bus format.	H'FEE6 bit6	0
	IE	Disables interrupts when format is switched automatically.	H'FEE6 bit5	0
	CLR3 to CLR0	Control initialization of an internal state in IIC0.	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels module stop mode in channel 0 in SCI.	H'FF87 bit7	0
	MSTP4	Cancels module stop mode in channel 0 in IIC.	H'FF87 bit4	0
SCR0	CKE1, 0	Set P52/SCK0/SCL0 pin as I/O port.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets operating mode in SCI0 to synchronous mode.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set interrupt control mode in interrupt controller to be controlled by the 1 bit.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set MCU operating mode to mode 3 by latching input levels of MD1 and MD0 pins.	H'FFC5 bit1, 0	MDS1=1 MDS0=1

Table 4.18 On-Chip Register Description (cont)

Register		Function	Address	Setting Value
MRA	SM1, 0	Set SAR to be incremented after data transfer.	H'EC00 bit7, 6	SM1=1 SM0=0
	DM1, 0	Set DAR to be fixed after data transfer.	H'EC00 bit5, 4	DM1=0 DM0=0
	MD1, 0	Set DTC transfer mode to normal mode.	H'EC00 bit3, 2	MD1=0 MD0=0
	DTS	Sets the destination source to be repeat area or block area.	H'EC00 bit1	DTS=0
	Sz	Sets data size in data transfer to be in byte size.	H'EC00 bit0	Sz=0
MRB	CHNE	Sets the DTC chain transfer to be disabled.	H'EC04 bit7	CHNE=0
	DISEL	Sets an interrupt to CPU to be disabled if the transfer counter is not 0 after one data transfer is performed.	H'EC04 bit6	DISSEL=0
SAR		Sets the transfer source address of data transferred by the DTC to H'E100.	H'EC01	H'00E100
DAR		Sets the transfer destination address of data transferred by the DTC to H'FFDE.	H'EC05	H'00FFDE
CRA		Sets the number of data transfers by the DTC to 12.	H'EC08	H'000C
CRB		Sets the number of block data transfers by the DTC in block transfer mode to 0.	H'EC0A	H'0000
DTVECR	SWDTE	Sets the DTC software activation to be disabled.	H'FEF3 bit7	0
	DTVEC6 to DTVEC0	Set the vector address of the DTC software activation to H'00.	H'FEF3 bit6 to bit0	H'00
DTCERD	DTCED4	Enables the DTC activation by the IICI0 interrupt.	H'FEF1 bit4	1
MSTPCRH	MSTP14	Cancels module stop mode of the DTC.	H'FF86 bit6	0

(3) Variable Description

Table 4.19 describes the variable in this task example.

Table 4.19 Variable Description

Variable	Function	Data Length	Initial Value	Module in Use
dummy	MDCR read value	1 byte	—	main
i	Transmit data counter	1 byte	H'00	initialize
dt_trs[0]	Slave address + W data	1 byte	H'A0	initialize
dt_trs[1]	EEPROM memory address data	1 byte	H'00	initialize
dt_trs[2]	1st-byte transmission data	1 byte	H'01	initialize
dt_trs[3]	2nd-byte transmission data	1 byte	H'23	initialize
dt_trs[4]	3rd-byte transmission data	1 byte	H'45	initialize
dt_trs[5]	4th-byte transmission data	1 byte	H'67	initialize
dt_trs[6]	5th-byte transmission data	1 byte	H'89	initialize
dt_trs[7]	6th-byte transmission data	1 byte	H'98	initialize
dt_trs[8]	7th-byte transmission data	1 byte	H'76	initialize
dt_trs[9]	8th-byte transmission data	1 byte	H'54	initialize
dt_trs[10]	9th-byte transmission data	1 byte	H'32	initialize
dt_trs[11]	10th-byte transmission data	1 byte	H'10	initialize

(4) Description of RAM Used

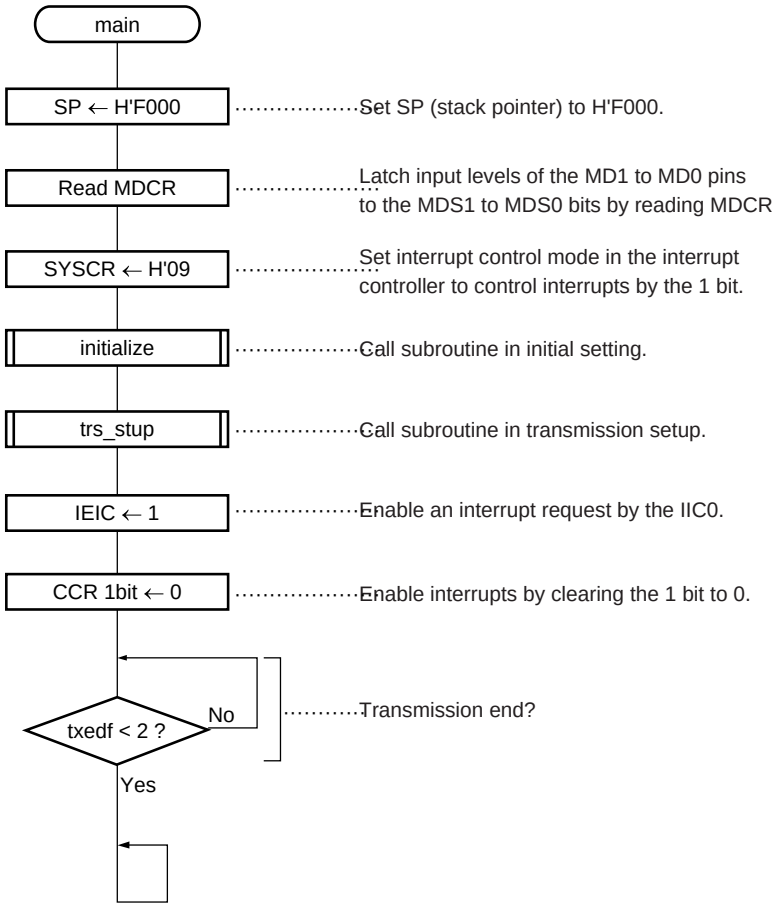
Table 4.20 describes the RAM used in this task example.

Table 4.20 Description of RAM Used

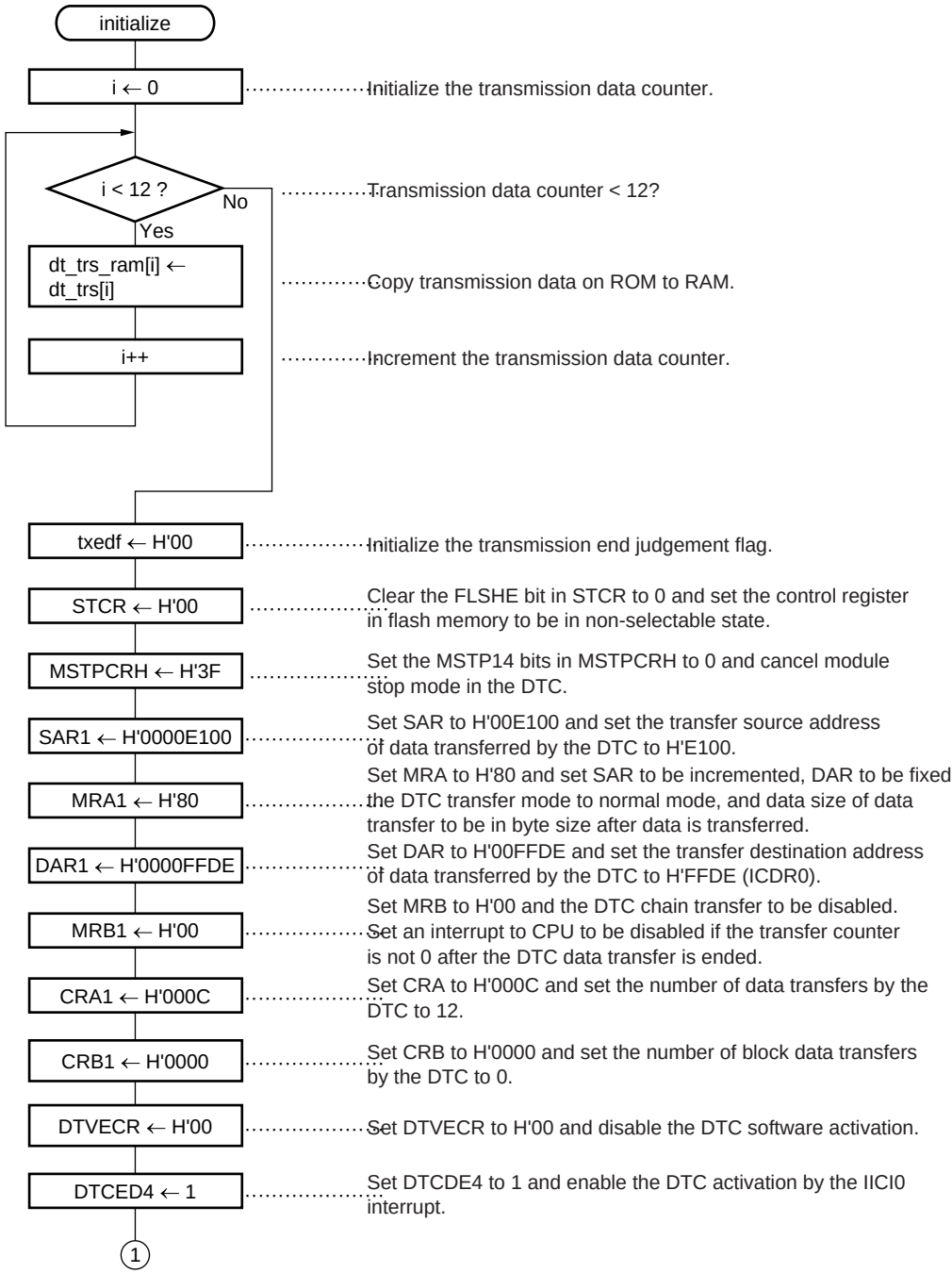
Label	Function	Data Length	Address	Module in Use
MRA1	DTC mode register A (MRA)	1 byte	H'EC00	initialize
SAR1	DTC source address register (SAR)	4 bytes	H'EC00	initialize
MRB1	DTC mode register B (MRB)	1 byte	H'EC04	initialize
DAR1	DTC destination address register (DAR)	4 bytes	H'EC04	initialize
CRA1	DTC transfer count register A (CRA)	2 bytes	H'EC08	initialize
CRB1	DTC transfer count register B (CRB)	2 bytes	H'EC0A	initialize
txedf	Transmission end judgement flag	1 byte	H'E200	main iici0
dt_trs_ram [0]	Stores slave address + R/W data.	1 byte	H'E100	initialize
dt_trs_ram [1]	Stores EEPROM memory address data.	1 byte	H'E101	initialize
dt_trs_ram [2] to dt_trs_ram [11]	Stores 10-byte transmission data.	10 bytes	H'E102 or H'E10B	initialize

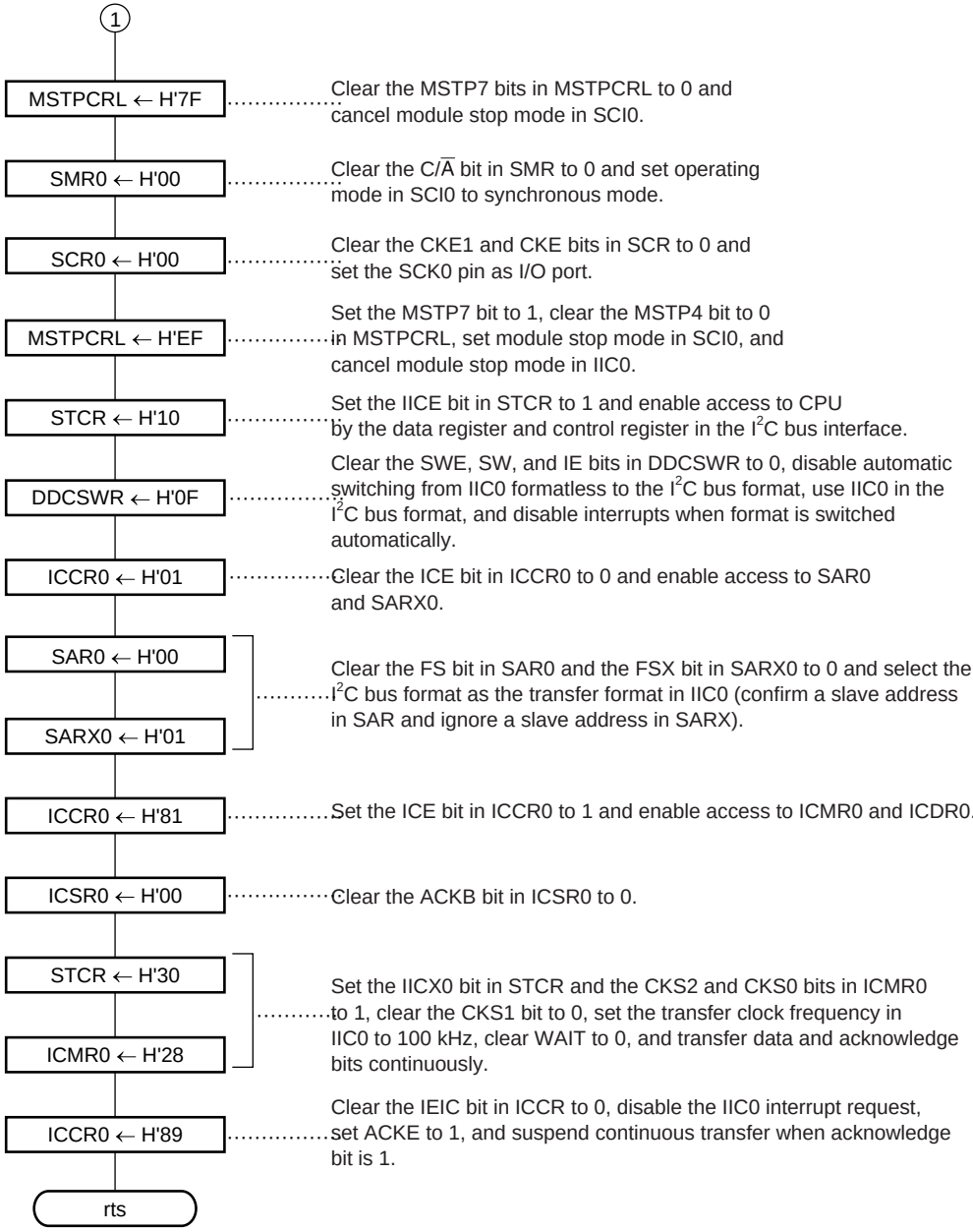
4.6.4 Flowchart

(1) Main Routine

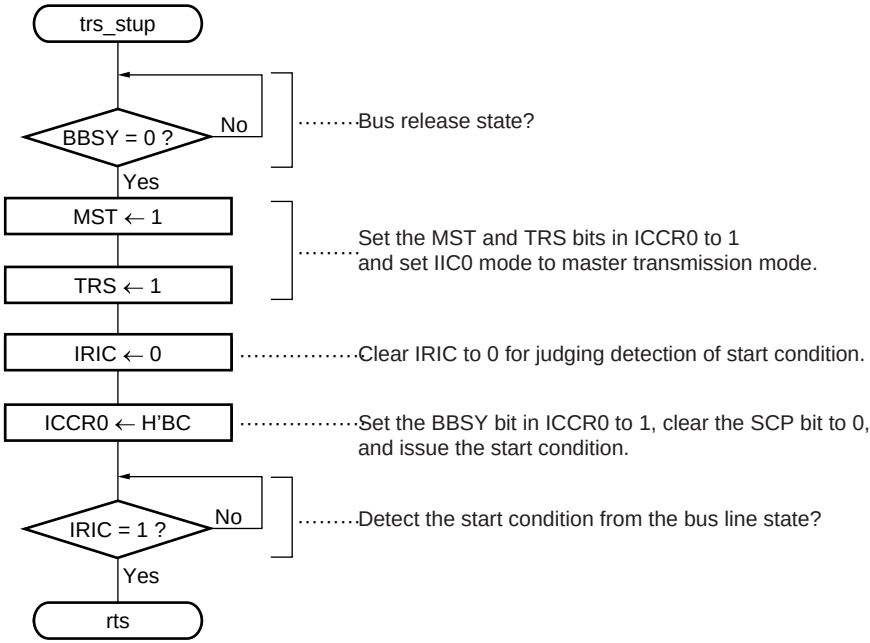


(2) Initial Setting Subroutine

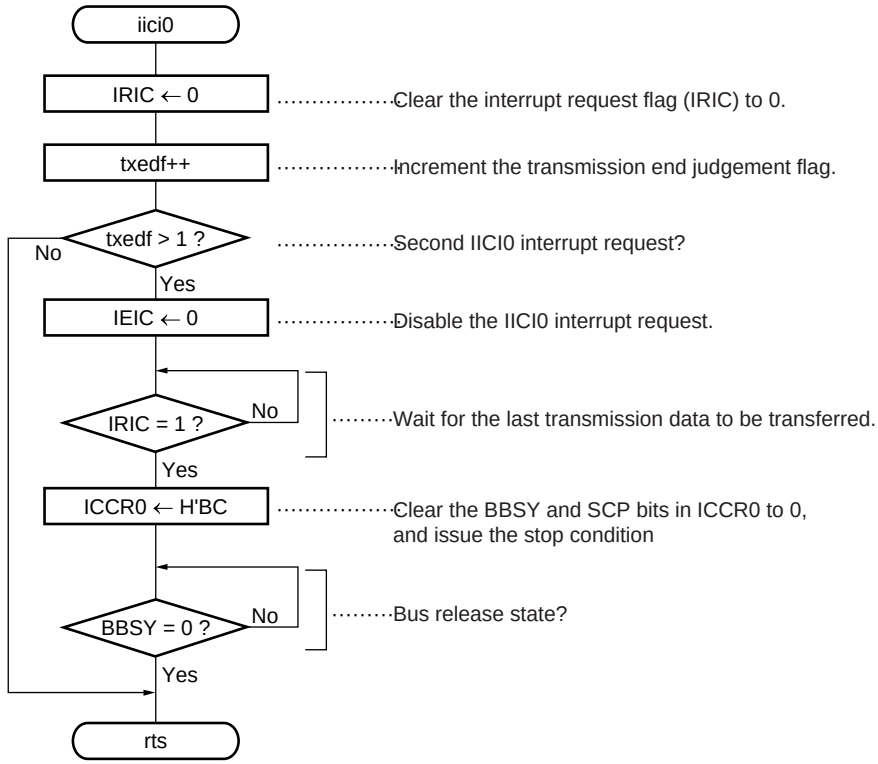



1
MSTPCRL ← H'7F
Clear the MSTP7 bits in MSTPCRL to 0 and cancel module stop mode in SCI0.
SMR0 ← H'00
Clear the C/$\overline{A}$ bit in SMR to 0 and set operating mode in SCI0 to synchronous mode.
SCR0 ← H'00
Clear the CKE1 and CKE bits in SCR to 0 and set the SCK0 pin as I/O port.
MSTPCRL ← H'EF
Set the MSTP7 bit to 1, clear the MSTP4 bit to 0 in MSTPCRL, set module stop mode in SCI0, and cancel module stop mode in IIC0.
STCR ← H'10
Set the IICE bit in STCR to 1 and enable access to CPU by the data register and control register in the I2C bus interface.
DDCSWR ← H'0F
Clear the SWE, SW, and IE bits in DDCSWR to 0, disable automatic switching from IIC0 formatless to the I2C bus format, use IIC0 in the I2C bus format, and disable interrupts when format is switched automatically.
ICCR0 ← H'01
Clear the ICE bit in ICCR0 to 0 and enable access to SAR0 and SARX0.
SAR0 ← H'00
SARX0 ← H'01
Clear the FS bit in SAR0 and the FSX bit in SARX0 to 0 and select the I2C bus format as the transfer format in IIC0 (confirm a slave address in SAR and ignore a slave address in SARX).
ICCR0 ← H'81
Set the ICE bit in ICCR0 to 1 and enable access to ICMR0 and ICDR0.
ICSR0 ← H'00
Clear the ACKB bit in ICSR0 to 0.
STCR ← H'30
ICMR0 ← H'28
Set the IICX0 bit in STCR and the CKS2 and CKS0 bits in ICMR0 to 1, clear the CKS1 bit to 0, set the transfer clock frequency in IIC0 to 100 kHz, clear WAIT to 0, and transfer data and acknowledge bits continuously.
ICCR0 ← H'89
Clear the IEIC bit in ICCR to 0, disable the IIC0 interrupt request, set ACKE to 1, and suspend continuous transfer when acknowledge bit is 1.
rts

(3) Transmission Setup Subroutine



(4) IIC0 Interrupt Processing Routine



4.6.5 Program List

```
/******************************************************
*  H8S/2138 IIC bus application note                  *
*    5.Single master transmit by DTC                  *
*                       File name   :   DTCtx.c       *
*                       Fai         :   20MHz         *
*                       Mode        :   3             *
******************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/******************************************************
*  Prototype                                          *
******************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* RAM & DTC & IIC0 initialize */
void trs_stup(void);                                /* Master transmit by DTC set up */

/******************************************************
*  RAM allocation                                     *
******************************************************/
#define MRA1 (*(volatile unsigned char *)0xec00)     /* DTC mode register A */
#define SAR1 (*(volatile unsigned long *)0xec00)     /* DTC source address register */
#define MRB1 (*(volatile unsigned char *)0xec04)     /* DTC mode register B */
#define DAR1 (*(volatile unsigned long *)0xec04)     /* DTC destination address register */
#define CRA1 (*(volatile unsigned short *)0xec08)    /* DTC transfer count register A */
#define CRB1 (*(volatile unsigned short *)0xec0a)    /* DTC transfer count register B */

#define txedf (*(volatile unsigned char *)0xe200)    /* Transmit end flag */

#pragma section ramerea
unsigned char dt_trs_ram[12];                        /* Transmit data store area */
#pragma section
```

```
/*****************************************************
*  Data table                                        *
*****************************************************/
const unsigned char dt_trs[12] =
{
     0xa0,                                          /* Slave address + W data */
      0x00,                                         /* EEPROM memory address data */
     0x01,                                          /* 1st transmit data */
     0x23,                                          /* 2nd transmit data */
     0x45,                                          /* 3rd transmit data */
     0x67,                                          /* 4th transmit data */
     0x89,                                          /* 5th transmit data */
     0x98,                                          /* 6th transmit data */
     0x76,                                          /* 7th transmit data */
     0x54,                                          /* 8th transmit data */
     0x32,                                          /* 9th transmit data */
     0x10                                           /* 10th transmit data */
};


/*****************************************************
*  main : Main routine                               *
*****************************************************/
void main(void)
#pragma asm
          mov.l   #h'f000,sp                        ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                             /* MCU mode set */
     SYSCR.BYTE = 0x09;                             /* Interrupt control mode set */

     initialize();                                  /* Initialize */
     trs_stup();                                    /* Master transmit by DTC set up */

     IIC0.ICCR.BIT.IEIC = 1;                        /* IIC0 interrupt enable */
     set_imask_ccr(0);                              /* Interrupt enable */
```

```
    while(txedf < 2);                           /* Transmit end ? */
    while(1);                                   /* End */
}

/*****************************************************
*  initialize : RAM & IIC0 Initialize               *
*****************************************************/
void initialize(void)
{
    unsigned char i;                            /* Transmit data counter */

    for(i=0; i<12; i++)                         /* Transmit data copy ROM -> RAM */
    {
        dt_trs_ram[i] = dt_trs[i];
    }

    txedf = 0x00;                               /* Transmit end flag initialize */

    STCR.BYTE = 0x00;                           /* FLSHE = 0 */

    MSTPCR.BYTE.H = 0x3f;                       /* DTC module stop mode reset */
    SAR1 = 0x0000e100;                          /* SAR = H'00E100 */
    MRA1 = 0x80;                                /* MRA = H'80 */
    DAR1 = 0x0000ffde;                          /* DAR = H'00FFED (ICDR0) */
    MRB1 = 0x00;                                /* MRB = H'00 */
    CRA1 = 0x000c;                              /* CRA = H'000C */
    CRB1 = 0x0000;                              /* CRB = H'0000 */
    DTC.VECR.BYTE = 0x00;                       /* SWDTE = 0, DTVEC = H'00 */
    DTC.ED.BIT.B4 = 1;                          /* DTCED4 = 1 */

    MSTPCR.BYTE.L = 0x7f;                       /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                       /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    MSTPCR.BYTE.L = 0xef;                       /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                           /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                         /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                      /* ICE = 0 */
```

```
    IIC0.SAR.BYTE = 0x00;                       /* FS = 0 */
    IIC0.SARX.BYTE = 0x01;                      /* FSX = 1 */
    IIC0.ICCR.BYTE = 0x81;                      /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                      /* ACKB = 0 */
    STCR.BYTE = 0x30;                           /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                      /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                      /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  trs_stup : Master transmit by DTC set up          *
*****************************************************/
void trs_stup(void)
{
    while(IIC0.ICCR.BIT.BBSY == 1);             /* Bus empty (BBSY=0) ? */
    IIC0.ICCR.BIT.MST = 1;                      /* Matser transmit mode set */
    IIC0.ICCR.BIT.TRS = 1;                      /* MST = 1, TRS = 1 */

    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    IIC0.ICCR.BYTE = 0xbc;                      /* Start condition set (BBSY=1,SCP=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Start condition set (IRIC=1) ? */
}

/*****************************************************
*  iici0 : IIC0 interrupt routine                    *
*****************************************************/
#pragma interrupt(iici0)
void iici0(void)
{
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    txedf++;

    if(txedf > 1)
    {
        IIC0.ICCR.BIT.IEIC = 0;                 /* IIC0 interrupt disable */
        while(IIC0.ICCR.BIT.IRIC == 0);         /* Transmit end (IRIC=0) ? */
```

```
            IIC0.ICCR.BYTE = 0xb8;                      /* Stop condition set (BBSY=0,SCP=0) */
            while(IIC0.ICCR.BIT.BBSY == 1);             /* Bus empty (BBSY=0) ? */
        }
    }
```

4.7 Single-Master Reception by DTC

4.7.1 Specifications

- 10-byte data is read from EEPROM (HN58X2408) using channel 0 of the I^2C bus interface in the H8S/2138 and the data transfer controller (DTC).
- The slave address of EEPROM to be connected is 1010000, and data is read from addresses H'00 to H'09 of the EEPROM memory.
- 10-byte data to be read is stored at addresses H'E100 to H'E109 in RAM.
- The device connected to the I^2C bus in this system is a single–master configuration—one master device (H8S/2138) and one slave device (EEPROM).
- The transfer clock frequency is 100 kHz.
- Figure 4.23 shows an example of the H8S/2138 and EEPROM connection.

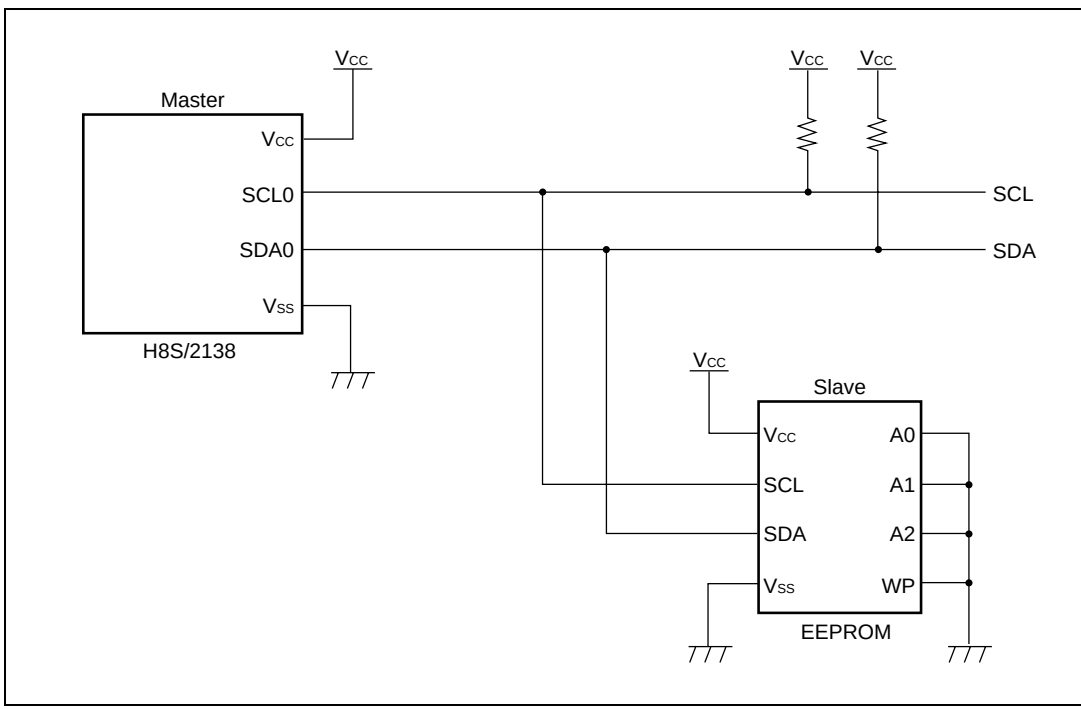


Figure 4.23 Example of H8S/2138 and EEPROM Connection

- Figure 4.24 shows the I^2C bus format used in this task example.

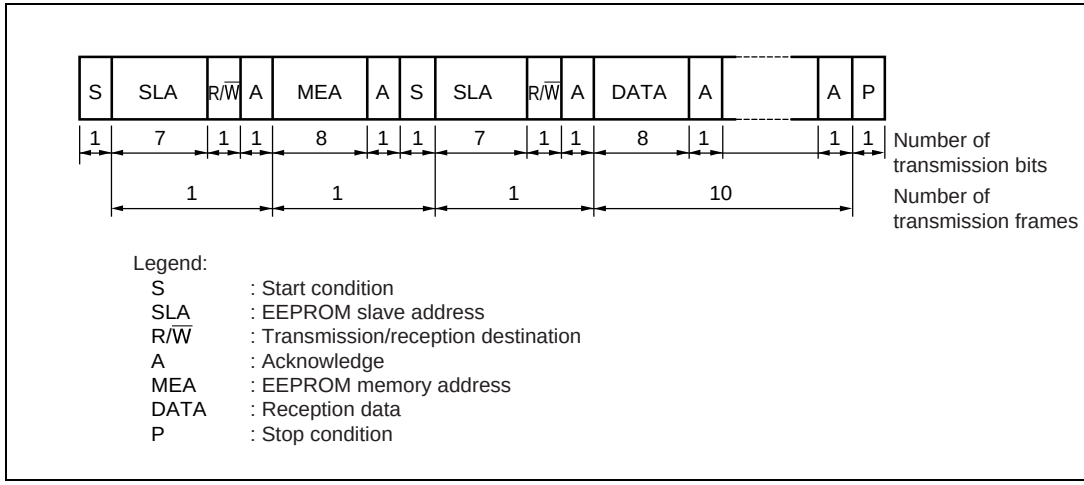


Figure 4.24 Transfer Format Used in This Task Example

- An example usage of the data transfer controller (DTC) in the H8S/2138 Series used in this task example is described below.

(a) The DTC is activated by an interrupt request of channel 0 in the I^2C bus interface (IICI0) and reception data is transferred.

(b) Normal mode is used for the DTC transfer mode.

(c) Figure 4.25 shows a block diagram of the DTC used in this task example.

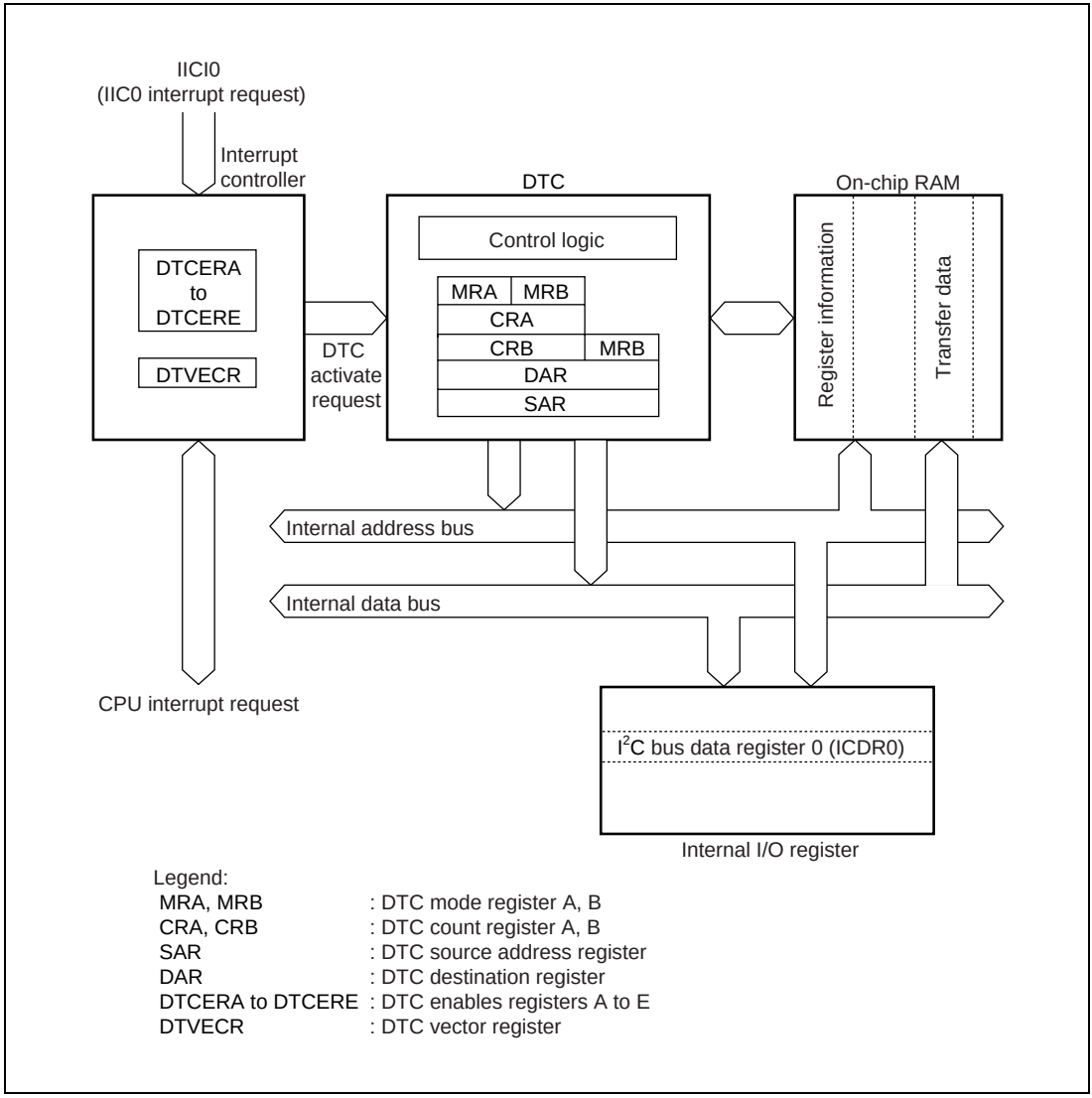


Figure 4.25 Block Diagram of DTC in This Task Example

(d) Figure 4.26 shows the location of transfer data on on-chip RAM.

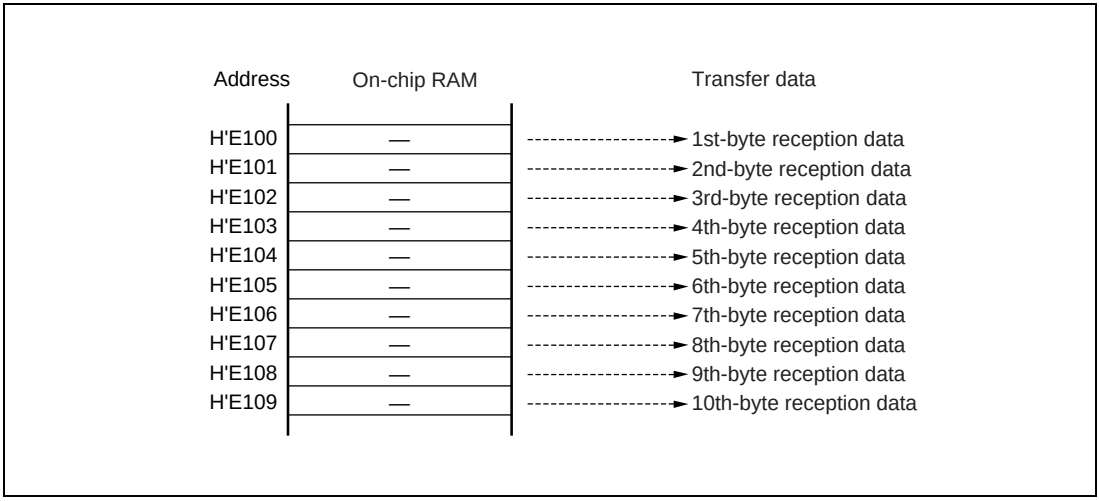


Figure 4.26 Location of Transfer Data on On-Chip RAM

(e) Figure 4.27 shows the location of DTC vector table and register information on the on-chip RAM in this task example. DTC register information is provided from address H'EC00 to the MRA, SAR, MRB, DAR, CRA, and CRB registers in that order.

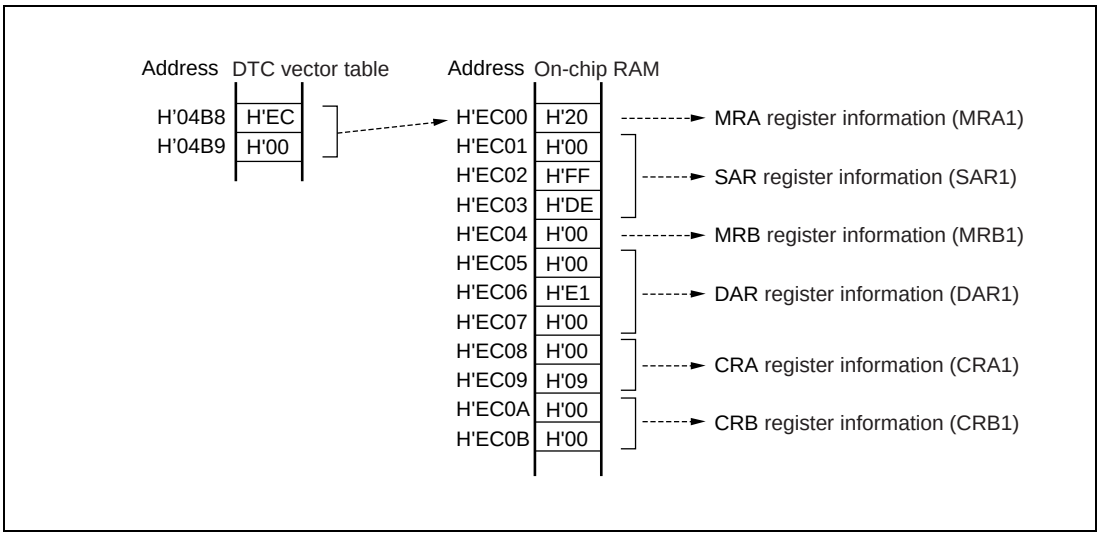


Figure 4.27 Location of DTC Vector Table and Register Information on On-Chip RAM

(f) Table 4.21 describes the register of the DTC used in this task example.

Table 4.21 DTC Register Description

Register		Function
MRA		DTC mode register A
		Controls DTC operating mode.
	SM1, 0	Source address mode 1, 0
	(bit7, 6)	Specify whether SAR is incremented, decremented, or fixed after data transfer is performed.
		When SM1 =0 and SM0 = *, SAR is fixed (*: 0 or 1).
		When SM1 = 1 and SM0 = 0, SAR is incremented after transfer (when Sz = 0: + 1, when Sz = 1: + 2).
		When SM1 =1 and SM0 = 1, SAR is decremented after transfer (when Sz = 0: – 1, when Sz = 1: – 2).
	DM1, 0	Destination address mode 1, 0
	(bit5, 4)	Specify whether DAR is incremented, decremented, or fixed after data transfer is performed.
		When DM1 = 0 and DM0 = *, DAR is fixed (*: 0 or 1).
		When DM1 = 1 and DM0 = 0, DAR is incremented after transfer (when Sz = 0: + 1, when Sz = 1: + 2).
		When DM1 = 1 and DM0 = 1, DAR is decremented after transfer (when Sz = 0: – 1, when Sz = 1: – 2).
	MD1, 0	DTC mode 1, 0
	(bit3, 2)	Specify DTC transfer mode.
		When MD1 = 0 and MD0 = 0, normal mode.
		When MD1 = 0 and MD0 = 1, repeat mode.
		When MD1 = 1 and MD0 = 0, block transfer mode.
		When MD1 = 1 and MD0 = 1, setting prohibited.
	DTS	DTC transfer mode select
	(bit1)	Specifies either source side or destination side becomes repeat area or block area in repeat mode or block transfer mode.
		When DTS = 0, destination side becomes repeat area or block area.
		When DTS = 1, source side becomes repeat area or block area.
	Sz	DTC data transfer size
	(bit0)	Specifies data size in data transfer.

Table 4.21 DTC Register Description (cont)

Register		Function
MRB		DTC mode register B
		Controls DTC mode.
	CHEN	DTC chain transfer enable
	(bit7)	Specifies chain transfer.
		When CHEN = 0, DTC data transfer is ended.
		When CHEN = 1, DTC chain transfer.
MRB	DISEL	DTC interrupt select
	(bit6)	Specifies an interrupt request to CPU is disabled or enabled after one data transfer is performed.
		When DISEL = 0, an interrupt to CPU is disabled if transfer counter is not 0 after the DTC data transfer is ended.
		When DISEL = 1, an interrupt to CPU is enabled after the DTC data transfer is ended.
SAR		DTC source address register
		Specifies the transfer source address of data to be transferred by the DTC.
DAR		DTC destination address register
		Specifies the transfer destination address of data to be transferred by the DTC.
CRA		DTC transfer count register A
		Specifies the number of data transfers by the DTC.
CRB		DTC transfer count register B
		Specifies the number of block data transfers by the DTC in block transfer mode.
DTVECR (H'FEF3)		DTC vector register
		Sets the DTC activation to be enabled or disabled by software and sets the vector address for the software activation interrupt.
	SWDTE	DTC software activation enable
	(bit7)	Sets the DTC software activation to be enabled or disabled.
		When SWDTE = 0, the DTC software activation is disabled.
		When SWDTE = 1, the DTC software activation is enabled.
	DTVEC 6-0	DTC software activation vectors 6 to 0
	(bit6-0)	Set the vector address for the DTC software activation.

Table 4.21 DTC Register Description (cont)

Register		Function
DTCERD (H'FEF1)		DTC enable register
		Controls the enabling or disabling of DTC activation by each interrupt source.
	DTCED4	DTC activation enable D4
	(bit4)	When DTCED4 = 0, the DTC activation is disabled by the IICI0 interrupt.
		When DTCED4 = 2, the DTC activation is enabled by the IICI0 interrupt.

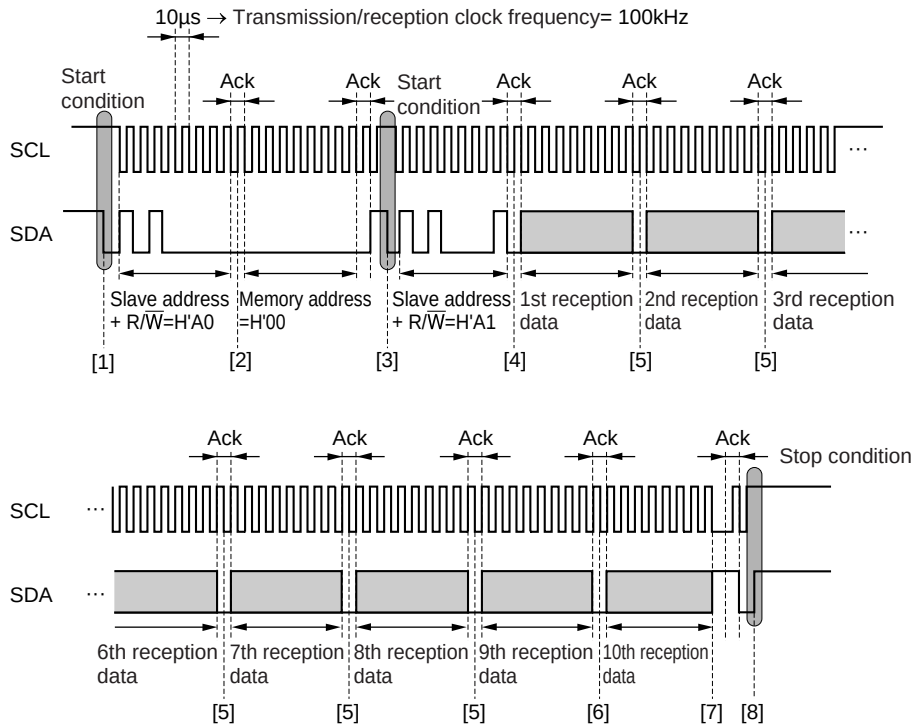
(g) The I^2C bus format provides for selection of the slave device and transfer direction by means of the slave address and the R/$\overline{W}$ bit, confirmation of reception with the acknowledge bit, indication of the last frame, and so on. Therefore, continuous data transfer using the DTC must be carried out in conjunction with CPU processing by means of interrupts. Table 4.22 shows an example of processing using the DTC in master transmission mode in this task example.

Table 4.22 Operation Example by DTC (master reception mode)

Item	Master Transmission Mode
Slave address + R/W bit transmission	Transmission by CPU (ICDR write)
Dummy data read	Processing by CPU (ICDR read)
Actual data transmission	Reception by DTC (ICDR read)
Dummy data (H'FF) write	—
Last frame processing	Not necessary
Transfer request processing after last frame processing	Not necessary
Setting of number of DTC transfer data frames	Reception: Actual data count

4.7.2 Description of Operation

Figure 4.28 shows the principle of operation.



	CPU processing	DTC processing
[1]	IRIC=0, Setting of start condition (BBSY=1, SCP =0), transmission of slave address + W bit, IRIC=0	No operation
[2]	EEPROM address transmission, IRIC=0	No operation
[3]	IRIC=0, Setting of start condition (BBSY=1, SCP =0), transmission of slave address + R bit, IRIC=0	No operation
[4]	Dummy read, IRIC=0, IEIC=1	No operation, IRIC = 0
[5]	No operation	Reading of the data for reception
[6]	IEIC=1, ACKB=1, WAIT=1 Reading of the 9th byte of data for reception, IRIC=0	No operation
[7]	TRS=1 Reading of the 10th byte of data for reception, IRIC=0	No operation
[8]	Setting of the stop condition (BBSY=1, SCP=0)	No operation

Figure 4.28 Principle of a Single-Master Receive Operation by DTC

4.7.3 Description of Software

(1) Description of Modules

Table 4.23 describes the modules of this example of a task.

Table 4.23 Description of Modules

Module Name	Label Name	Function
Main routine	main	Sets the stack pointer and the MCU mode, and enables the interrupt.
Initial settings	initialize	Sets the RAM area to be used, and makes initial settings for IIC0 and DTC.
Setting of start condition	set_start	Sets the start condition.
Setting of stop condition	set_stop	Sets the stop condition.
Transmission of slave address + W	trs_slvadr_a0	Transmits the EEPROM's slave address and W data (H'A0).
Transmission of slave address + R	trs_slvadr_a1	Transmits the EEPROM's slave address and W data (H'A1).
Transmission of the EEPROM memory address	trs_memadr	Transmits the EEPROM's address in memory (H'00).
Processing of the IIC0 interrupt	iici0	Clears IrIC, disables the IICI0 interrupt, and sets the reception-completed flag.

(2) Description of the On-chip Registers

Table 4.24 describes the on-chip registers used in this example of a task.

Table 4.24 Description of the On-chip Registers

Register		Function	Address	Setting
ICDR0		Stores the received data.	H'FFDE	—
SAR0	FS	Along with the settings of the FSX bit of SARX0 and the SW bit of DDCSWR, sets the transfer format.	H'FFDF bit0	0
SARX0	FSX	Along with the settings of the FS bit of SAR0 and the SW bit of DDCSWR, sets the transfer format.	H'FFDE bit0	1
ICMR0	MLS	Sets the transfer of data as MSB first.	H'FFDF bit7	0
	WAIT	Selects insertion and non-insertion of wait cycles between the data and the acknowledge bit.	H'FFDF bit6	0/1
	CKS2 to CKS0	Along with the setting in the IICX0 bit of STCR, set the frequency of the transfer clock to 100 kHz.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set the number of bits of data for the next transfer in the I^{2}C bus format to 9 bits/frame.	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Controls access to the ICMR0, ICDR0/SAR, and SARX registers, and selects operation (port function for the SCL0/SDA0 pin) or non-operation (bus-drive state for the SCL/SDA pin) of the I^{2}C bus interface.	H'FFD8 bit7	0/1
	IEIC	Disables the generation of interrupt requests by the I^{2}C bus interface.	H'FFD8 bit6	0/1
ICCR0	MST	Uses the I^{2}C bus interface in the master mode.	H'FFD8 bit5	1
	TRS	Sets transmission/reception mode for the I^{2}C bus interface.	H'FFD8 bit4	0/1
	ACKE	Suspends continuous transfer when the acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms whether or not the I^{2}C bus is occupied, and uses the SCP bit to set the start and stop conditions.	H'FFD8 bit2	0/1
	IRIC	Detects the start condition, determines the end of data transfer, and detects acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Along with the BBSY bit, sets the start/stop conditions.	H'FFD8 bit0	0

Table 4.24 Description of On-chip Registers (cont)

Register		Function	Address	Setting
ICSR0	ACKB	Stores the acknowledgement received from the EEPROM during transmission. Sets the acknowledge bit for transmission to the EEPROM during reception.	H'FFD9 bit0	—
STCR	IICX0	Along with the settings in CKS2 to CKS0 of ICMR0, selects the frequency of the transfer clock.	H'FFC3 bit5	1
	IICE	Enables CPU access to the data and control registers of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets the control registers of the flash memory to non-selected.	H'FFC3 bit3	0
DDCSWR	SWE	Prohibits automatic change from format-less transfer to transfer in the I^2C bus format on the channel 0 I^2C interface.	H'FEE6 bit7	0
	SW	Uses the channel 0 I^2C interface in the I^2C bus format.	H'FEE6 bit6	0
	IE	Prohibits interrupts during automatic changes of format.	H'FEE6 bit5	0
	CLR3 to CLR0	Control the initialization of the internal state of the I^2C interface	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels the module-stopped mode for SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels the module stopped mode for I^2C channel 0.	H'FF87 bit4	0
SCR0	CKE1, 0	Makes the I/O port setting for the P52/SCK0/SCL0 pin.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets the mode for SCI transfer on channel 0 as asynchronous.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set the interrupt control mode of the interrupt controller to 1-bit control.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set the MCU's operating mode to mode 3 by latching the input levels on the MD1 and 0 pins.	H'FFC5 bit1, 0	MDS1=1 MDS0=1

Table 4.24 Description of On-chip Registers (cont)

Register		Function	Address	Setting
MRA	SM1, 0	Set SAR to remain fixed after data has been transferred.	H'EC00 bit7, 6	SM1=0 SM0=0
	DM1, 0	Set DAR to be incremented after data has been transferred.	H'EC00 bit5, 4	DM1=1 DM0=0
	MD1, 0	Set the DTC transfer mode to normal.	H'EC00 bit3, 2	MD1=0 MD0=0
MRA	DTS	Sets the destination area to the repeat area or the block area.	H'EC00 bit1	DTS=0
	Sz	Sets bytes as the unit for data transfer.	H'EC00 bit0	Sz=0
MRB	CHNE	Disables DTC-chain transfer.	H'EC04 bit7	CHNE=0
	DISEL	Prohibits the generation of an interrupt signal for the CPU after a single transfer of data unless the transfer counter is 0.	H'EC04 bit6	DISEL=0
SAR		Sets the transfer source address transferred by the DTC to H'FFDE.	H'EC01	H'00FFDE
DAR		Sets the transfer destination address transferred by the DTC to H'E100.	H'EC05	H'00E100
CRA		Sets the DTC transfer count to 12.	H'EC08	H'000C
CRB		Sets the DTC block-data transfer count to 0 during transfer in block-transfer mode.	H'EC0A	H'0000
DTVECR	SWDTE	Prohibits the activation of the DTC software.	H'FEF3 bit7	0
	DTVEC6 to DTVEC0	Set the vector number of for the activation of the DTC software to H'00.	H'FEF3 bit6 to bit0	H'00
DTCERD	DTCED4	Enables DTC activation by the I^2CI0 interrupt.	H'FEF1 bit4	1
MSTPCRH	MSTP14	Removes the DTC from its module-stopped mode.	H'FF86 bit6	0

(3) Description of Variables

Table 4.25 describes the variables used in this task.

Table 4.25 Description of Variables

Variable	Function	Size	Initial Value	Module Name
dummy	MDCR read value	1 byte	—	Main
i	Received data counter	1 byte	H'00	Initialize

(4) Description of RAM Usage

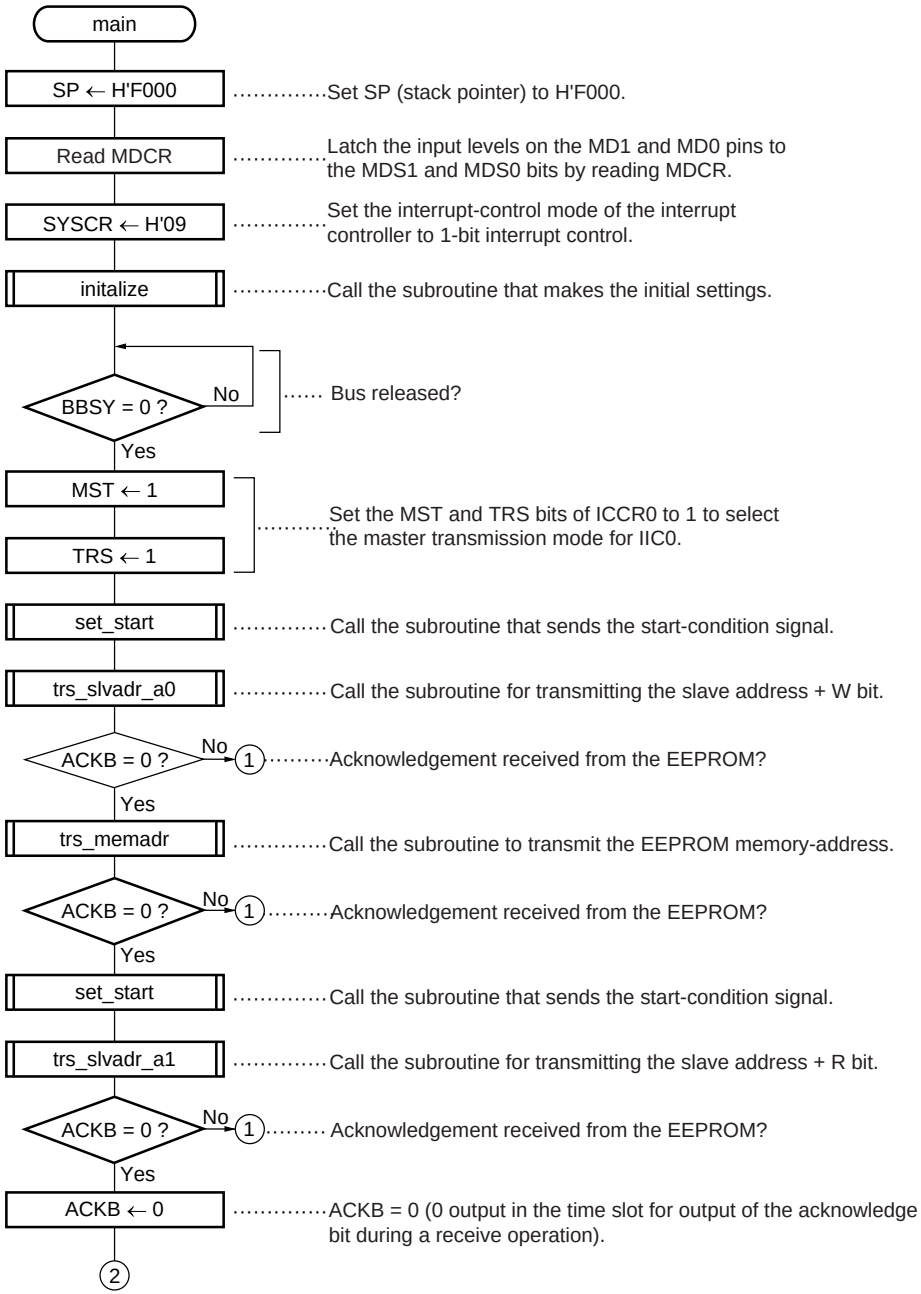
Table 4.26 describes the usage of RAM in this example of a task.

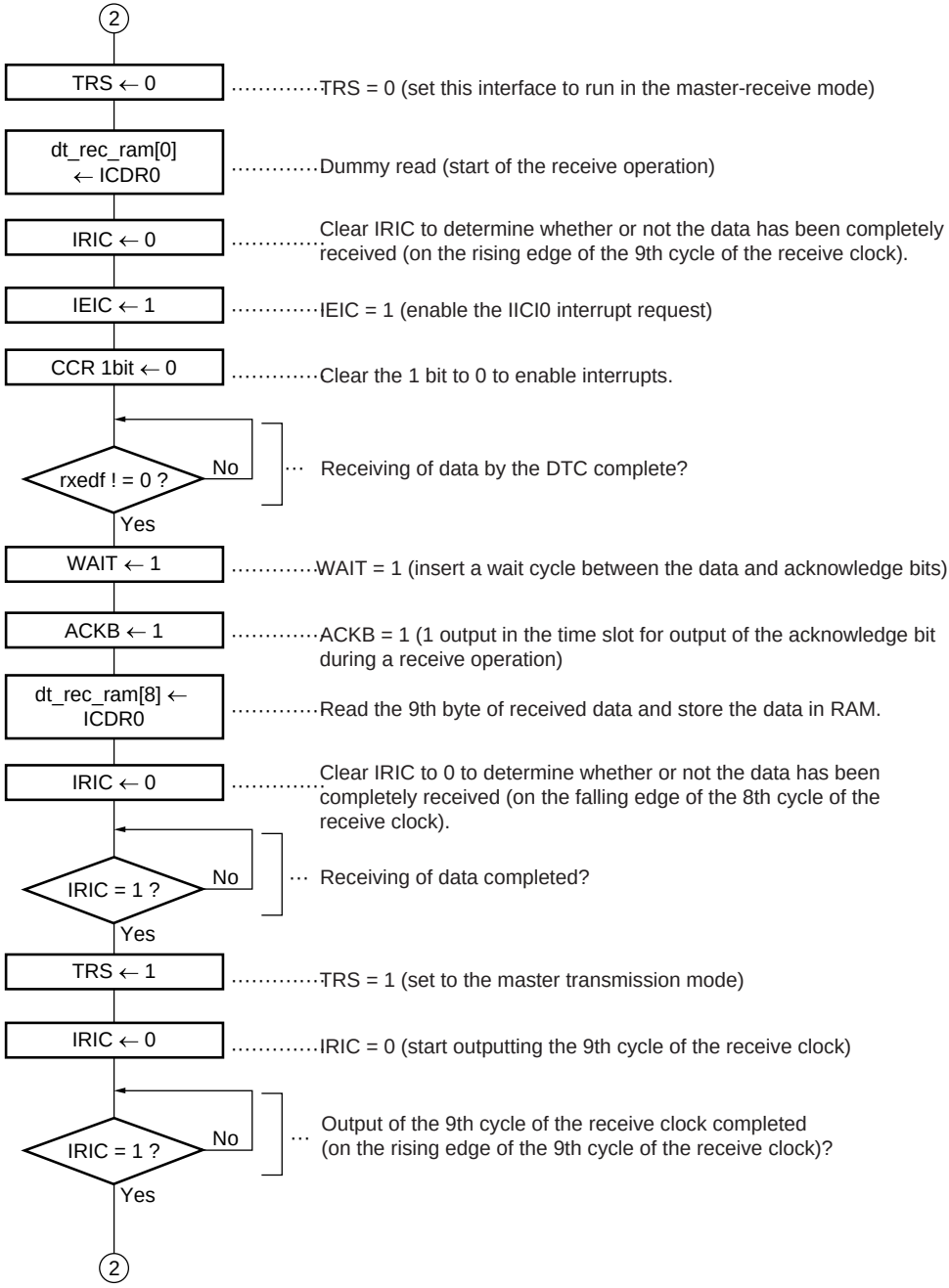
Table 4.26 Description of RAM Usage

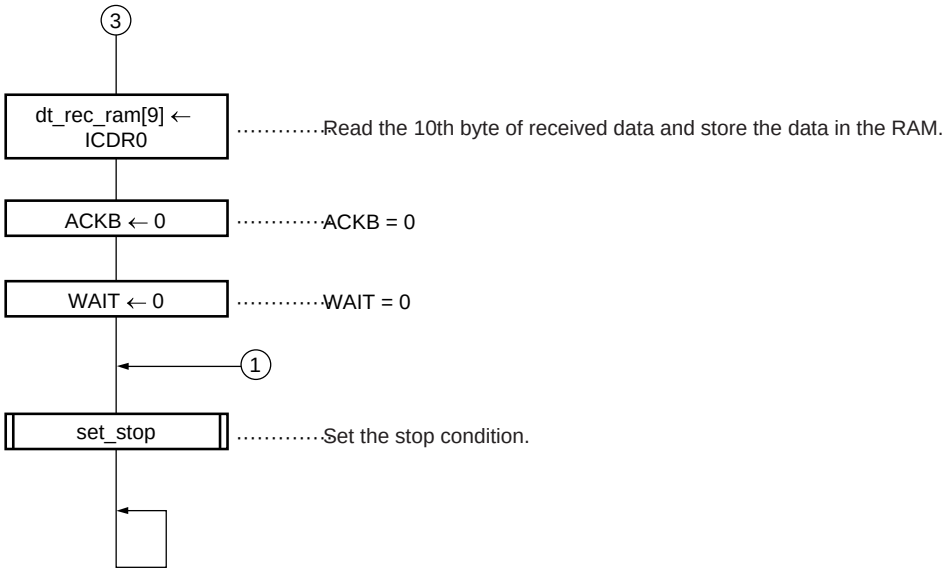
Label	Function	Size	Address	Module Name
MRA1	DTC mode register	1 byte	H'EC00	initialize
SAR1	DTC source address register	4 bytes	H'EC00	initialize
MRB1	DTC mode register B	1 byte	H'EC04	initialize
DAR1	DTC destination address register	4 bytes	H'EC04	initialize
CRA1	DTC transfer count register A	2 bytes	H'EC08	initialize
CRB1	DTC transfer count register B	2 bytes	H'EC0A	initialize
rxedf	Reception-completed flag	1 byte	H'E200	main iici0
dt_rec_ram [0] to dt_rec_ram [9]	Stores 10 bytes of received data.	10 bytes	H'E100 to H'E109	main initialize

4.7.4 Flowchart

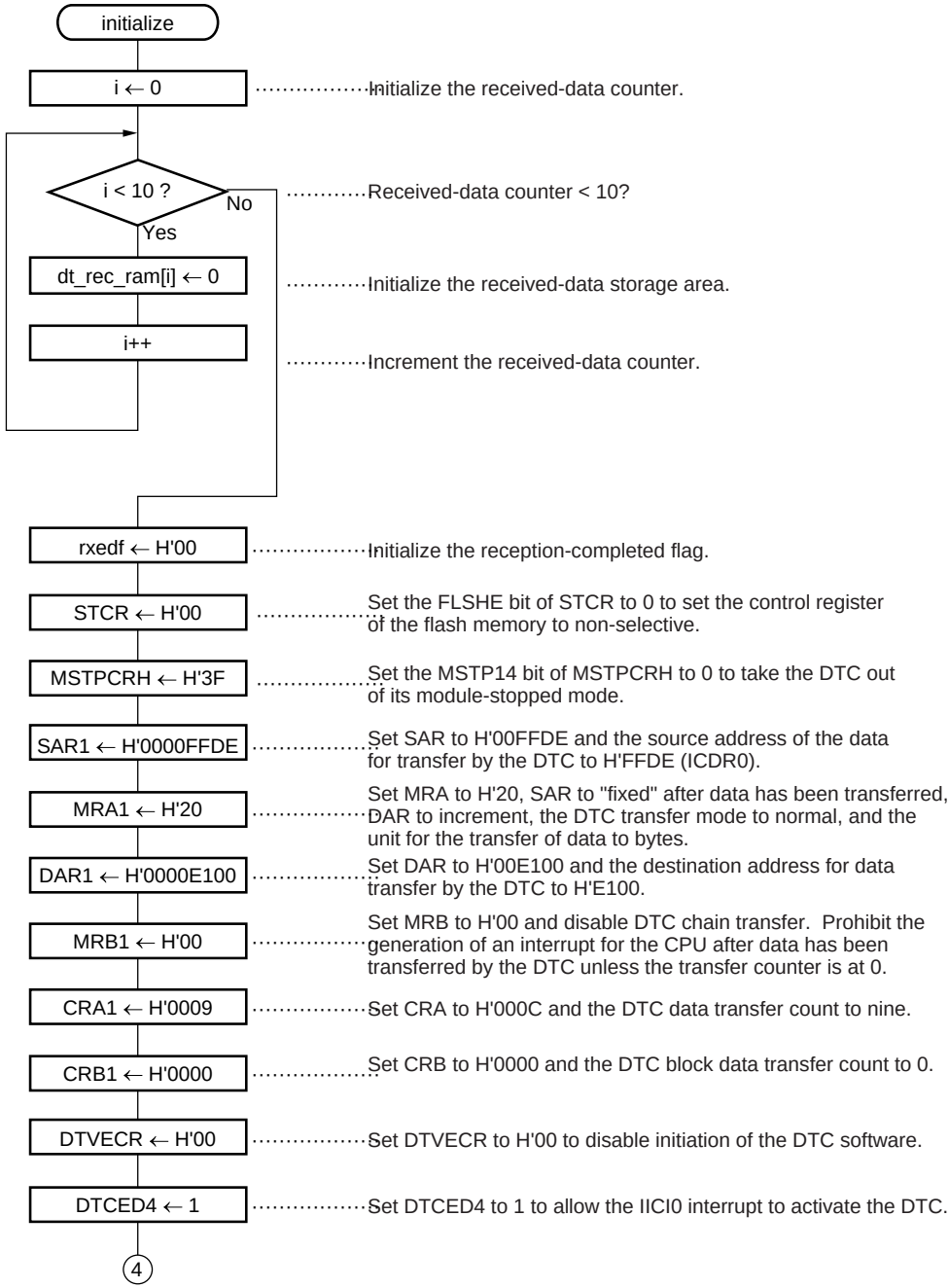
(1) Main Routine

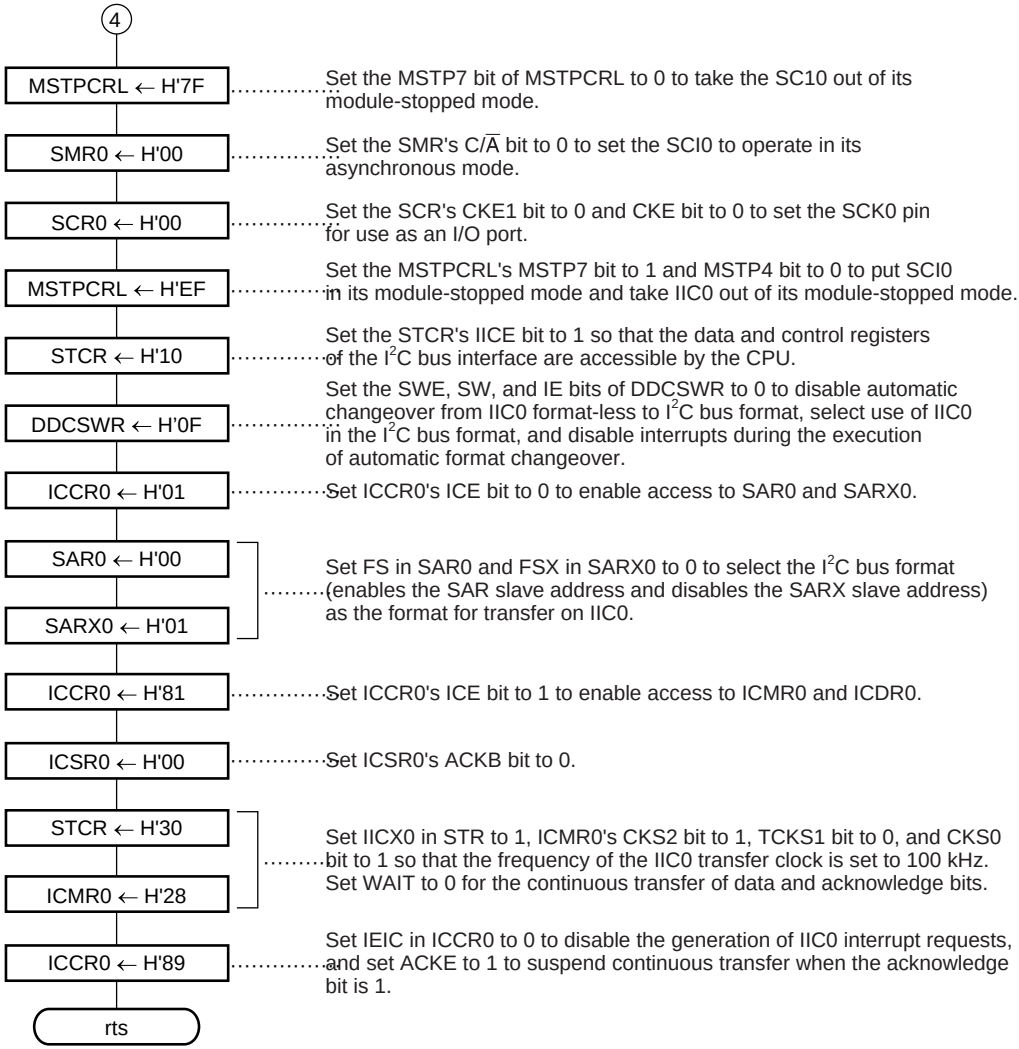



2
TRS ← 0
TRS = 0 (set this interface to run in the master-receive mode)
dt_rec_ram[0] ← ICDR0
Dummy read (start of the receive operation)
IRIC ← 0
Clear IRIC to determine whether or not the data has been completely received (on the rising edge of the 9th cycle of the receive clock).
IEIC ← 1
IEIC = 1 (enable the IICI0 interrupt request)
CCR 1bit ← 0
Clear the 1 bit to 0 to enable interrupts.
rxedf ! = 0 ?
No
Yes
Receiving of data by the DTC complete?
WAIT ← 1
WAIT = 1 (insert a wait cycle between the data and acknowledge bits)
ACKB ← 1
ACKB = 1 (1 output in the time slot for output of the acknowledge bit during a receive operation)
dt_rec_ram[8] ← ICDR0
Read the 9th byte of received data and store the data in RAM.
IRIC ← 0
Clear IRIC to 0 to determine whether or not the data has been completely received (on the falling edge of the 8th cycle of the receive clock).
IRIC = 1 ?
No
Yes
Receiving of data completed?
TRS ← 1
TRS = 1 (set to the master transmission mode)
IRIC ← 0
IRIC = 0 (start outputting the 9th cycle of the receive clock)
IRIC = 1 ?
No
Yes
Output of the 9th cycle of the receive clock completed (on the rising edge of the 9th cycle of the receive clock)?
2

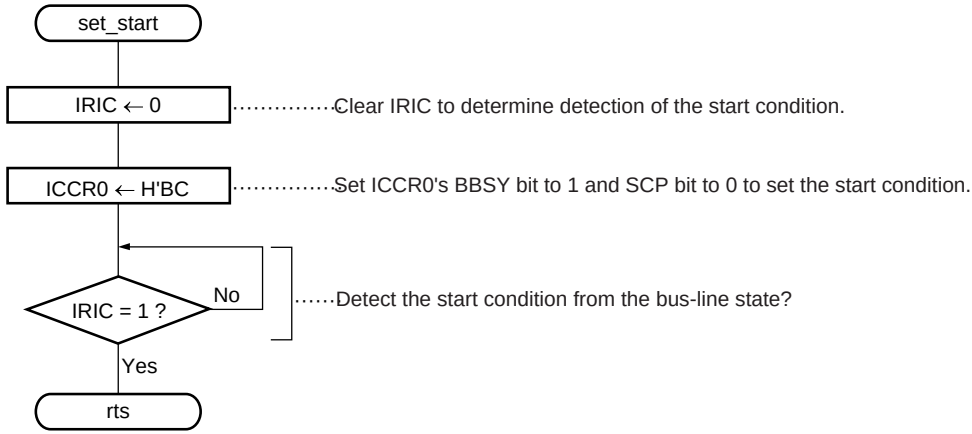

3
dt_rec_ram[9] ← ICDR0
Read the 10th byte of received data and store the data in the RAM.
ACKB ← 0
ACKB = 0
WAIT ← 0
WAIT = 0
1
set_stop
Set the stop condition.

(2) Subroutine for Making Initial Settings

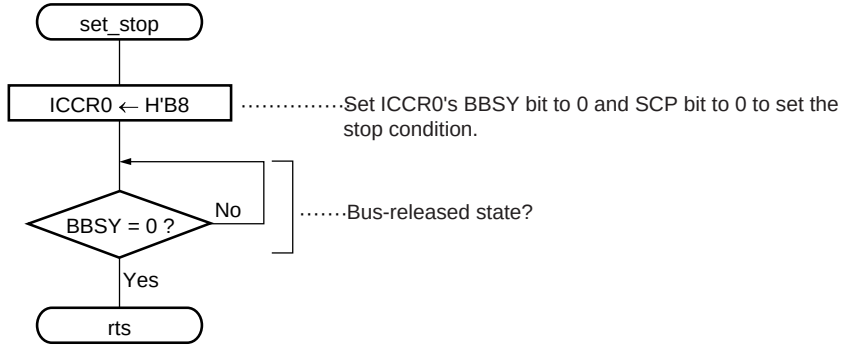



4
MSTPCRL ← H'7F
Set the MSTP7 bit of MSTPCRL to 0 to take the SC10 out of its module-stopped mode.
SMR0 ← H'00
Set the SMR's C/$\overline{A}$ bit to 0 to set the SCI0 to operate in its asynchronous mode.
SCR0 ← H'00
Set the SCR's CKE1 bit to 0 and CKE bit to 0 to set the SCK0 pin for use as an I/O port.
MSTPCRL ← H'EF
Set the MSTPCRL's MSTP7 bit to 1 and MSTP4 bit to 0 to put SCI0 in its module-stopped mode and take IIC0 out of its module-stopped mode.
STCR ← H'10
Set the STCR's IICE bit to 1 so that the data and control registers of the I2C bus interface are accessible by the CPU.
DDCSWR ← H'0F
Set the SWE, SW, and IE bits of DDCSWR to 0 to disable automatic changeover from IIC0 format-less to I2C bus format, select use of IIC0 in the I2C bus format, and disable interrupts during the execution of automatic format changeover.
ICCR0 ← H'01
Set ICCR0's ICE bit to 0 to enable access to SAR0 and SARX0.
SAR0 ← H'00
SARX0 ← H'01
Set FS in SAR0 and FSX in SARX0 to 0 to select the I2C bus format (enables the SAR slave address and disables the SARX slave address) as the format for transfer on IIC0.
ICCR0 ← H'81
Set ICCR0's ICE bit to 1 to enable access to ICMR0 and ICDR0.
ICSR0 ← H'00
Set ICSR0's ACKB bit to 0.
STCR ← H'30
ICMR0 ← H'28
Set IICX0 in STR to 1, ICMR0's CKS2 bit to 1, TCKS1 bit to 0, and CKS0 bit to 1 so that the frequency of the IIC0 transfer clock is set to 100 kHz. Set WAIT to 0 for the continuous transfer of data and acknowledge bits.
ICCR0 ← H'89
Set IEIC in ICCR0 to 0 to disable the generation of IIC0 interrupt requests, and set ACKE to 1 to suspend continuous transfer when the acknowledge bit is 1.
rts

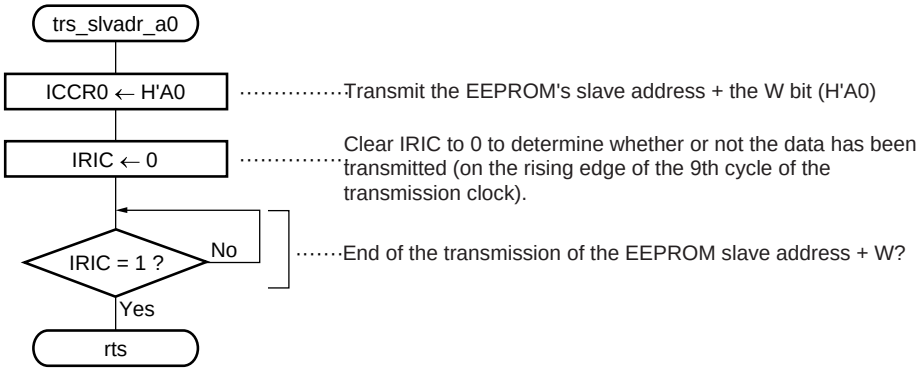
(3) Subroutine for Setting the Start Condition



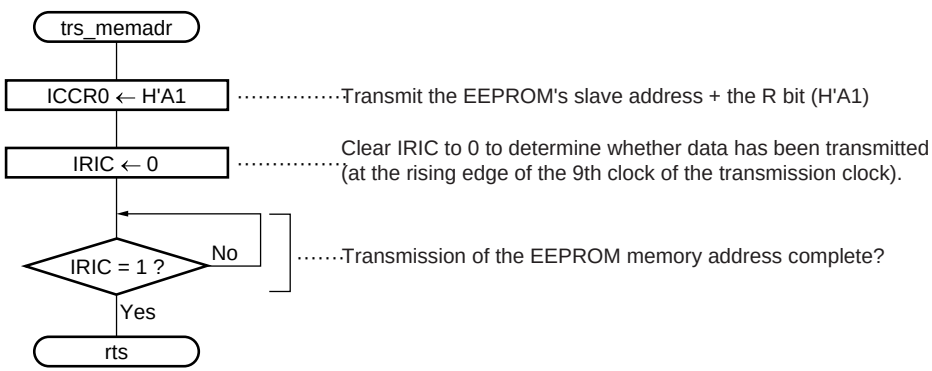
(4) Subroutine for Setting the Stop Condition



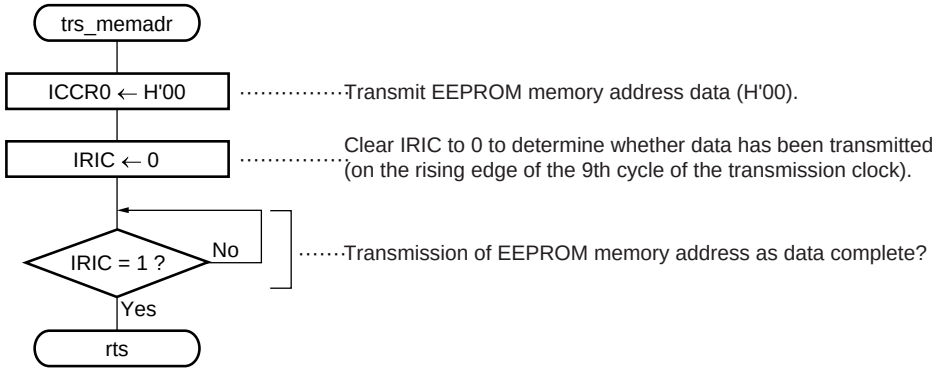
(5) Subroutine for Transmitting the Slave Address + W



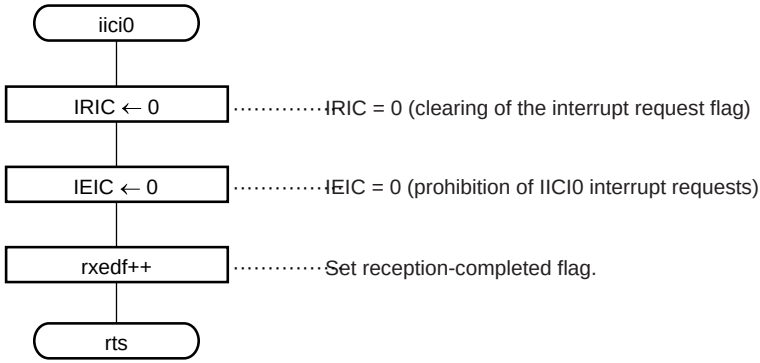
(6) Subroutine for Transmitting the Slave Address + R



(7) Subroutine for Transmitting the EEPROM memory address



(8) IIC0 Interrupt-Processing Routine



4.7.5 Program List

```
/*******************************************************
*  H8S/2138 IIC bus application note                   *
*        6.Single master receive by DTC                *
*                      File name   :   DTCrx.c         *
*                      Fai         :   20MHz           *
*                      Mode        :   3               *
*******************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*******************************************************
*  Prototype                                           *
*******************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* RAM & DTC & IIC0 initialize */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_slvadr_a1(void);                           /* Slave address + R data transmit */
void trs_memadr(void);                              /* EEPROM memory address data transmit */

/*******************************************************
*  RAM allocation                                      *
*******************************************************/
#define MRA1 (*(volatile unsigned char *)0xec00)     /* DTC mode register A */
#define SAR1 (*(volatile unsigned long *)0xec00)     /* DTC source address register */
#define MRB1 (*(volatile unsigned char *)0xec04)     /* DTC mode register B */
#define DAR1 (*(volatile unsigned long *)0xec04)     /* DTC destination address register */
#define CRA1 (*(volatile unsigned short *)0xec08)    /* DTC transfer count register A */
#define CRB1 (*(volatile unsigned short *)0xec0a)    /* DTC transfer count register B */

#define rxedf (*(volatile unsigned char *)0xe200)    /* Receive end flag */

#pragma section ramarea
```

```
unsigned char dt_rec_ram[10];                    /* Receive data store erea */
#pragma section

/*****************************************************
*  main : Main routine                               *
*****************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                      ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                          /* MCU mode set */
     SYSCR.BYTE = 0x09;                          /* Interrupt control mode set */
     initialize();                               /* Initialize */

     while(IIC0.ICCR.BIT.BBSY == 1);             /* Bus empty (BBSY=0) ? */
     IIC0.ICCR.BIT.MST = 1;                      /* Master transmit mode set */
     IIC0.ICCR.BIT.TRS = 1;                      /* MST=1, TRS=1 */
     set_start();                                /* Start condition set */
     trs_slvadr_a0();                            /* Slave address + W data transmit */
     if(IIC0.ICSR.BIT.ACKB == 0)                 /* ACKB = 0 ? */
     {
          trs_memadr();                          /* EEPROM memory address data transmit */
          if(IIC0.ICSR.BIT.ACKB == 0)            /* ACKB = 0 ? */
          {
               set_start();                      /* Re-start condition set */
               trs_slvadr_a1();                  /* Slave address + R data transmit */
               if(IIC0.ICSR.BIT.ACKB == 0)       /* ACKB = 0 ? */
               {
                    IIC0.ICSR.BIT.ACKB = 0;      /* ACKB = 0 */
                    IIC0.ICCR.BIT.TRS = 0;       /* Master receive mode set */

                    dt_rec_ram[0] = IIC0.ICDR;   /* Dummy read */

                    IIC0.ICCR.BIT.IRIC = 0;      /* IRIC = 0 */
                    IIC0.ICCR.BIT.IEIC = 1;      /* IEIC = 1 (IICI0 interrupt enable) */
```

```
                set_imask_ccr(0);                   /* Interrupt enable */

                while(rxedf == 0x00);               /* rxedf != 0 ? */

                IIC0.ICMR.BIT.WAIT = 1;             /* WAIT = 1 */
                IIC0.ICSR.BIT.ACKB = 1;             /* ACKB = 1 */

                dt_rec_ram[8] = IIC0.ICDR;          /* 9th receive data read */
                IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);     /* Receieve end (IRIC=1) ? */

                IIC0.ICCR.BIT.TRS = 1;              /* Master transmit mode set */

                IIC0.ICCR.BIT.IRIC = 0;             /* 9th clock transmit (IRIC=0) */
                while(IIC0.ICCR.BIT.IRIC == 0);     /* 9th clock transmit end (IRIC=1) ? */

                dt_rec_ram[9] = IIC0.ICDR;          /* 10th (last) receive data read */

                IIC0.ICSR.BIT.ACKB = 0;             /* ACKB = 0 */
                IIC0.ICMR.BIT.WAIT = 0;             /* WAIT = 0 */
            }
        }
    }
    set_stop();                                     /* Stop condition set */

    while(1);                                       /* End */
}

/*****************************************************
*  initialize : RAM & IIC0 Initialize                *
*****************************************************/
void initialize(void)
{
    unsigned char i;                                /* Receive data counter */

    for(i=0; i<10; i++)                             /* Receive data store area initialize */
    {
```

```
        dt_rec_ram[i] = 0x00;
    }

    rxedf = 0x00;                               /* Receive end flag initialize */

    STCR.BYTE = 0x00;                           /* FLSHE = 0 */

    MSTPCR.BYTE.H = 0x3f;                       /* DTC module stop mode reset */
    SAR1 = 0x0000ffde;                          /* SAR = H'00FFDE (ICDR0) */
    MRA1 = 0x20;                                /* MRA = H'20 */
    DAR1 = 0x0000e100;                          /* DAR = H'00E100 */
    MRB1 = 0x00;                                /* MRB = H'00 */
    CRA1 = 0x0009;                              /* CRA = H'0009 */
    CRB1 = 0x0000;                              /* CRB = H'0000 */
    DTC.VECR.BYTE = 0x00;                       /* SWDTE = 0, DTVEC = H'00 */
    DTC.ED.BIT.B4 = 1;                          /* DTCED4 = 1 */

    MSTPCR.BYTE.L = 0x7f;                       /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                       /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    MSTPCR.BYTE.L = 0xef;                       /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                           /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                         /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                      /* ICE = 0 */
    IIC0.SAR.BYTE = 0x00;                       /* FS = 0 */
    IIC0.SARX.BYTE = 0x01;                      /* FSX = 1 */
    IIC0.ICCR.BYTE = 0x81;                      /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                      /* ACKB = 0 */
    STCR.BYTE = 0x30;                           /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                      /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                      /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
void set_start(void)
```

```
{
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    IIC0.ICCR.BYTE = 0xbc;                          /* Start condition set (BBSY=1,SCP=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Start condition set (IRIC=1) ? */
}

/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
    IIC0.ICCR.BYTE = 0xb8;                          /* Stop condition set (BBSY=0,SCP=0) */
    while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
}

/*****************************************************
*  trs_slvadr_a0 : Slave address + W data transmit   *
*****************************************************/
void trs_slvadr_a0(void)
{
    IIC0.ICDR = 0xa0;                               /* Slave address + W data(H'A0) write */
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_slvadr_a1 : Slave address + R data transmit   *
*****************************************************/
void trs_slvadr_a1(void)
{
    IIC0.ICDR = 0xa1;                               /* Slave address + R data(H'A1) write */
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}
```

```
/*****************************************************
*  trs_memadr : EEPROM memory address data transmit  *
*****************************************************/
void trs_memadr(void)
{
     IIC0.ICDR = 0x00;                              /* EEPROM memory address data(H'00) write */
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                /* Transmit end (IRIC=1) ? */
}


/*****************************************************
*  iici0 : IIC0 interrupt routine                    *
*****************************************************/
#pragma interrupt(iici0)
void iici0(void)
{
     IIC0.ICCR.BIT.IRIC = 0;                        /* IRIC = 0 */
     IIC0.ICCR.BIT.IEIC = 0;                        /* IEIC = 0 (IICI0 interrupt disable) */
     rxedf++;                                       /* rxedf flag set */
}
```

4.8 Slave Transmission

4.8.1 Specifications

- Channel 0 of the I²C bus interface is used to transmit, from one H8S/2138 in the slave-transmission mode, 10 bytes of data to the master H8S/2138.
- The slave address of the H8S/2138 that acts as the slave transmitter is [0011100].
- The data to be transmitted is H'00, H'11, H'22, H'33, H'44, H'55, H'66, H'77, H'88, and H'99.
- The connection of devices to the I²C bus in this system is in the single-master configuration: there is one master device (H8S/2138) and one slave device (H8S/2138).
- The frequency of the transfer clock is 100 kHz.
- Figure 4.29 shows an example of such a connection between two H8S/2138s.

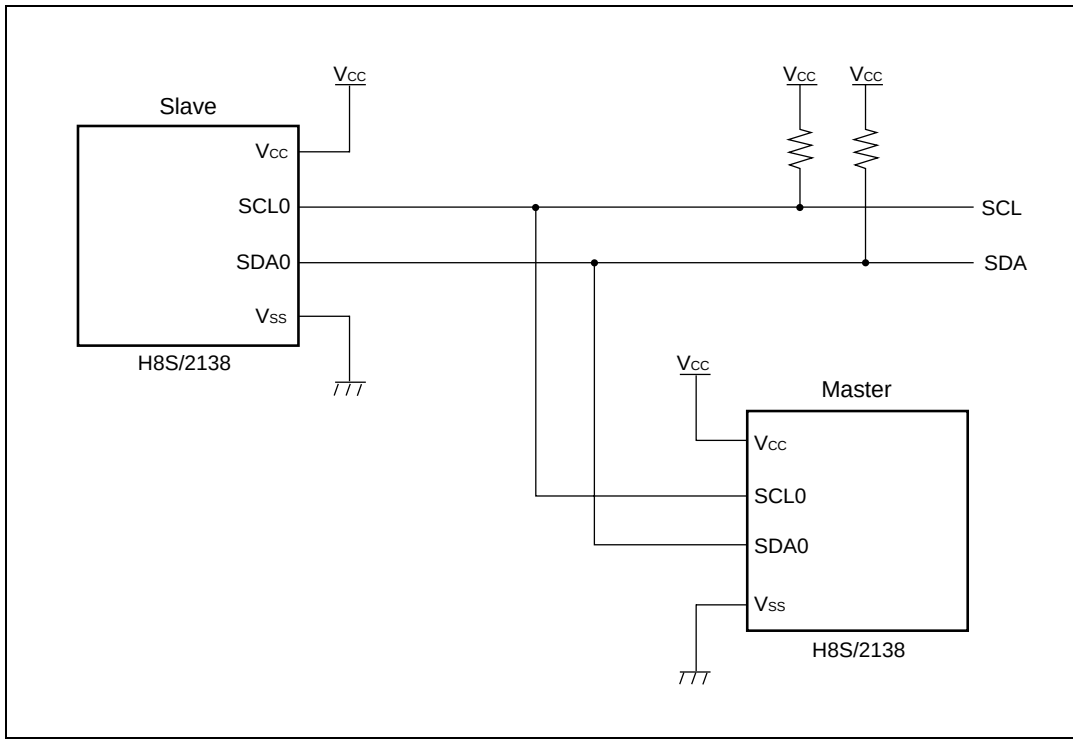


Figure 4.29 Example of Two H8S/2138s Connected in a Single-Master Configuration

- The I²C bus format used in this example of a task is shown in Fig. 4.30.

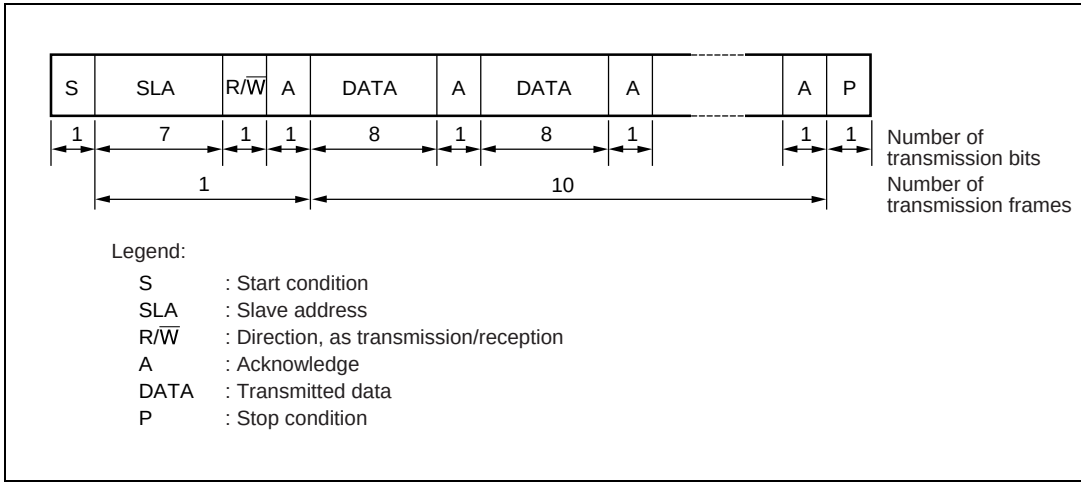
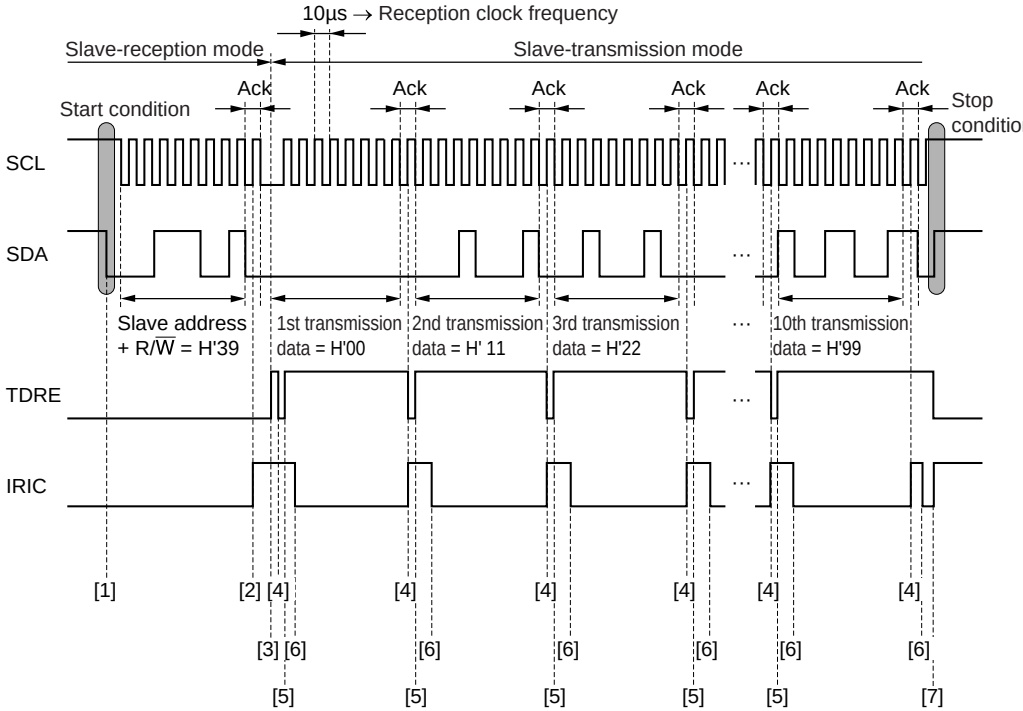


Figure 4.30 Transfer Format used in this Example of a Task

4.8.2 Description of Operation

Figure 4.31 shows this example's principle of operation.



	Software processing	Hardware processing
[1]	No operation	No operation
[2]	No operation	IRIC = 1 (at rising of 9th clock)
[3]	No operation	TDRE =1 (TRS = 0 → TRS = 1)
[4]	Writes the data for transmission to ICDR0	TDRE = 0 (writes data to ICDRT with TRS being 1)
[5]	No operation	TDRE = 1 (transfers data from ICDRT to ICDRS)
[6]	Clear IRIC to 0.	No operation
[7]	Clear IRIC to 0.	IRIC=1, TDRE=0 (detects stop condition from the bus line state)

Figure 4.31 Slave Transmission: Principle of Operation

4.8.3 Description of Software

(1) Description of Modules

Table 4.27 describes the details of the modules used in this example of a task.

Table 4.27 Description of Modules

Name	Label	Function
Main routine	main	Sets stack pointers and the MCU mode, and enables an interrupt.
Initial settings	initialize	Makes initial settings of IIC0.
Slave transmission	slv_trs	Uses slave transmission to transmit 10 bytes of data to the other H8S/2138.

(2) Description of On-chip Registers

Table 4.28 describes the usage of on-chip registers in this example of a task.

Table 4.28 On-chip Registers

Register		Function	Address	Setting
ICDR0		Stores the data for transmission.	H'FFDE	—
SAR0	FS	Along with the settings in the FSX bit of SARX0 and the SW bit of DDCSWR, sets the format for transfer.	H'FFDF bit0	0
	SVA6 to SVA0	Hold the slave address of the slave H8S/2138.	H'FFDF bit7 to bit1	SVA6=0 SVA5=0 SVA4=1 SVA3=1 SVA2=1 SVA1=0 SVA0=0
SARX0	FSX	Along with the settings in the FS bit of SAR0 and the SW bit of DDSWR, sets the format for transfer.	H'FFDE bit0	1

Table 4.28 On-chip Registers (Continued)

Register		Function	Address	Setting
ICMR0	MLS	Sets data transfer as MSB first.	H'FFDF bit7	0
	WAIT	Sets continuous transfer of data and acknowledge bits.	H'FFDF bit6	0
	CKS2 to CKS0	Along with the setting in the IICX0 bit of STCR, set the frequency of the transfer clock to 100 kHz.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set the number of bits for the next transfer in the I^2C bus format to 9 (9 bits/frame).	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Controls access to the ICMR0, ICDR0/SAR, and SARX registers, and selects the operation (the port function for the SCL0/SDA0 pin) or non-operation (bus-drive state for the SCL/SDA pin) of the I^2C bus interface.	H'FFD8 bit7	0/1
	IEIC	Disables the generation of interrupt requests by the I^2C bus interface.	H'FFD8 bit6	0
ICCR0	MST	Uses the I^2C bus interface in its slave mode.	H'FFD8 bit5	1
	TRS	Uses the I^2C bus interface in its transmission mode.	H'FFD8 bit4	1
	ACKE	Suspends the continuous transfer of data when the acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms whether or not the I^2C bus is occupied, and, in combination with the SCP bit, sets the start and stop conditions.	H'FFD8 bit2	0/1
	IRIC	Detects the start condition, determines the end of data transfer, and detects acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Along with the BBSY bit, sets the start/stop conditions.	H'FFD8 bit0	0
ICSR0	ACKB	Stores the acknowledgement received from the EEPROM during transmission. Sets the acknowledge bit for transmission to the EEPROM during reception.	H'FFD9 bit0	-
STCR	IICX0	Along with the settings in CKS2 to CKS0 of ICMR0, selects the frequency of the transfer clock.	H'FFC3 bit5	1
	IICE	Enables CPU access to the data and control registers of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets the control registers of the flash memory to non-selected.	H'FFC3 bit3	0

Table 4.28 On-chip Registers (Continued)

Register		Function	Address	Setting
DDCSWR	SWE	Prohibits automatic change from format-less transfer to transfer in the I^2C bus format on the channel 0 I^2C interface.	H'FEE6 bit7	0
	SW	Uses the channel 0 I^2C interface in the I^2C bus format.	H'FEE6 bit6	0
	IE	Prohibits interrupts during automatic changes of format.	H'FEE6 bit5	0
	CLR3 to CLR0	Control the initialization of the internal state of the I^2C interface	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels the module-stopped mode for SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels the module stopped mode for I^2C channel 0.	H'FF87 bit4	0
SCR0	CKE1, 0	Make the I/O port setting for the P52/SCK0/SCL0 pin.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets the mode for SCI transfer on channel 0 as asynchronous.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set the interrupt control mode of the interrupt controller to 1-bit control.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set the MCU's operating mode to mode 3 by latching the input levels on the MD1 and 0 pins.	H'FFC5 bit1, 0	MDS1=1 MDS0=1

(3) Description of Variables

Table 4.29 describes the variables used in this task.

Table 4.29 Description of Variables

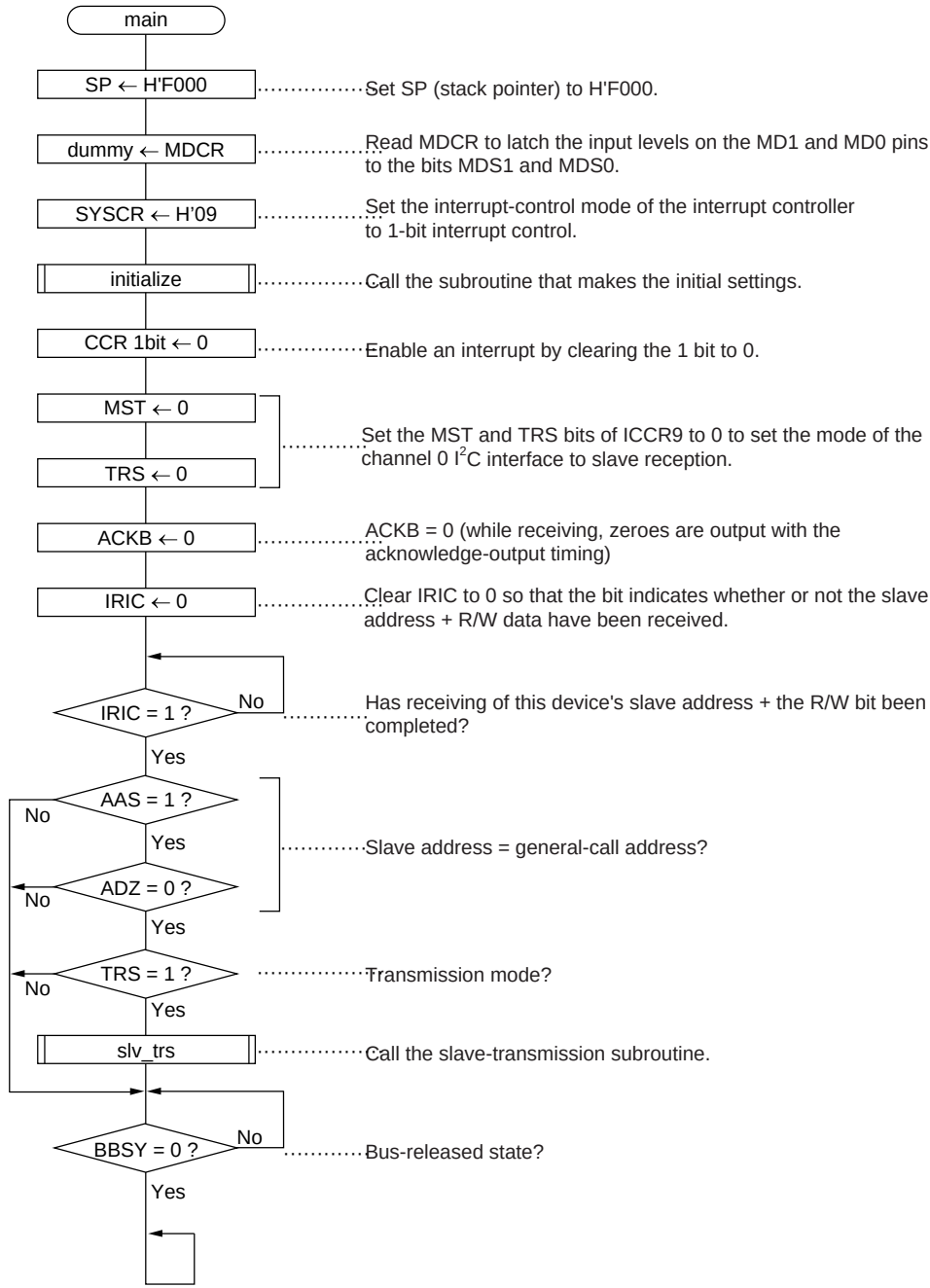
Variable	Function	Size	Initial Value	Module Name
dt_trs[0]	Stores first byte of data for transmission.	1 byte	H'00	slv_trs
dt_trs[1]	Stores second byte of data for transmission.	1 byte	H'11	slv_trs
dt_trs[2]	Stores third byte of data for transmission.	1 byte	H'22	slv_trs
dt_trs[3]	Stores fourth byte of data for transmission.	1 byte	H'33	slv_trs
dt_trs[4]	Stores fifth byte of data for transmission.	1 byte	H'44	slv_trs
dt_trs[5]	Stores sixth byte of data for transmission.	1 byte	H'55	slv_trs
dt_trs[6]	Stores seventh byte of data for transmission.	1 byte	H'66	slv_trs
dt_trs[7]	Stores eighth byte of data for transmission.	1 byte	H'77	slv_trs
dt_trs[8]	Stores nineth byte of data for transmission.	1 byte	H'88	slv_trs
dt_trs[9]	Stores tenth byte of data for transmission.	1 byte	H'99	slv_trs
i	Transmission data counter	1 byte	H'00	slv_trs
dummy	Stores the MDCR value.	1 byte	—	main
dmyrd	Storage for the value obtained by the dummy read.	1 byte	—	slv_trs

(4) Description of RAM Usage

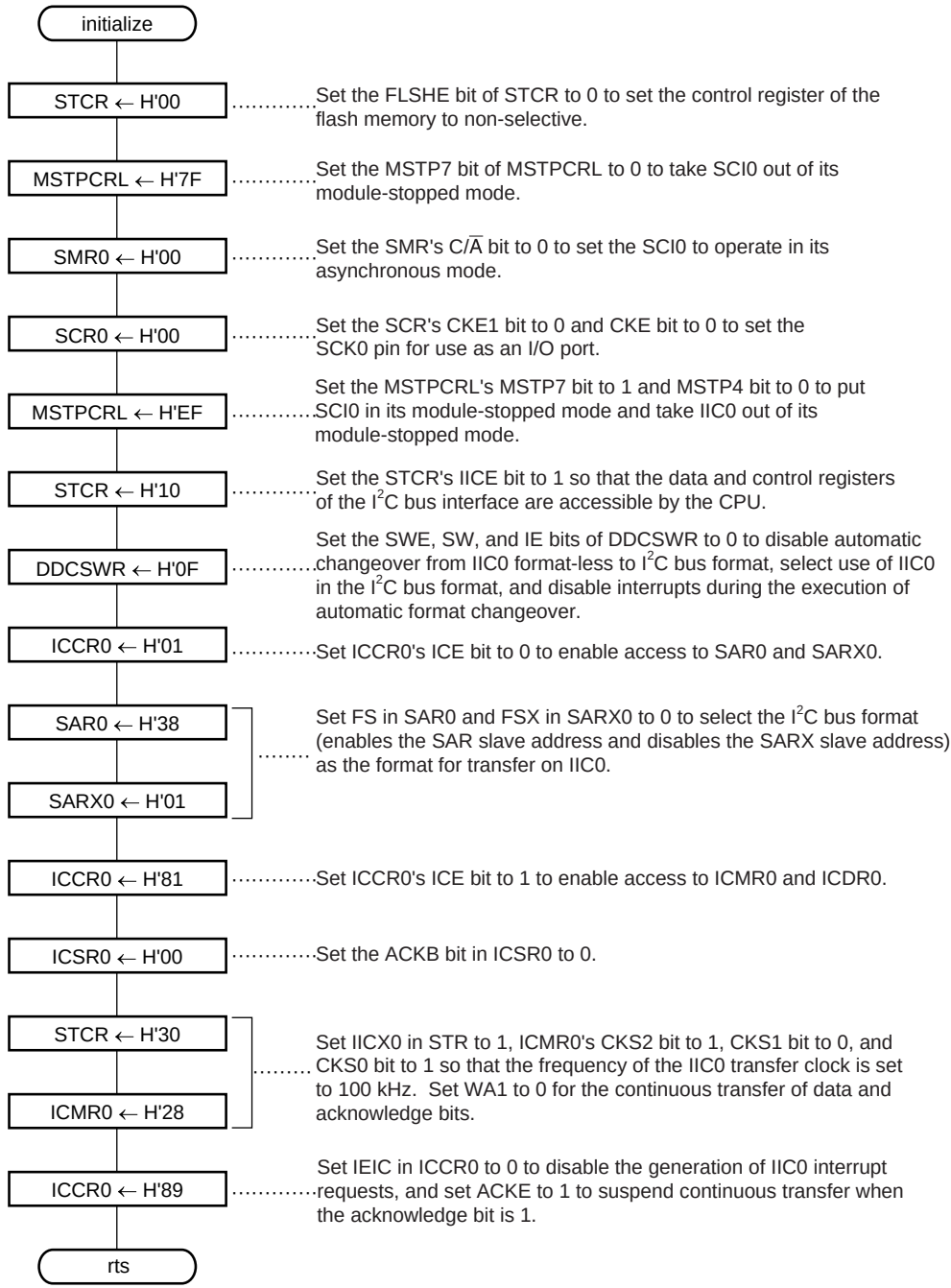
In this example of a task, the only RAM used is that required for the variables.

4.8.4 Flowcharts

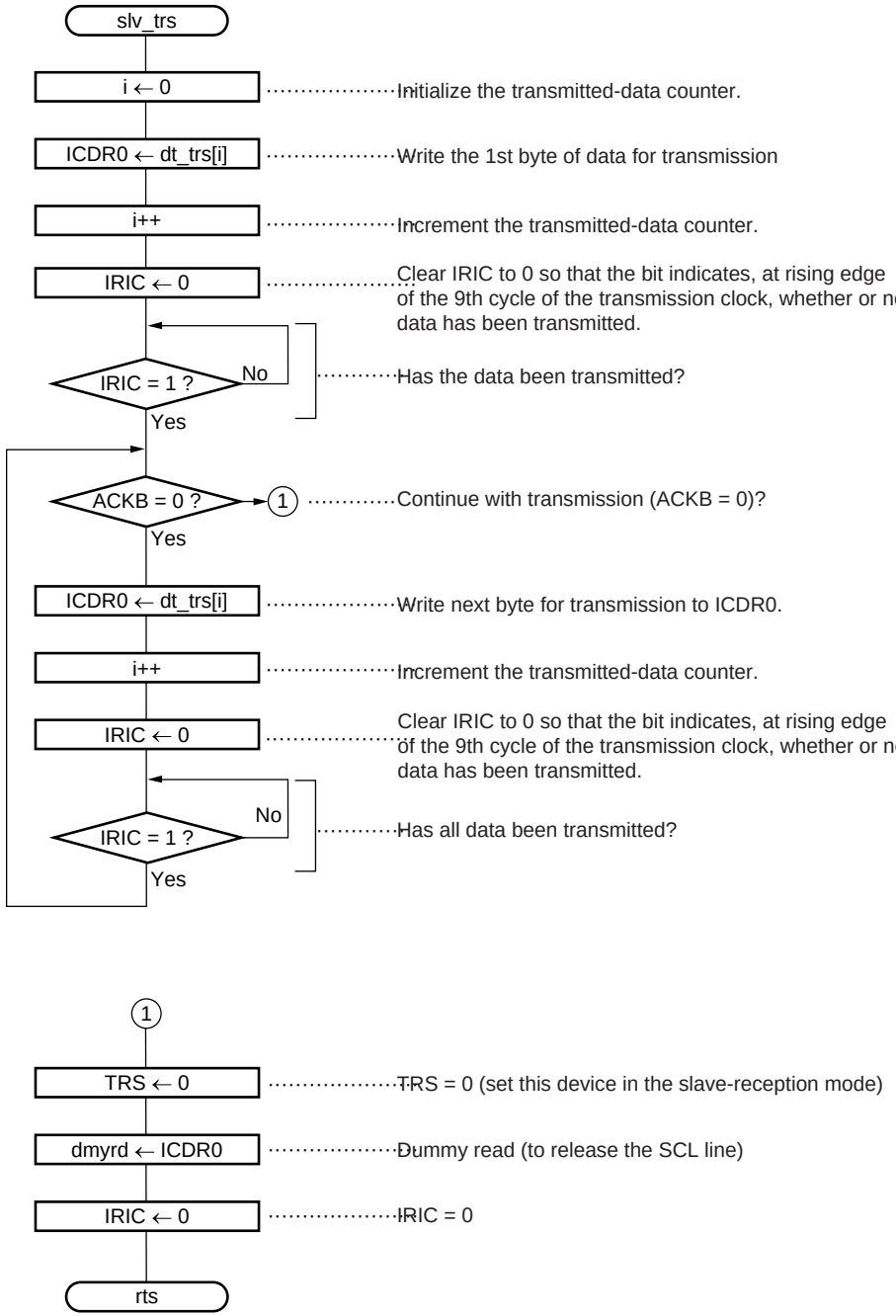
(1) Main Routine



(2) Subroutine for Making Initial Settings



(3) Slave Transmission Subroutine



4.8.5 Program List

```
/*****************************************************
*  H8S/2138 IIC bus application note                 *
*    7.Slave transmit to H8S/2138                    *
*                       File name   :   SVTxd.c       *
*                       Fai         :   20MHz         *
*                       Mode        :   3             *
*****************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*****************************************************
*  Prototype                                         *
*****************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* IIC0 initialize */
void slv_trs(void);                                 /* Slave transmit to H8S/2138 */

/*****************************************************
*  Data table                                        *
*****************************************************/
const unsigned char dt_trs[10] =
{
    0x00,                                           /* 1st transmit data */
    0x11,                                           /* 2nd transmit data */
    0x22,                                           /* 3rd transmit data */
    0x33,                                           /* 4th transmit data */
    0x44,                                           /* 5th transmit data */
    0x55,                                           /* 6th transmit data */
    0x66,                                           /* 7th transmit data */
    0x77,                                           /* 8th transmit data */
    0x88,                                           /* 9th transmit data */
    0x99                                            /* 10th transmit data */
};
```

```
/*****************************************************
*  main : Main routine                                *
*****************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                          ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                              /* MCU mode set */

     SYSCR.BYTE = 0x09;                              /* Interrupt control mode set */
     initialize();                                   /* Initialize */

     set_imask_ccr(0);                               /* Interrupt enable */

     IIC0.ICCR.BIT.MST = 0;                          /* Slave receive mode set */
     IIC0.ICCR.BIT.TRS = 0;                          /* MST = 0, TRS = 0 */
     IIC0.ICSR.BIT.ACKB = 0;                         /* ACKB = 0 */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Receive end (IRIC=1) ? */

     if(IIC0.ICSR.BIT.AAS == 1)                      /* General call address receive ? */
     {
          if(IIC0.ICSR.BIT.ADZ == 0)
          {
               if(IIC0.ICCR.BIT.TRS == 1)            /* Transmit mode (TRS=1) ? */
               {
                    slv_trs();                       /* Slave transmit */
               }
          }
     }
     while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
     while(1);/* End */
}
```

```
/*****************************************************
*  initialize : IIC0 Initialize                      *
*****************************************************/
void initialize(void)
{
     STCR.BYTE = 0x00;                                /* FLSHE = 0 */
     MSTPCR.BYTE.L = 0x7f;                            /* SCI0 module stop mode reset */
     SCI0.SMR.BYTE = 0x00;                            /* SCL0 pin function set */
     SCI0.SCR.BYTE = 0x00;
     MSTPCR.BYTE.L = 0xef;                            /* IIC0 module stop mode reset */
     STCR.BYTE = 0x10;                                /* IICE = 1 */
     DDCSWR.BYTE = 0x0f;                              /* IIC bus format initialize */
     IIC0.ICCR.BYTE = 0x01;                           /* ICE = 0 */
     IIC0.SAR.BYTE = 0x38;                            /* FS = 0 */
     IIC0.SARX.BYTE = 0x01;                           /* FSX = 1 */
     IIC0.ICCR.BYTE = 0x81;                           /* ICE = 1 */
     IIC0.ICSR.BYTE = 0x00;                           /* ACKB = 0 */
     STCR.BYTE = 0x30;                                /* IICX0 = 1 */
     IIC0.ICMR.BYTE = 0x28;                           /* Transfer rate = 100kHz */
     IIC0.ICCR.BYTE = 0x89;                           /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  slv_trs : Slave transmit to H8S/2138              *
*****************************************************/
void slv_trs(void)
{
     unsigned char i = 0;                             /* Transmit data counter initialize */
     unsigned char dmyrd;                             /* Dummy read data store */

     IIC0.ICDR = dt_trs[i++];                         /* 1st transmit data write */
     IIC0.ICCR.BIT.IRIC = 0;                          /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                  /* Transmit end (IRIC=1) ? */

     while(IIC0.ICSR.BIT.ACKB == 0)                   /* Transmit continue (ACKB=0) ? */
     {
          IIC0.ICDR = dt_trs[i++];                    /* Transmit data write */
```

```
        IIC0.ICCR.BIT.IRIC = 0;                 /* IRIC = 0 */
        while(IIC0.ICCR.BIT.IRIC == 0);         /* Transmit end (IRIC=1) ? */
    }

    IIC0.ICCR.BIT.TRS = 0;                      /* Slave receive mode set (MST=0,TRS=0) */
    dmyrd = IIC0.ICDR;                          /* Dummy read */
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
}
```

4.9 Slave Reception

4.9.1 Specifications

- One H8S/2138 in slave receive mode receives, through channel 0 of its I^2C bus interface, 10 bytes of data from another H8S/2138.
- The slave address of the H8S/2138 that acts as the slave receiver is [0011100].
- The connection of devices to the I^2C bus in this system is in the single-master configuration: there is one master device (H8S/2138) and one slave device (H8S/2138).
- The frequency of the transfer clock is 100 kHz.
- The slave receiver uses the output of its acknowledge bit to control the number of bytes it receives.
- Figure 4.32 shows an example of such a connection between two H8S/2138s.

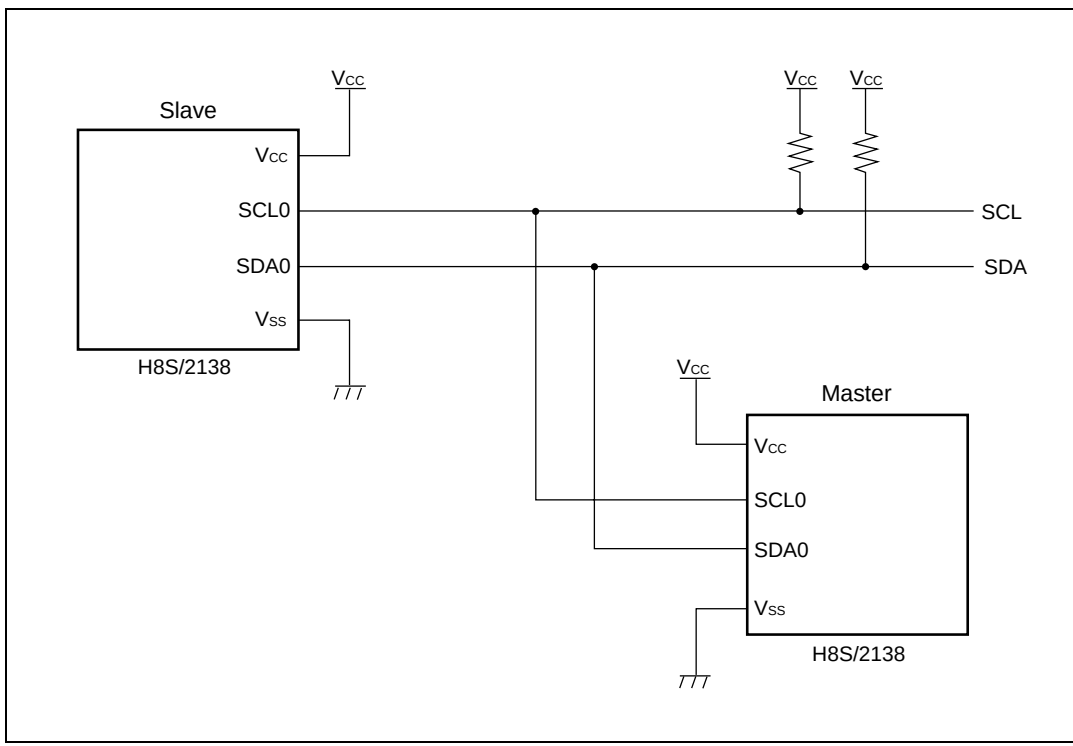


Figure 4.32 Example of Two H8S/2138s Connected in a Single-Master Configuration

- The I^2C bus format used in this example of a task is shown in Fig. 4.33.

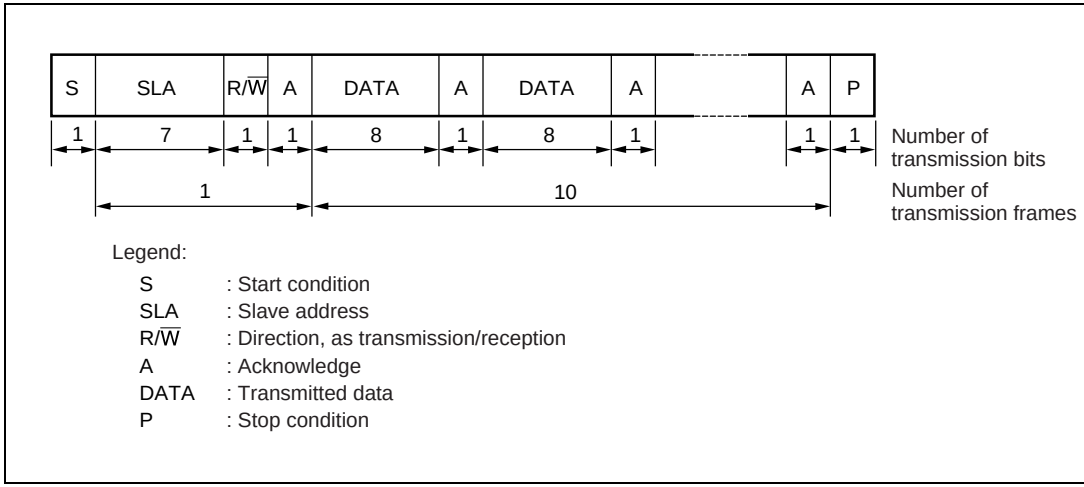
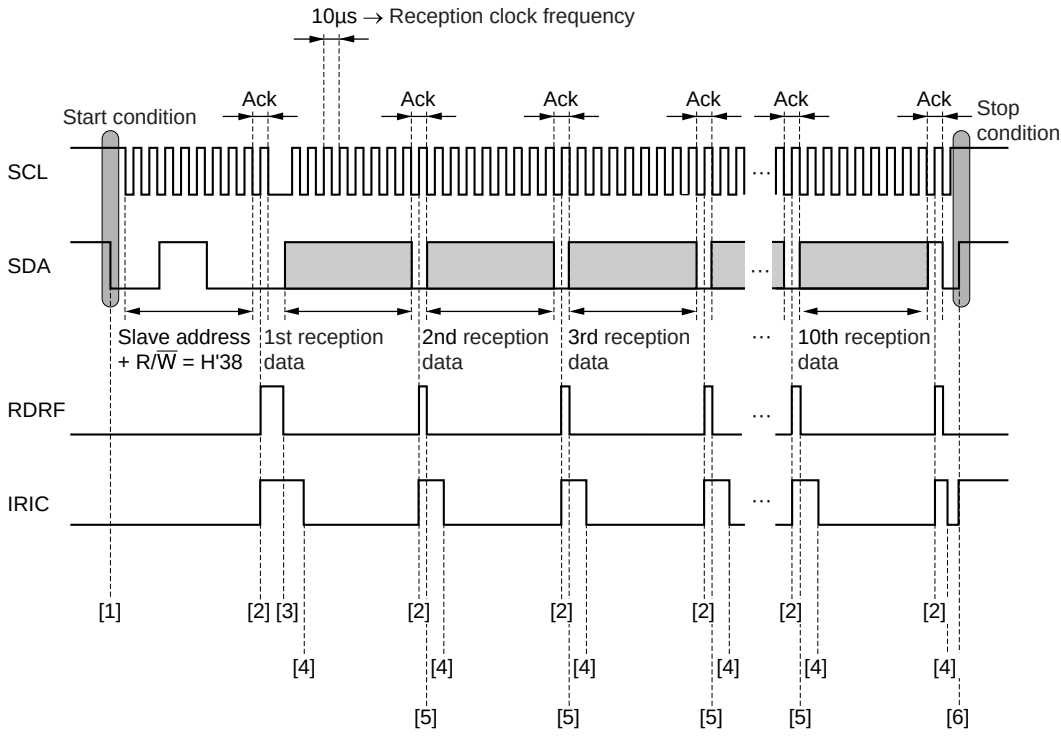


Figure 4.33 Transfer Format Used in this Example of a Task

4.9.2 Description of Operation

Figure 4.34 shows this example's principle of operation.



	Software processing	Hardware processing
[1]	No operation	No operation
[2]	No operation	RDRF = 1, IRIC = 1 (at rising of 9th clock)
[3]	Dummy read of ICDR0 (start of the receive operation)	RDRF = 0
[4]	Clears IRIC to 0.	No operation
[5]	Reads the receive data from ICDR0.	RDRF = 0
[6]	No operation	IRIC = 1 (detects stop condition from the bus line state)

Figure 4.34 Slave Reception: Principle of Operation

4.9.3 Description of Software

(1) Descriptions of modules

Table 4.30 describes the functions of the modules used in this example of a task.

Table 4.30 Descriptions of Modules

Name	Label	Function
Main Routine	main	Sets stack pointers and the MCU mode, and enables an interrupt.
Initial Settings	initialize	Sets the RAM area to be used and makes initial settings of IIC0.
Slave reception	slv_rec	Uses slave reception to receive 10 bytes of data from the other H8S/2138.

(2) Description of On-chip Registers

Table 4.31 describes the usage of on-chip registers in this example of a task.

Table 4.31 On-chip Registers

Register		Function	Address	Setting
ICDR0		Stores the received data.	H'FFDE	—
SAR0	SVA6 to SVA0	Hold the slave address of the slave H8S/2138.	H'FFDF bit7 to bit1	SVA6=0 SVA5=0 SVA4=1 SVA3=1 SVA2=1 SVA1=0 SVA0=0
	FS	Along with the settings in the FSX bit of SARX0 and the SW bit of DDCSWR, sets the format for transfer.	H'FFDF bit0	0
SARX0	FSX	Along with the settings in the FS bit of SAR0 and the SW bit of DDSWR, sets the format for transfer.	H'FFDE bit0	1

Table 4.31 On-chip Registers (Continued)

Register		Function	Address	Setting
ICMR0	MLS	Sets data transfer as MSB first.	H'FFDF bit7	0
	WAIT	Sets continuous transfer of data and acknowledge bits.	H'FFDF bit6	0
	CKS2 to CKS0	Along with the setting in the IICX0 bit of STCR, set the frequency of the transfer clock to 100 kHz.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set the number of bits for the next transfer in the I^2C bus format to 9 (9 bits/frame).	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Controls access to the ICMR0, ICDR0/SAR, and SARX registers, and selects the operation (the port function for the SCL0/SDA0 pin) or non-operation (bus-drive state for the SCL/SDA pin) of the I^2C bus interface.	H'FFD8 bit7	0/1
	IEIC	Disables the generation of interrupt requests by the I^2C bus interface.	H'FFD8 bit6	0
ICCR0	MST	Uses the I^2C bus interface in its slave mode.	H'FFD8 bit5	1
	TRS	Uses the I^2C bus interface in its reception mode.	H'FFD8 bit4	1
	ACKE	Suspends the continuous transfer of data when the acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms whether or not the I^2C bus is occupied, and, in combination with the SCP bit, sets the start and stop conditions.	H'FFD8 bit2	0/1
	IRIC	Detects the start condition, determines the end of data transfer, and detects acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Along with the BBSY bit, sets the start/stop conditions.	H'FFD8 bit0	0
ICSR0	ACKB	Stores the data that has, in accordance with the timing of the output of the slave device's acknowledge bit, been output by the other device.	H'FFD9 bit0	0/1
STCR	IICX0	Along with the settings in CKS2 to CKS0 of ICMR0, selects the frequency of the transfer clock.	H'FFC3 bit5	1
	IICE	Enables CPU access to the data and control registers of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets the control registers of the flash memory to non-selected.	H'FFC3 bit3	0

Table 4.31 On-chip Registers (Continued)

Register		Function	Address	Setting
DDCSWR	SWE	Prohibits automatic change from format-less transfer to transfer in the I²C bus format on the channel 0 I²C interface.	H'FEE6 bit7	0
	SW	Uses the channel 0 I²C interface in the I²C bus format.	H'FEE6 bit6	0
	IE	Prohibits interrupts during automatic changes of format.	H'FEE6 bit5	0
	CLR3 to CLR0	Control the initialization of the internal state of the I²C interface	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels the module-stopped mode for SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels the module stopped mode for I²C channel 0.	H'FF87 bit4	0
SCR0	CKE1, 0	Make the I/O port setting for the P52/SCK0/SCL0 pin.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets the mode for SCI transfer on channel 0 as asynchronous.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set the interrupt control mode of the interrupt controller to 1-bit control.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set the MCU's operating mode to mode 3 by latching the input levels on the MD1 and 0 pins.	H'FFC5 bit1, 0	MDS1=1 MDS0=1

(3) Description of Variables

Table 4.32 describes the variables used in this task example.

Table 4.32 Description of Variables

Variables	Function	Size	Initial Value	Name of Using Module
i	Received data counter	1 byte	H'00	initialize
dummy	Value read from MDCR	1 byte	—	main
dmyrd	ICDR0 dummy-read value	1 byte	—	slv_trs

(4) Description of RAM Usage

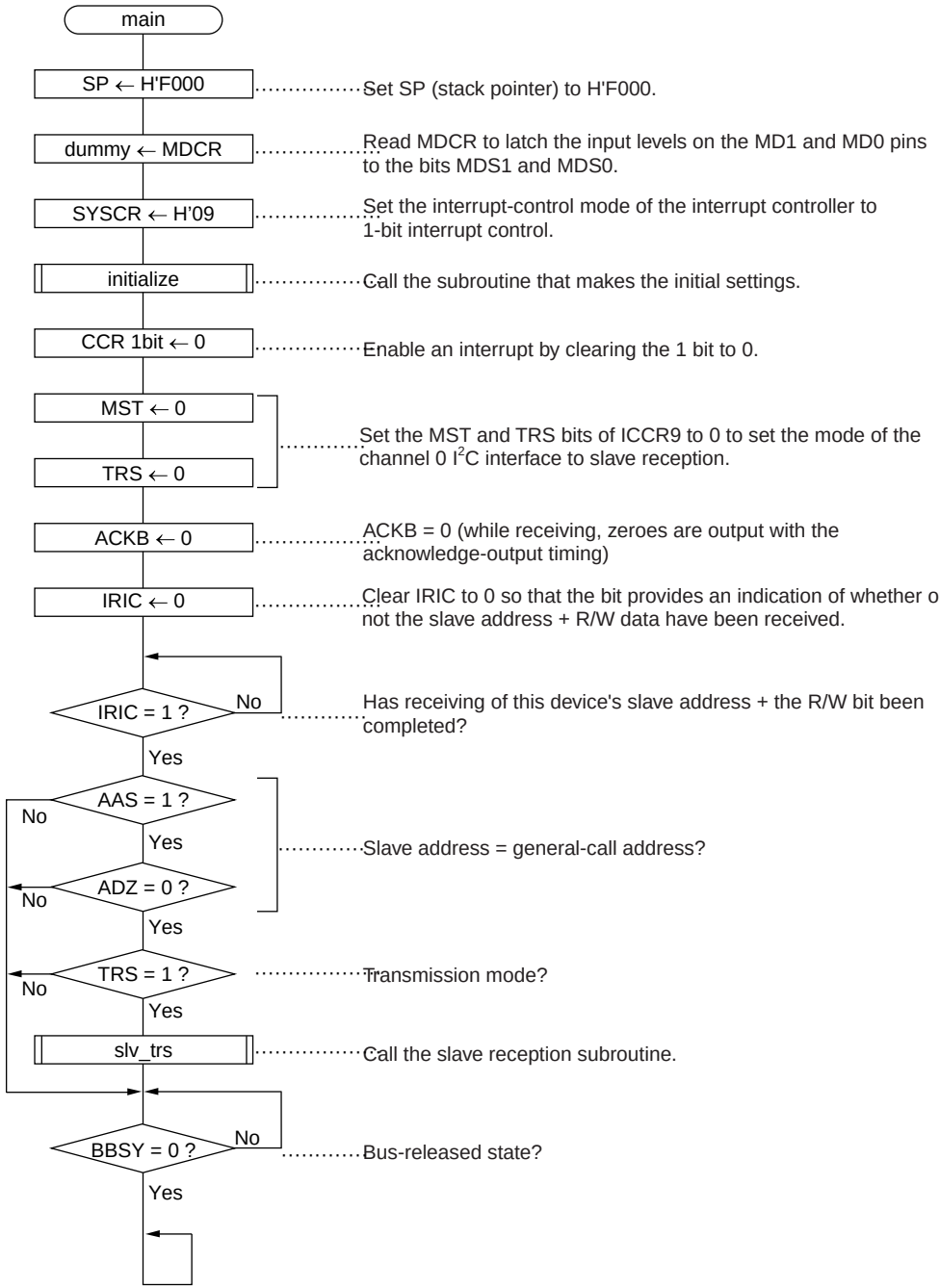
Table 4.33 describes the RAM used in this task example.

Table 4.33 Description of RAM Usage

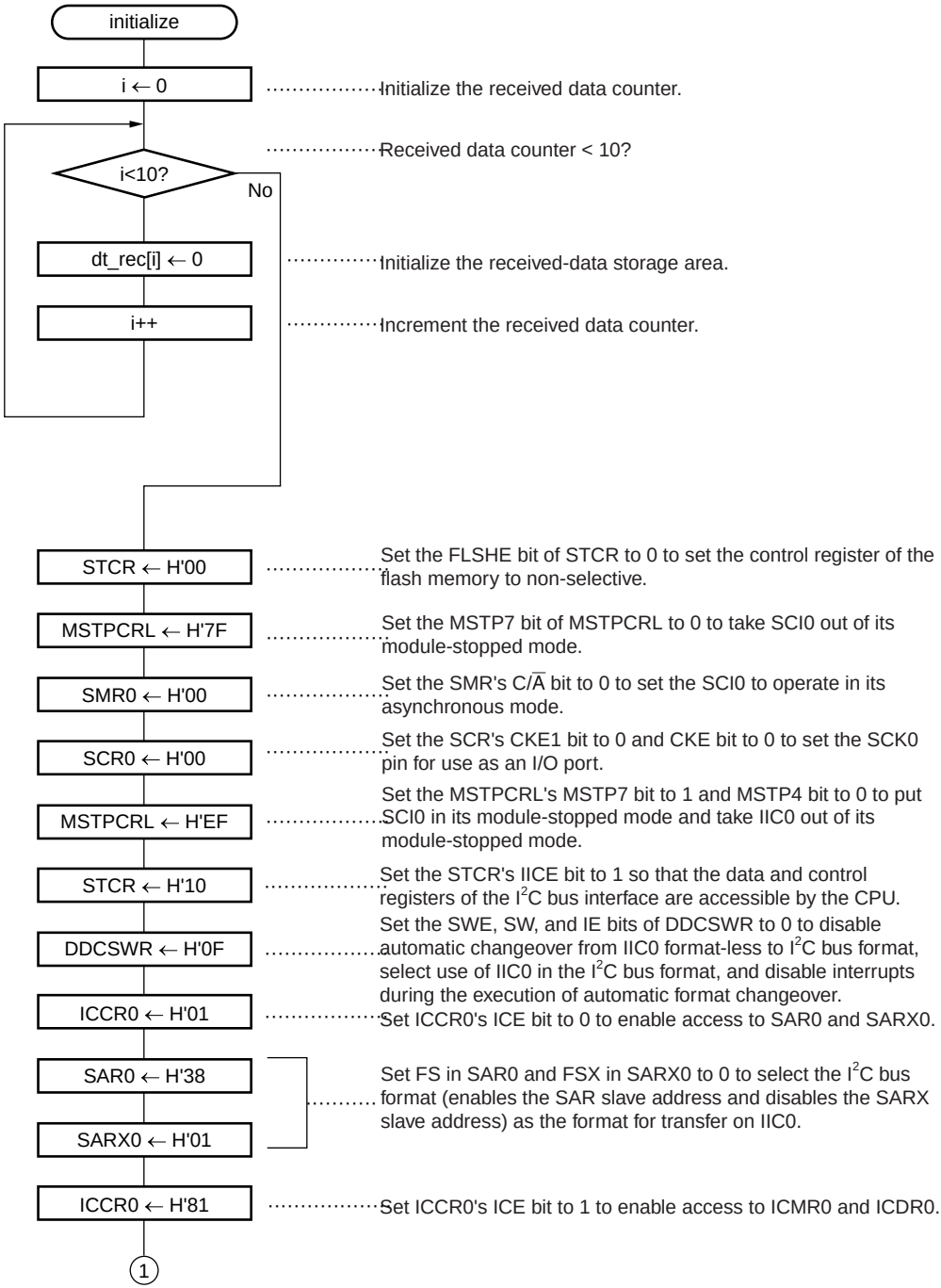
Label	Function	Size	Address	Name of Using Module
dt_rec[i]	Stores the received data.	10 bytes	H'E100 to H'E109	initialize slv_rec

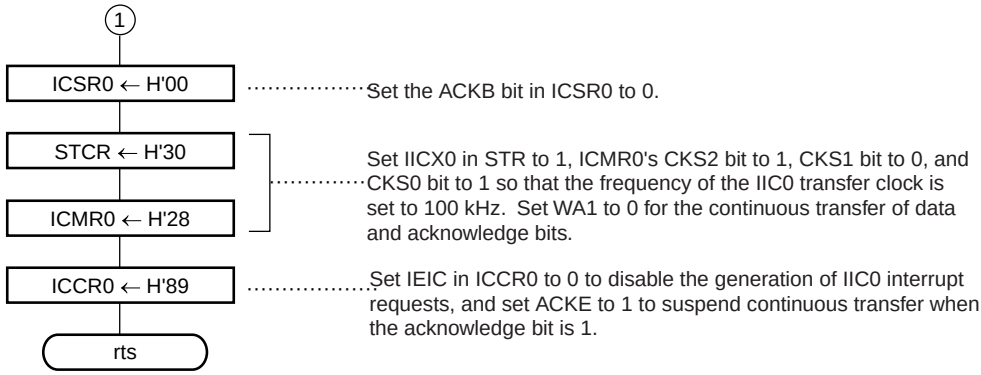
4.9.4 Flowcharts

(1) Main Routine

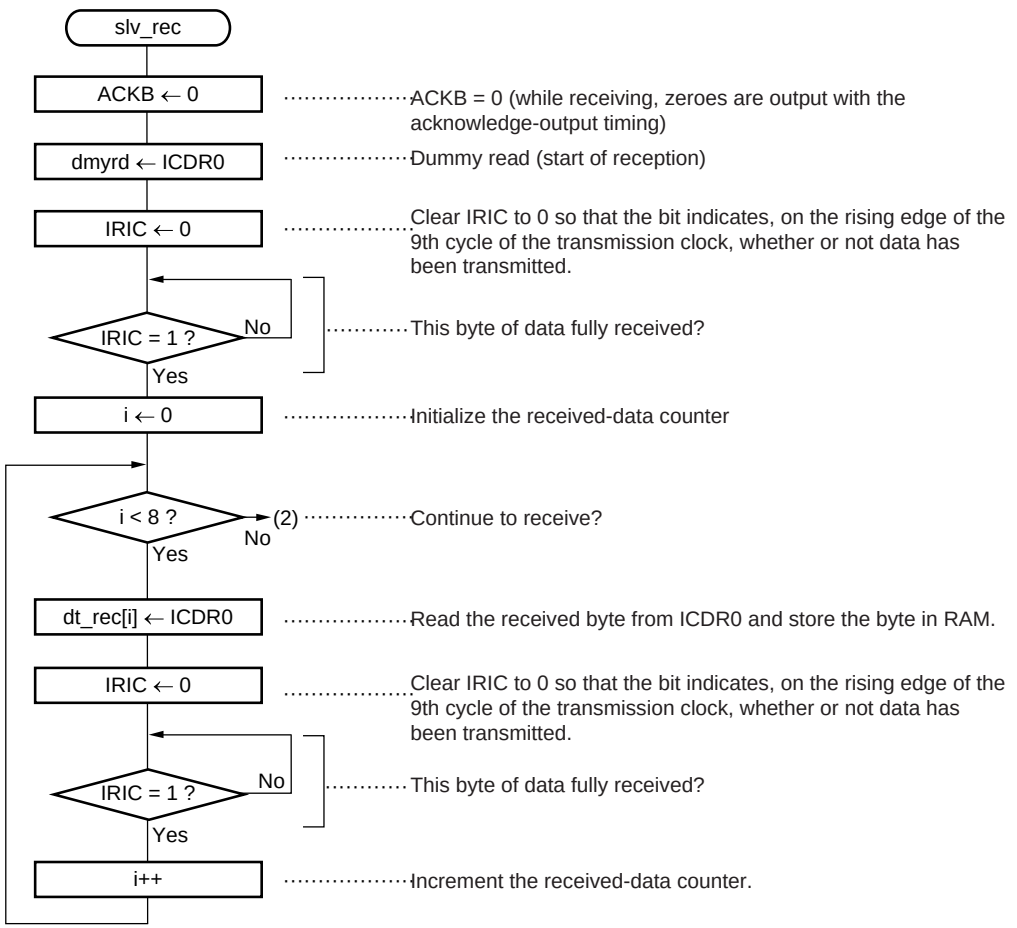


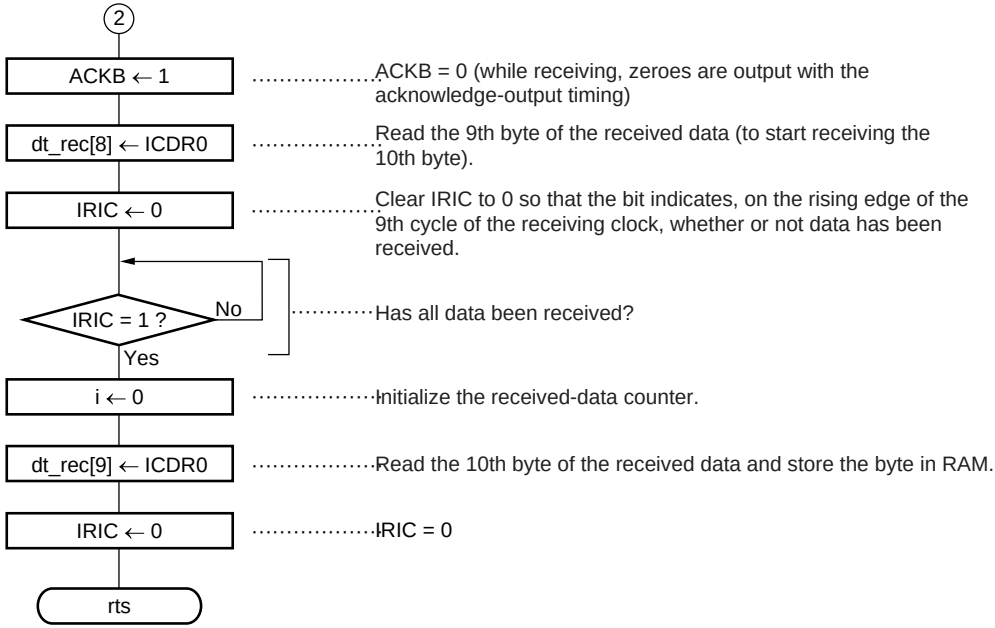
(2) Subroutine for Making Initial Settings





(3) Slave Reception Subroutine




2
ACKB ← 1
ACKB = 0 (while receiving, zeroes are output with the acknowledge-output timing)
dt_rec[8] ← ICDR0
Read the 9th byte of the received data (to start receiving the 10th byte).
IRIC ← 0
Clear IRIC to 0 so that the bit indicates, on the rising edge of the 9th cycle of the receiving clock, whether or not data has been received.
IRIC = 1 ?
No
Yes
Has all data been received?
i ← 0
Initialize the received-data counter.
dt_rec[9] ← ICDR0
Read the 10th byte of the received data and store the byte in RAM.
IRIC ← 0
IRIC = 0
rts

4.9.5 Program List

```
/******************************************************
*  H8S/2138 IIC bus application note                  *
*    8.Slave receive from H8S/2138                    *
*                       File name   :   SVRxd.c       *
*                       Fai         :   20MHz         *
*                       Mode        :   3             *
******************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/******************************************************
*  Prototype                                          *
******************************************************/
void main(void);                                      /* Main routine */
void initialize(void);                                /* RAM & IIC0 initialize */
void slv_rec(void);                                   /* Slave transmit to H8S/2138 */

/******************************************************
*  RAM allocation                                     *
******************************************************/
#pragma section ramarea
unsigned char dt_rec[10];                             /* Receive data store area */
#pragma section

/******************************************************
*  main : Main routine                                *
******************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                           ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                               /* MCU mode set */
```

```
    SYSCR.BYTE = 0x09;                          /* Interrupt control mode set */
    initialize();                               /* Initialize */

    set_imask_ccr(0);                           /* Interrupt enable */

    IIC0.ICCR.BIT.MST = 0;                      /* Slave receive mode set */
    IIC0.ICCR.BIT.TRS = 0;                      /* MST = 0, TRS = 0 */
    IIC0.ICSR.BIT.ACKB = 0;                     /* ACKB = 0 */
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Slave address receive end ? */

    if(IIC0.ICSR.BIT.AAS == 1)                  /* General call address receive ? */
    {
        if(IIC0.ICSR.BIT.ADZ == 0)
        {
            if(IIC0.ICCR.BIT.TRS == 0)          /* Slave receive mode (TRS=0) ? */
            {
                slv_rec();                      /* Slave receive */
            }
        }
    }
    while(IIC0.ICCR.BIT.BBSY == 1);             /* Bus empty (BBSY=0) ? */
    while(1);                                   /* End */
}

/*****************************************************
*  initialize : RAM & IIC0 Initialize               *
*****************************************************/
void initialize(void)
{
    unsigned char i=0;                          /* Receive data counter initialize */

    for(i=0; i<10; i++)                         /* Receive data store area initialize */
    {
        dt_rec[i] = 0x00;
    }
```

```
    STCR.BYTE = 0x00;                           /* FLSHE = 0 */
    MSTPCR.BYTE.L = 0x7f;                       /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                       /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    MSTPCR.BYTE.L = 0xef;                       /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                           /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                         /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                      /* ICE = 0 */
    IIC0.SAR.BYTE = 0x38;                       /* FS = 0 , Slave address = b'0011100*/
    IIC0.SARX.BYTE = 0x01;                      /* FSX = 1 */
    IIC0.ICCR.BYTE = 0x81;                      /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                      /* ACKB = 0 */
    STCR.BYTE = 0x30;                           /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                      /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                      /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  slv_rec : Slave receive from H8S/2138             *
*****************************************************/
void slv_rec(void)
{
    unsigned char i;                            /* Receive data counter initialize */
    unsigned char dmyrd;                        /* Dummy read data store area */

    IIC0.ICSR.BIT.ACKB = 0;                     /* ACKB = 0 */

    dmyrd = IIC0.ICDR;                          /* Dummy read (Receive start) */
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Receive end (IRIC=1) ? */

    for(i=0; i<8; i++)
    {
        dt_rec[i] = IIC0.ICDR;                  /* Receive data read */
        IIC0.ICCR.BIT.IRIC = 0;                 /* IRIC = 0 */
        while(IIC0.ICCR.BIT.IRIC == 0);         /* Receive end (IRIC=1) ? */
    }
```

```
    IC0.ICSR.BIT.ACKB = 1;                          /* ACKB = 1 */
    dt_rec[8] = IIC0.ICDR;                          /* 10th data receive start */
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Receive end (IRIC=1) ? */

    dt_rec[9] = IIC0.ICDR;                          /* 10th receive data read */
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
}
```

4.10 Example of Processing Bus Disconnection

4.10.1 Specification

- Writes 5 bytes of data to the EEPROM (HN58X2408) by having the H8S/2138 transmit, as the master device, on its channel 0 I^2C bus interface.
- The slave address of the EEPROM to be connected is [1010000]; the data is written to addresses H'00 to H'04 in the EEPROM's memory.
- The data to be written is [H'A1, H'B2, H'C3, H'D4, and H'E5].
- If the bus is disconnected during the transfer of data, the program stops the transfer clock (from the transmitting master device) is stopped 8 cycles into the transmission of the third byte, clears the ICE bit of ICCR0 to 0, and places the IIC0 module in its non-operational state (i.e., the SCL0/SDA0 pin is set to have a port function). After the period of an EEPROM write cycle has elapsed, the process of writing the 5 bytes of data to the EEPROM starts again, from the beginning.
- The devices are connected to the I^2C bus of this system in a single-master configuration with one master device (H8S/2138) and one slave device (H8S/2138).
- The transfer clock frequency is 100 kHz.
- Figure 4.35 shows an example of such a connection between a H8S/2138 and an EEPROM.

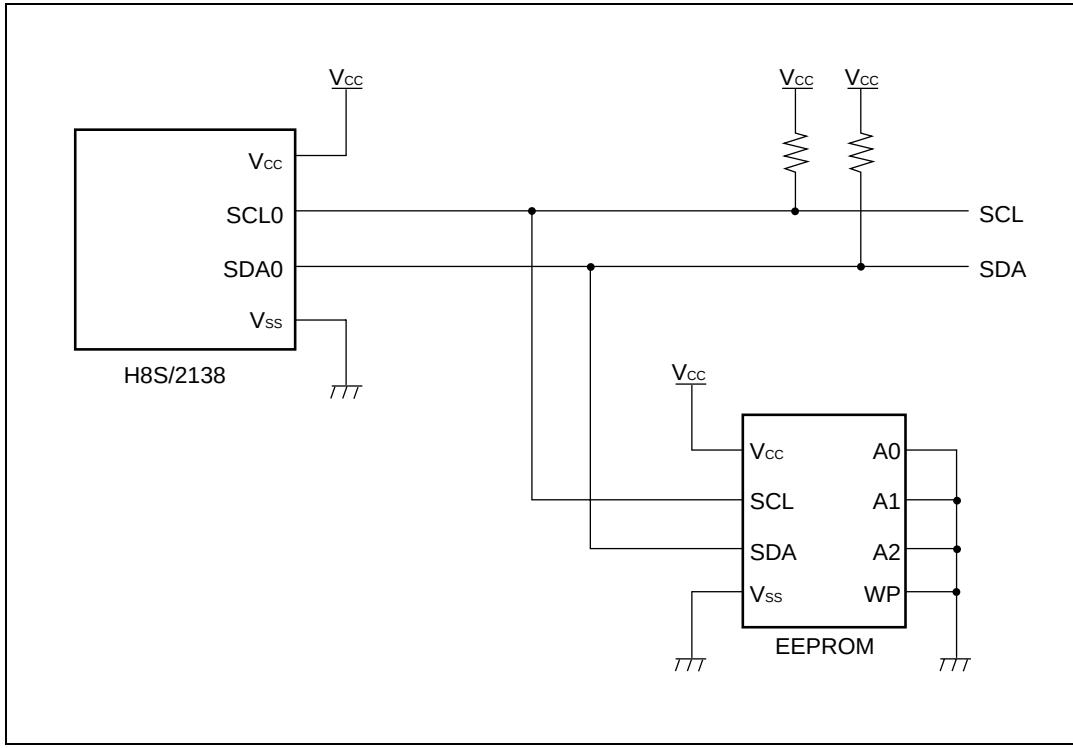
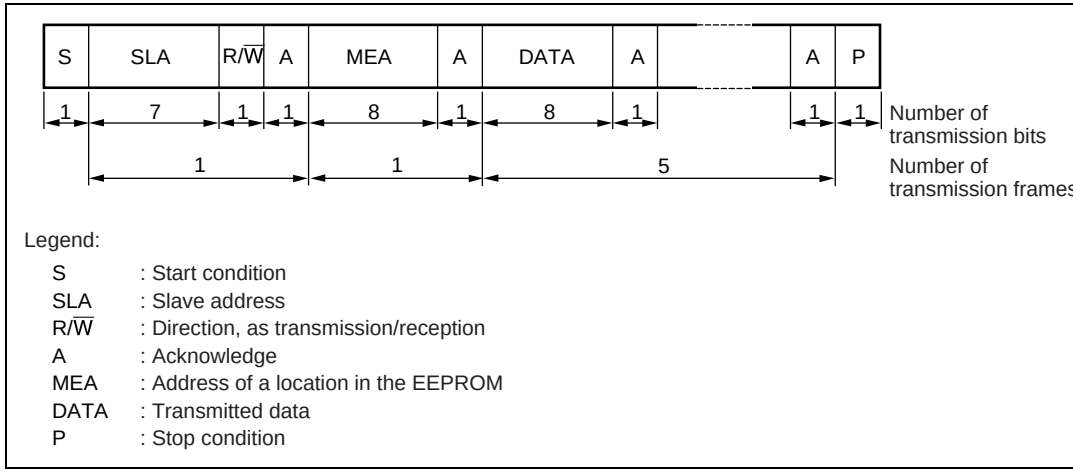


Figure 4.35 An Example of the Connection of a H8S/2138 and an EEPROM.

- The I^2C bus format used in this example of a task is shown in figure 4.36.



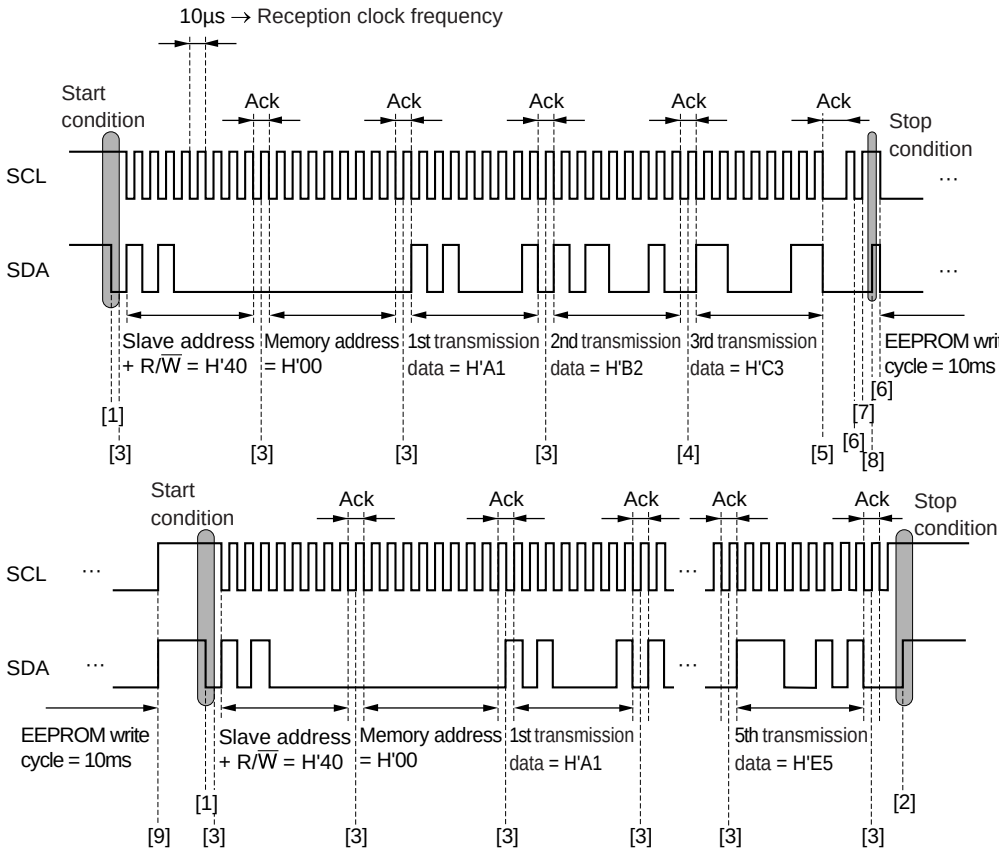
Legend:

S	: Start condition
SLA	: Slave address
R/$\overline{W}$	: Direction, as transmission/reception
A	: Acknowledge
MEA	: Address of a location in the EEPROM
DATA	: Transmitted data
P	: Stop condition

Figure 4.36 Transfer Format Used in this Example of a Task

4.10.2 Description of Operation

Figure 4.37 describes this example's principle of operation.



	Software processing
[1]	Sets the start condition (BBSY = 1, SCP = 0)
[2]	Sets the stop condition (BBSY=0, SCP=0)
[3]	Writes the data for transmission to ICDR0 and clears IRIC to 0 (at rising of 9th clock)
[4]	Write the data for transmission to ICDR0 and clears IRIC to 0 (at rising of 9th clock)
[5]	Outputs logic-low on the P52 (SCL0) and P97 (SDA0) pins (at falling of 8th clock).
[6]	Outputs logic-low on the P52 (SCL0) and P97 (SDA0) pins to generate 9th clock.
[7]	Outputs logic-high on the P52 (SCL0) pin to generate the stop condition.
[8]	Outputs logic-low on the P97 (SDA0) pin to generate the stop condition.
[9]	Initially sets the IIC0 module.

Figure 4.37 Principle of Operation when the Bus is Temporarily Disconnected

4.10.3 Description of Software

(1) Description of Modules

Table 4.33 describes the modules used in this example of a task.

Table 4.33 Module Description

Name	Label	Function
Main routine	main	Sets the stack pointer and the MCU mode, and enables an interrupt.
Initial setting	initialize	Makes initial settings of IIC0.
Master transmission 1	mst_trs_1	Uses master transmission to transmit 5 bytes of data to the EEPROM.
Master transmission 2	mst_trs_2	Uses master transmission to transmit 5 bytes of data to the EEPROM.
Start-condition setting	set_start	Sets the start condition.
Stop-condition setting	set_stop	Sets the stop condition.
Transmission of slave address + W	trs_slvadr_a0	Transmits the EEPROM's slave address + W bit as data (H'A0).
Transmission of location in EEPROM	trs_memadr	Transmits the address of a location within the EEPROM (H'00) as data.
Wait1	wait_1	Waits for 5 µs (in 20-MHz operation).
Wait2	wait_2	Waits for 10 µs (in 20-MHz operation).

(2) Description of On-chip Registers

Table 4.34 the usage of on-chip registers in this example of a task.

Table 4.34 On-chip Registers

Register		Function	Address	Setting
ICDR0		Stores the data for transmission.	H'FFDE	—
SAR0	FS	Along with the settings in the FSX bit of SARX0 and the SW bit of DDCSWR, sets the format for transfer.	H'FFDF bit0	0
SARX0	FSX	Along with the settings in the FS bit of SAR0 and the SW bit of DDSWR, sets the format for transfer.	H'FFDE bit0	1

Table 4.34 On-chip Registers (cont)

Register		Function	Address	Setting
ICMR0	MLS	Sets data transfer as MSB first.	H'FFDF bit7	0
	WAIT	Sets whether to insert wait cycles between the data bits and the acknowledge bit.	H'FFDF bit6	0/1
	CKS2 to CKS0	Along with the setting in the IICX0 bit of STCR, set the frequency of the transfer clock to 100 kHz.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set the number of bits for the next transfer in the I^2C bus format to 9 (9 bits/frame).	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Controls access to the ICMR0, ICDR0/SAR, and SARX registers, and selects the operation (the port function for the SCL0/SDA0 pin) or non-operation (bus-drive state for the SCL/SDA pin) of the I^2C bus interface.	H'FFD8 bit7	0/1
	IEIC	Disables the generation of interrupt requests by the I^2C bus interface.	H'FFD8 bit6	0
	MST	Uses the I^2C bus interface in its master mode.	H'FFD8 bit5	1
	TRS	Uses the I^2C bus interface in its transmission mode.	H'FFD8 bit4	1
	ACKE	Suspends the continuous transfer of data when the acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms whether or not the I^2C bus is occupied, and, in combination with the SCP bit, sets the start and stop conditions.	H'FFD8 bit2	0/1
	IRIC	Detects the start condition, determines the end of data transfer, and detects acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Along with the BBSY bit, sets the start/stop conditions.	H'FFD8 bit0	0
ICSR0	ACKB	Stores the acknowledgement to be transmitted to the EEPROM during the receive operation.	H'FFD9 bit0	—
STCR	IICX0	Along with the settings in CKS2 to CKS0 of ICMR0, selects the frequency of the transfer clock.	H'FFC3 bit5	1
	IICE	Enables CPU access to the data and control registers of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets the control registers of the flash memory to non-selected.	H'FFC3 bit3	0

Table 4.34 On-chip Registers (cont)

Register		Function	Address	Setting
DDCSWR	SWE	Prohibits automatic changeover from format-less transfer to transfer in the I^2C bus format on the channel 0 I^2C interface.	H'FEE6 bit7	0
	SW	Uses the channel 0 I^2C interface in the I^2C bus format.	H'FEE6 bit6	0
	IE	Prohibits interrupts during automatic changes of format.	H'FEE6 bit5	0
	CLR3 to CLR0	Control the initialization of the internal state of the channel 0 I^2C interface	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels the module-stopped mode for SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels the module stopped mode for I^2C channel 0.	H'FF87 bit4	0
SCR0	CKE1, 0	Make the I/O port setting for the P52/SCK0/SCL0 pin.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets the mode for SCI transfer on channel 0 as asynchronous.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set the interrupt-control mode of the interrupt controller to 1-bit control.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set the MCU's operating mode to mode 3 by latching the input levels on the MD1 and 0 pins.	H'FFC5 bit1, 0	MDS1=1 MDS0=1
P5DDR	P52DDR	Sets the P52 pin to act as an output pin.	H'FFB8 bit2	1
P5DR	P52DR	Sets the data for output on the P52 pin.	H'FFBA bit2	0/1
P9DDR	P97DDR	Sets the P97 pin to act as an output pin.	H'FFC0 bit7	1
P9DR	P97DR	Sets the data for output on the P97 pin.	H'FFC1 bit7	0/1

(3) Description of Variables

Table 4.35 describes the variables used in this task.

Table 4.35 Description of Variables

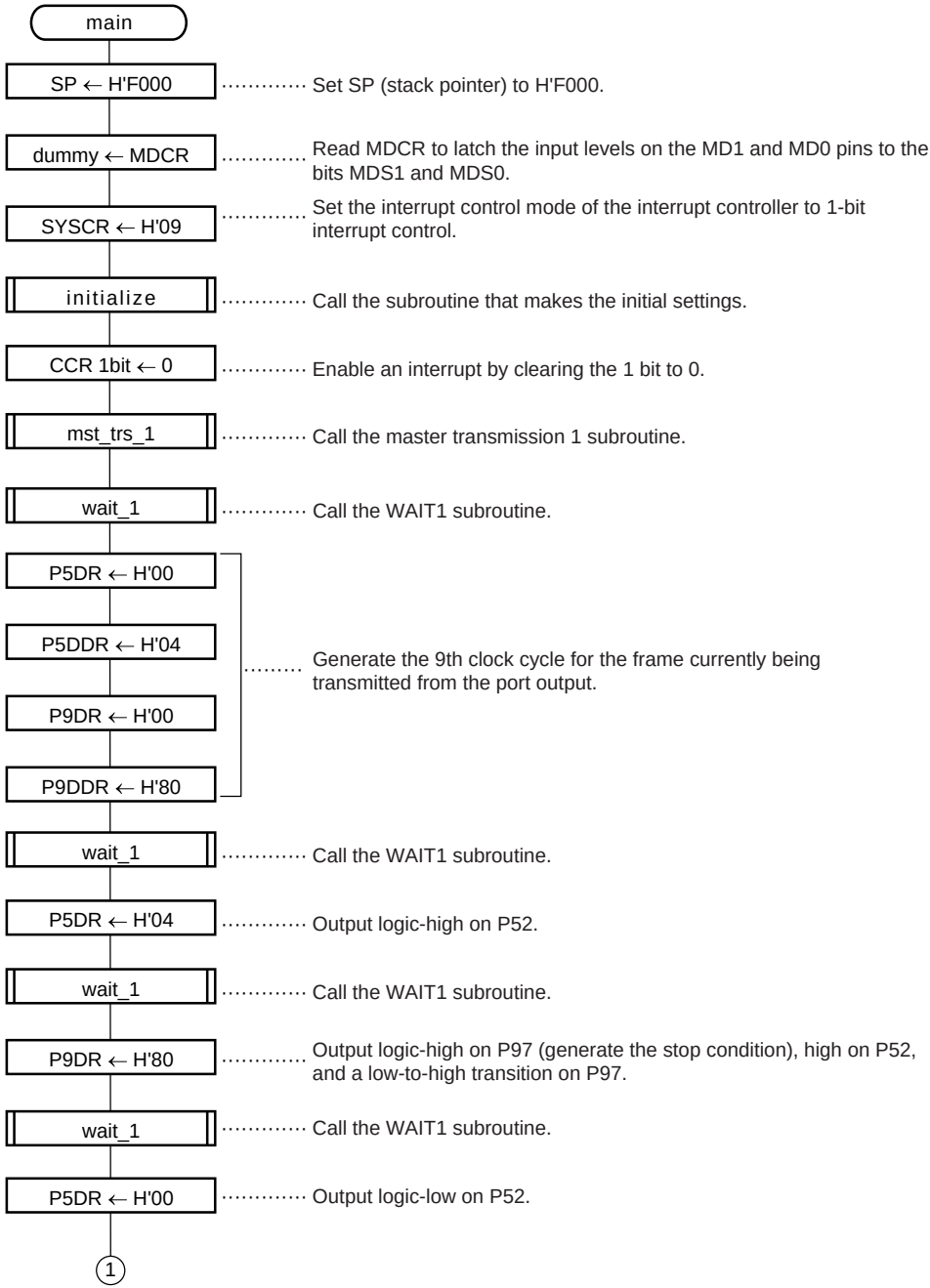
Variable	Function	Size	Initial Value	Name of Using Module
i	Transmission data counter	1 byte	H'00	mst_trs_1 mst_trs_2
dummy	Stores the MDCR value.	1 byte	—	main
dt_trs[i]	5 bytes transmission data	5 bytes	—	mst_trs_1 mst_trs_2

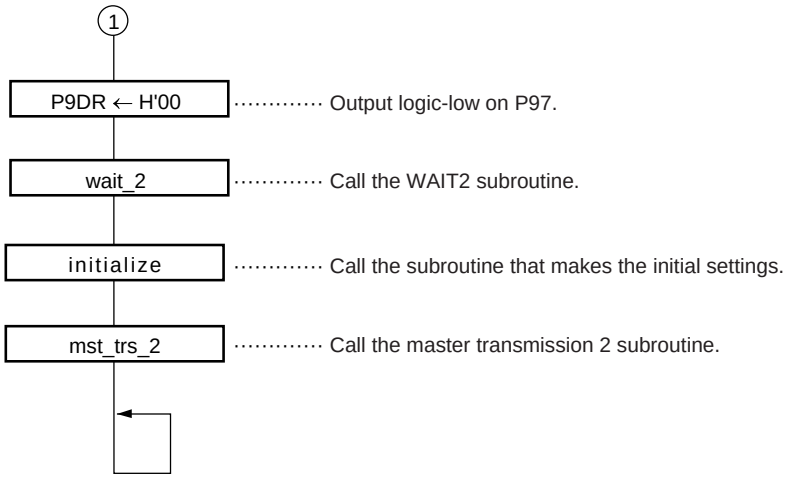
(4) Description of RAM Usage

In this example of a task, the only RAM used is that required for the variables.

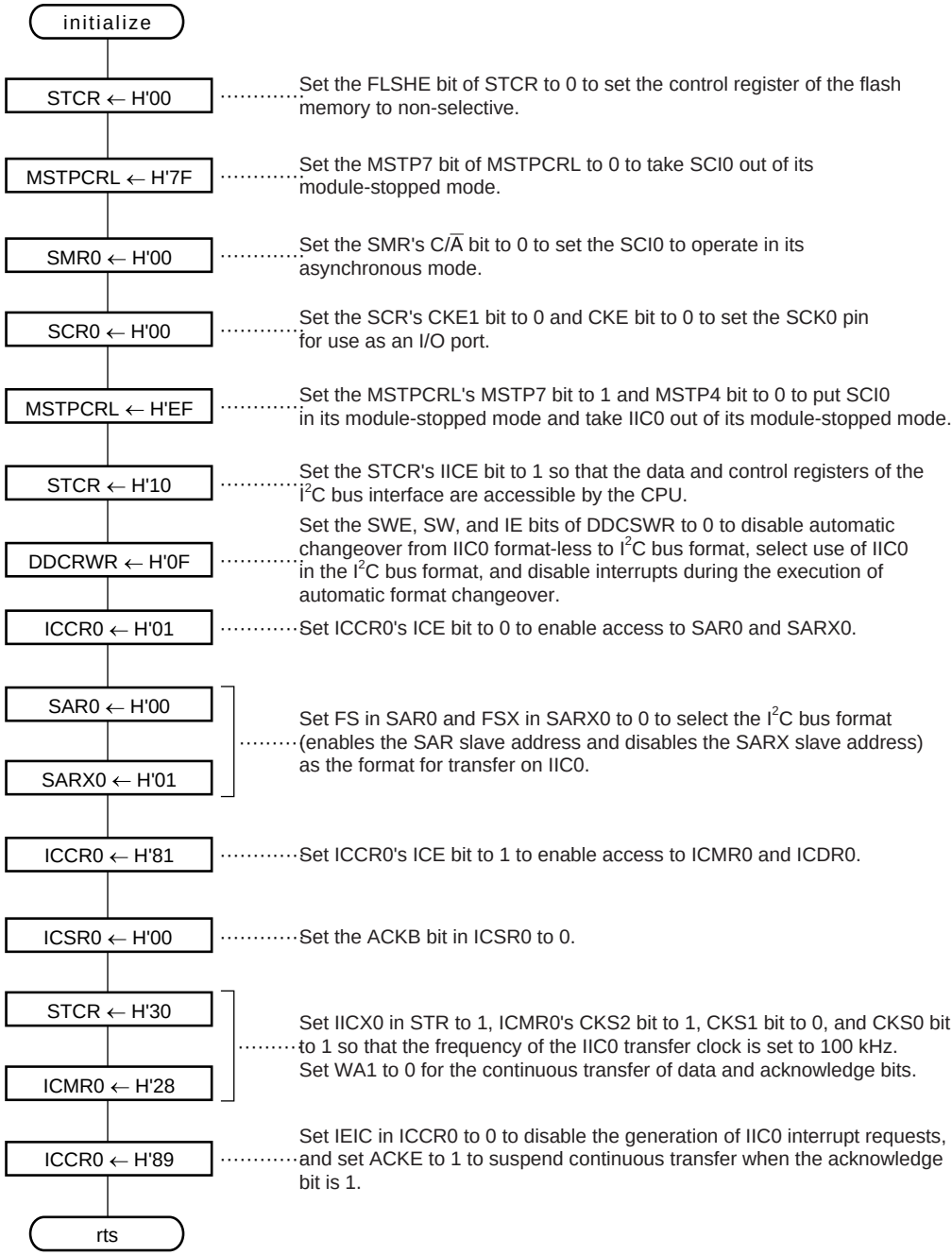
4.10.4 Flowcharts

(1) Main Routine

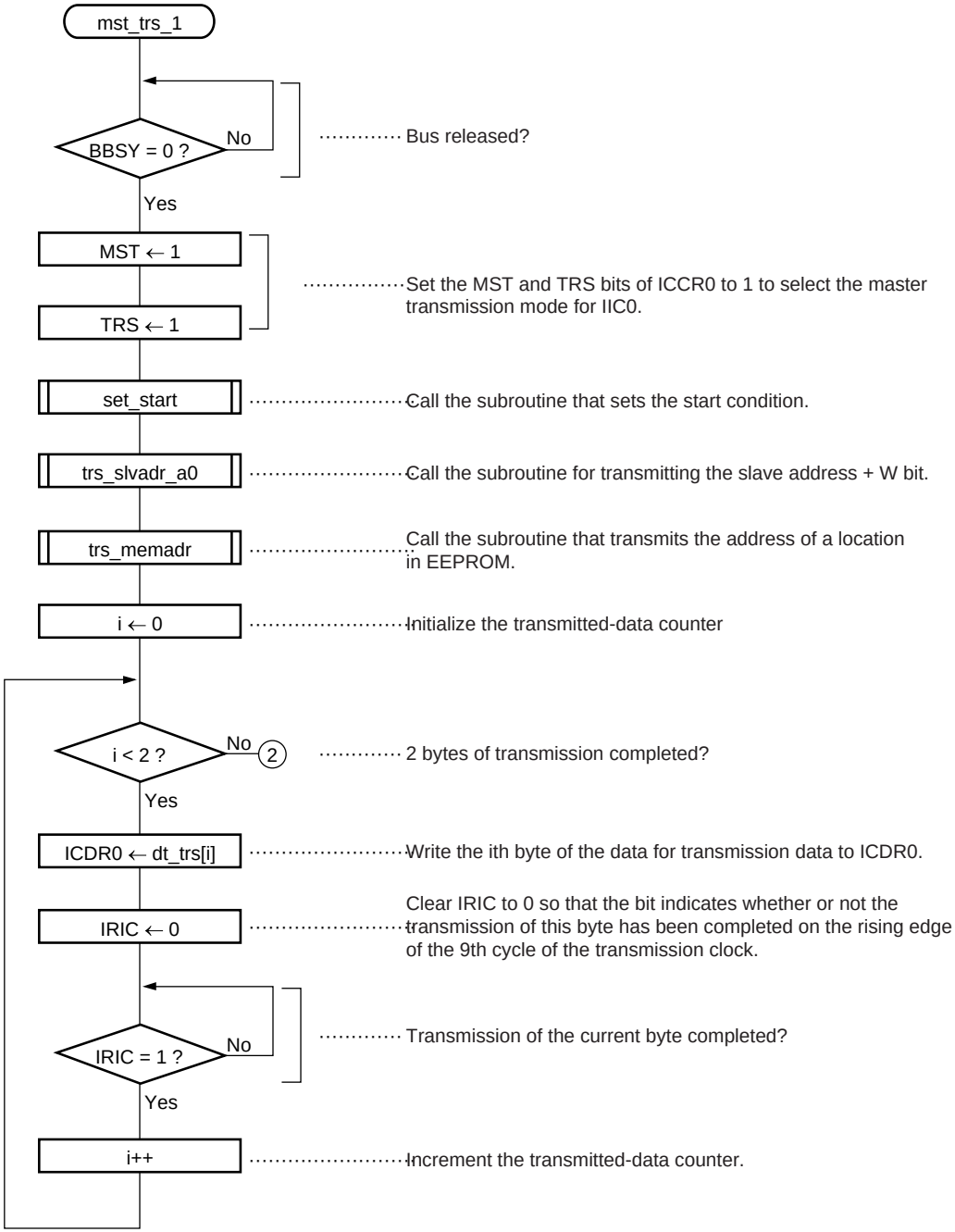


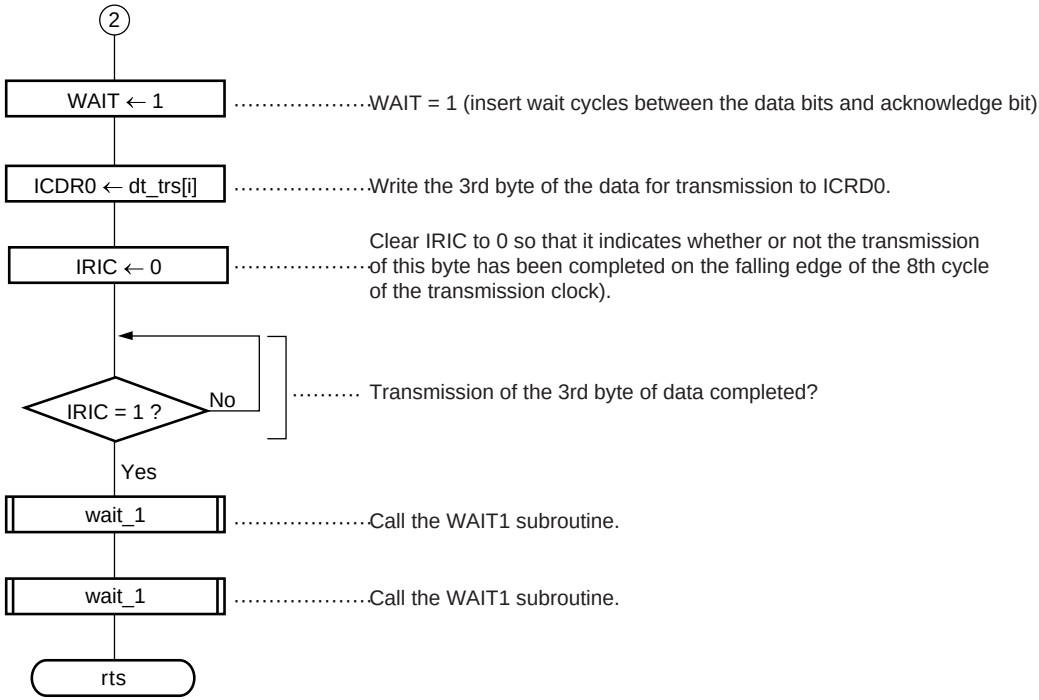

1
P9DR ← H'00
Output logic-low on P97.
wait_2
Call the WAIT2 subroutine.
initialize
Call the subroutine that makes the initial settings.
mst_trs_2
Call the master transmission 2 subroutine.

(2) Subroutine for Making Initial Settings

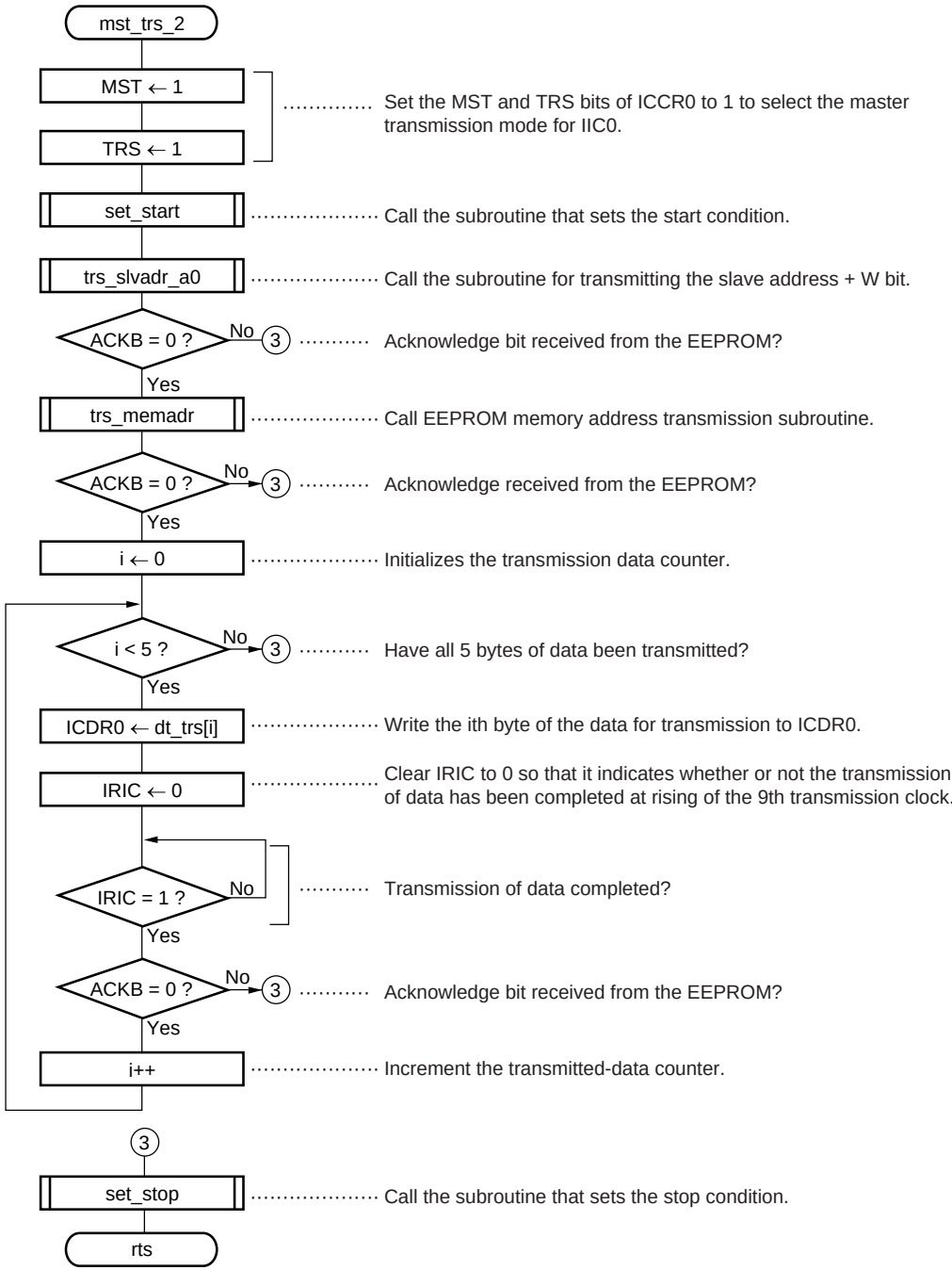


(3) Master transmission 1 subroutine

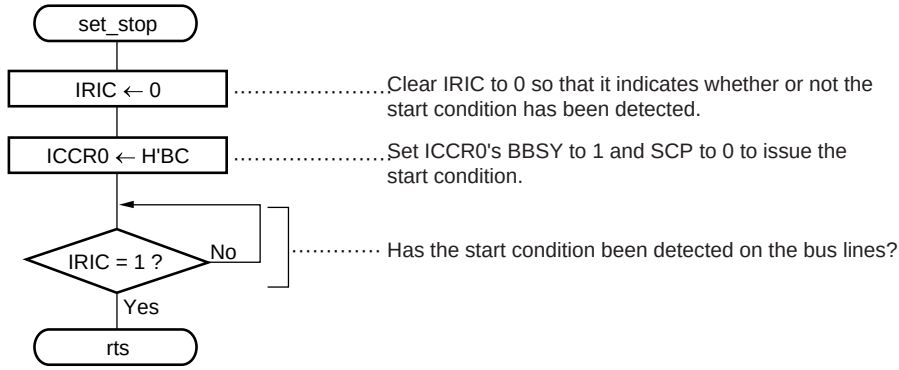



2
WAIT ← 1
WAIT = 1 (insert wait cycles between the data bits and acknowledge bit)
ICDR0 ← dt_trs[i]
Write the 3rd byte of the data for transmission to ICRD0.
IRIC ← 0
Clear IRIC to 0 so that it indicates whether or not the transmission of this byte has been completed on the falling edge of the 8th cycle of the transmission clock).
IRIC = 1 ?
No
Transmission of the 3rd byte of data completed?
Yes
wait_1
Call the WAIT1 subroutine.
wait_1
Call the WAIT1 subroutine.
rts

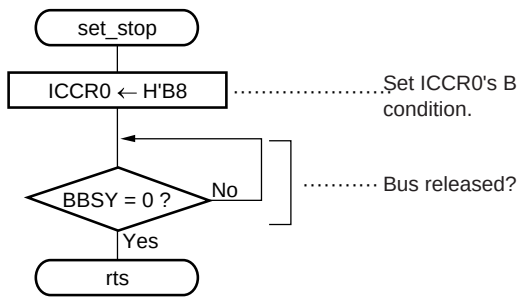
(4) Master Transmission 2 Subroutine



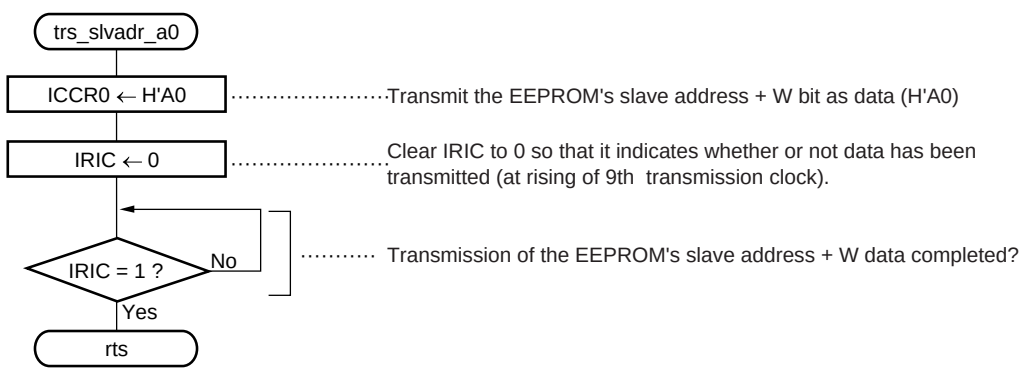
(5) Subroutine for Setting the Start Condition



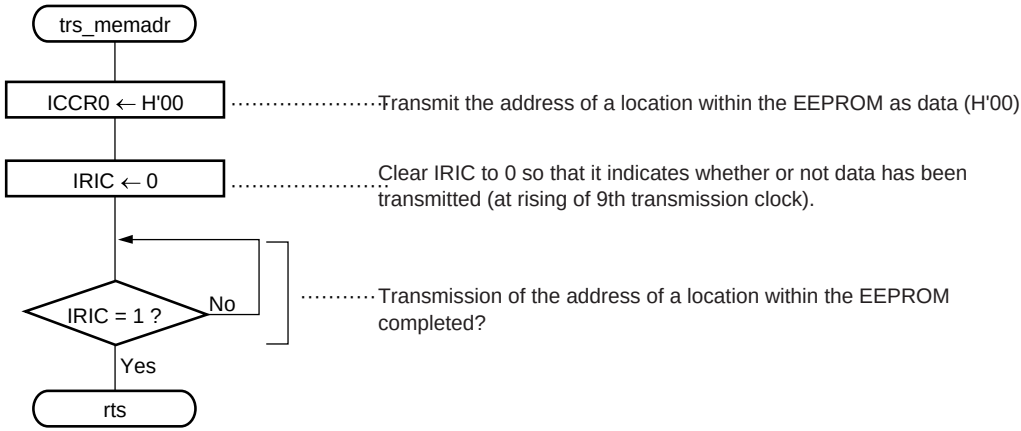
(6) Subroutine for Setting the Stop Condition



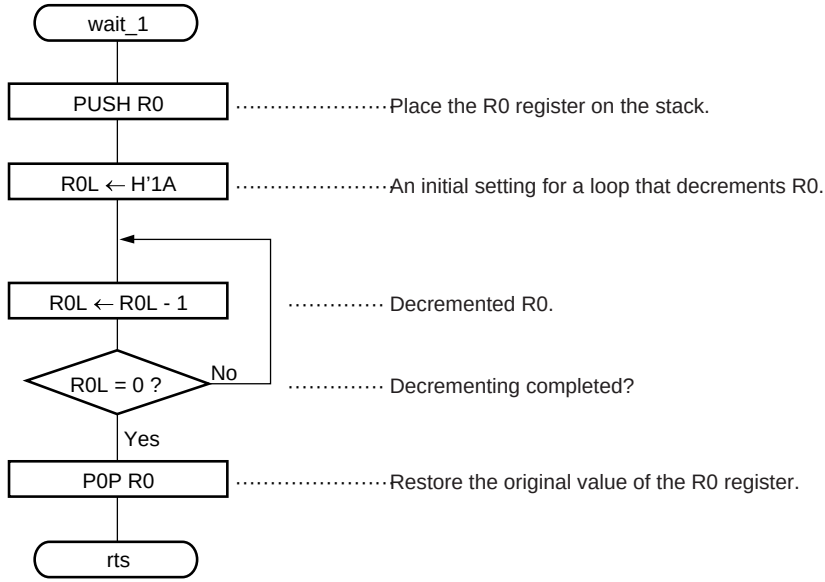
(7) Subroutine for transmitting the slave address + W bit



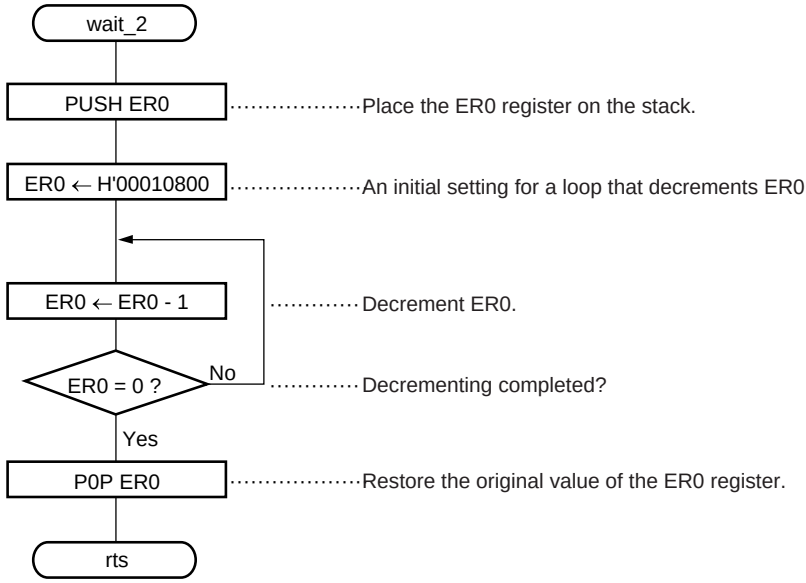
(8) Subroutine for transmitting the location within the EEPROM



(9) WAIT1 subroutine



(10) WAIT2 subroutine



4.10.5 Program List

```
/*****************************************************
*  H8S/2138 IIC bus application note                 *
*   9.Error process in single master transmit        *
*                      File name   :   Error.c       *
*                      Fai         :   20MHz         *
*                      Mode        :   3             *
*****************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*****************************************************
*  Prototype                                         *
*****************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* IIC0 initialize */
void mst_trs_1(void);                               /* Master transmit 1 */
void mst_trs_2(void);                               /* Master transmit 2 */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_memadr(void);                              /* EEPROM memory address data transmit */
void wait_1(void);                                  /* Wait 1 (5µs) */
void wait_2(void);                                  /* Wait 2 (10ms) */

/*****************************************************
*  Data table                                        *
*****************************************************/
const unsigned char dt_trs[5] =                     /* Transmit data (5 byte) */
{
    0xa1,                                           /* 1st transmit data */
    0xb2,                                           /* 2nd transmit data */
    0xc3,                                           /* 3rd transmit data */
    0xd4,                                           /* 4th transmit data */
    0xe5                                            /* 5th transmit data */
```

```
};

/*****************************************************
*  main : Main routine                               *
*****************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                          ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                              /* MCU mode set */

     SYSCR.BYTE = 0x09;                              /* Interrupt control mode set */
     initialize();                                   /* Initialize */

     set_imask_ccr(0);                               /* Interrupt enable */

     mst_trs_1();                                    /* Master transmit to EEPROM 1 */

     wait_1();                                       /* 5µs wait */
     P5.DR.BYTE = 0x00;                              /* P52DR = 0 */
     P5.DDR = 0x04;                                  /* P52DDR = 1 */
     P9.DR.BYTE = 0x00;                              /* P97DR = 0 */
     P9.DDR = 0x80;                                  /* P97DDR = 1 */
     wait_1();                                       /* 5µs wait */
     P5.DR.BYTE = 0x04;                              /* P52DR = 1 */
     wait_1();                                       /* 5µs wait */
     P9.DR.BYTE = 0x80;                              /* P92DR = 1 */
     wait_1();                                       /* 5µs wait */
     P5.DR.BYTE = 0x00;                              /* P52DR = 0 */
     P9.DR.BYTE = 0x00;                              /* P97DR = 0 */
     wait_2();                                       /* 10ms wait (EEPROM write cycle) */

     initialize();                                   /* IIC0 initialzie */

     mst_trs_2();                                    /* Master transmit to EEPROM 2 */
```

```
    while(1);/* End */
}

/*****************************************************
*  initialize : IIC0 Initialize                      *
*****************************************************/
void initialize(void)
{
    STCR.BYTE = 0x00;                                /* FLSHE = 0 */
    MSTPCR.BYTE.L = 0x7f;                            /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                            /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    BSC.WSCR.BYTE = 0x33;                            /* SDA0 pin function set */
    MSTPCR.BYTE.L = 0xef;                            /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                                /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                              /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                           /* ICE = 0 */
    IIC0.SAR.BYTE = 0x00;                            /* FS = 0 */
    IIC0.SARX.BYTE = 0x01;                           /* FSX = 1 */
    IIC0.ICCR.BYTE = 0x81;                           /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                           /* ACKB = 0 */
    STCR.BYTE = 0x30;                                /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                           /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                           /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  mst_trs_1 : Master transmit to EEPROM 1           *
*****************************************************/
void mst_trs_1(void)
{
    unsigned char i;                                 /* Transmit data counter */

    while(IIC0.ICCR.BIT.BBSY == 1);                  /* Bus empty (BBSY=0) ? */
    IIC0.ICCR.BIT.MST = 1;                           /* Master transmit mode set */
    IIC0.ICCR.BIT.TRS = 1;                           /* MST = 1, TRS = 1 */
    set_start();                                     /* Start condition set */
```

```
    trs_slvadr_a0();                        /* Slave address + W data transmit */
    trs_memadr();                           /* EEPROM memory address data transmit */
    for(i=0; i<2; i++)
    {
        IIC0.ICDR = dt_trs[i];              /* Transmit data write */
        IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
        while(IIC0.ICCR.BIT.IRIC == 0);     /* Transmit end (IRIC=1) ? */
    }
    IIC0.ICMR.BIT.WAIT = 1;                 /* WAIT = 1 */
    IIC0.ICDR = dt_trs[i];                  /* 3rd transmit data write */
    IIC0.ICCR.BIT.IRIC = 0;                 /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);         /* Transmit end (IRIC=1) ? */
    wait_1()                                /* 5µs wait */
    wait_1();                               /* 5µs wait */
    IIC0.ICCR.BIT.ICE = 0;                  /* ICE = 0 */
}

/*****************************************************
*  mst_trs_2 : Master transmit to EEPROM 2           *
*****************************************************/
void mst_trs_2(void)
{
    unsigned char i;                        /* Transmit data counter */

    IIC0.ICCR.BIT.MST = 1;                  /* Matser transmit mode set */
    IIC0.ICCR.BIT.TRS = 1;                  /* MST = 1, TRS = 1 */
    set_start();                            /* Start condition set */
    trs_slvadr_a0();                        /* Slave address + W data transmit */

    if(IIC0.ICSR.BIT.ACKB == 0)             /* ACKB = 0 ? */
    {
        trs_memadr();                       /* EEPROM memory address data transmit */

        if(IIC0.ICSR.BIT.ACKB == 0)         /* ACKB = 0 ? */
        {
            for(i=0; i<5; i++)
            {
```

```
                IIC0.ICDR = dt_trs[i];              /* Transmit data write */
                IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);     /* Transmit end (IRIC=1) ? */

                if(IIC0.ICSR.BIT.ACKB == 1)         /* ACKB = 0 ? */
                {
                    break;
                }
            }
        }
    }
    set_stop();                                     /* Stop condition set */
}

/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
void set_start(void)
{
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    IIC0.ICCR.BYTE = 0xbc;                          /* Start condition set (BBSY=1,SCP=0) */
    hile(IIC0.ICCR.BIT.IRIC == 0);                  /* Start condition set (IRIC=1) ? */
}

/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
    IIC0.ICCR.BYTE = 0xb8;                          /* Stop condition set (BBSY=0,SCP=0) */
    while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
}

/*****************************************************
*  trs_slvadr_a0 : Slave addres + W data transmit    *
*****************************************************/
void trs_slvadr_a0(void)
```

```
{
     IC0.ICDR = 0xa0;                                /* Slave address + W data(H'A0) write */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_memadr : EEPROM memory address data transmit  *
*****************************************************/
void trs_memadr(void)
{
     IIC0.ICDR = 0x00;                               /* EEPROM memory address data(H'00) write */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  wait_1 : xxms wait                                *
*****************************************************/
void wait_1(void)
{
#pragma asm
         push.w  r0                                  ;Push R0
         mov.b   #h'1a,r0l                           ;Decrement data set
wait1_1:
         dec.b   r0l                                 ;Decrement
         bne     wait1_1                             ;Decrement end ?
         pop.w   r0    ;Pop R0
#pragma endasm
}

/*****************************************************
*  wait_2 : 10ms wait                                *
*****************************************************/
void wait_2(void)
{
#pragma asm
```

```
        push.l  er0                             ;Push ER0
        mov.l   #h'00010800,er0                 ;Decrement data set
wait2_1:
        dec.l   #1,er0                          ;Decrement
        bne     wait2_1                         ;Decrement end ?
        pop.l   er0                             ;Pop ER0
#pragma endasm
}
```

4.11 Bus Conflict

4.11.1 Specifications

- The system is in a multiple-task configuration with master devices 1 and 2 (H8S/2138) and a slave device (EEPROM: HN58X2408).
- When the IRQ interrupt switch which is connected to masters 1 and 2 is pressed, masters 1 and 2 write 2 bytes of data to the slave device.
- The data transmitted from master 1 displays “10” in the 7-segment LED.
- The data transmitted from master 2 displays “20” in the 7-segment LED.
- Since masters 1 and 2 attempt to start the transmission of data at the same time, a bus conflict occurs. In this case, the master that has acquired the bus right continues to write data to the EEPROM and lights up the LED. The master that failed to acquire the bus right reads the data written by the other master from the EEPROM and displays the data on the 7-segment LED after the other master has finished writing to the EEPROM.
- The slave address of the EEPROM, which is the slave device in this example, is [1010000]. Data is read from and written to the locations at addresses H'00 and H'01 in the EEPROM.
- The frequency of the transfer clock, for both receiving and transmission, is 100 kHz.
- Figure 4.38 shows the configuration of the system used in this example of a task.

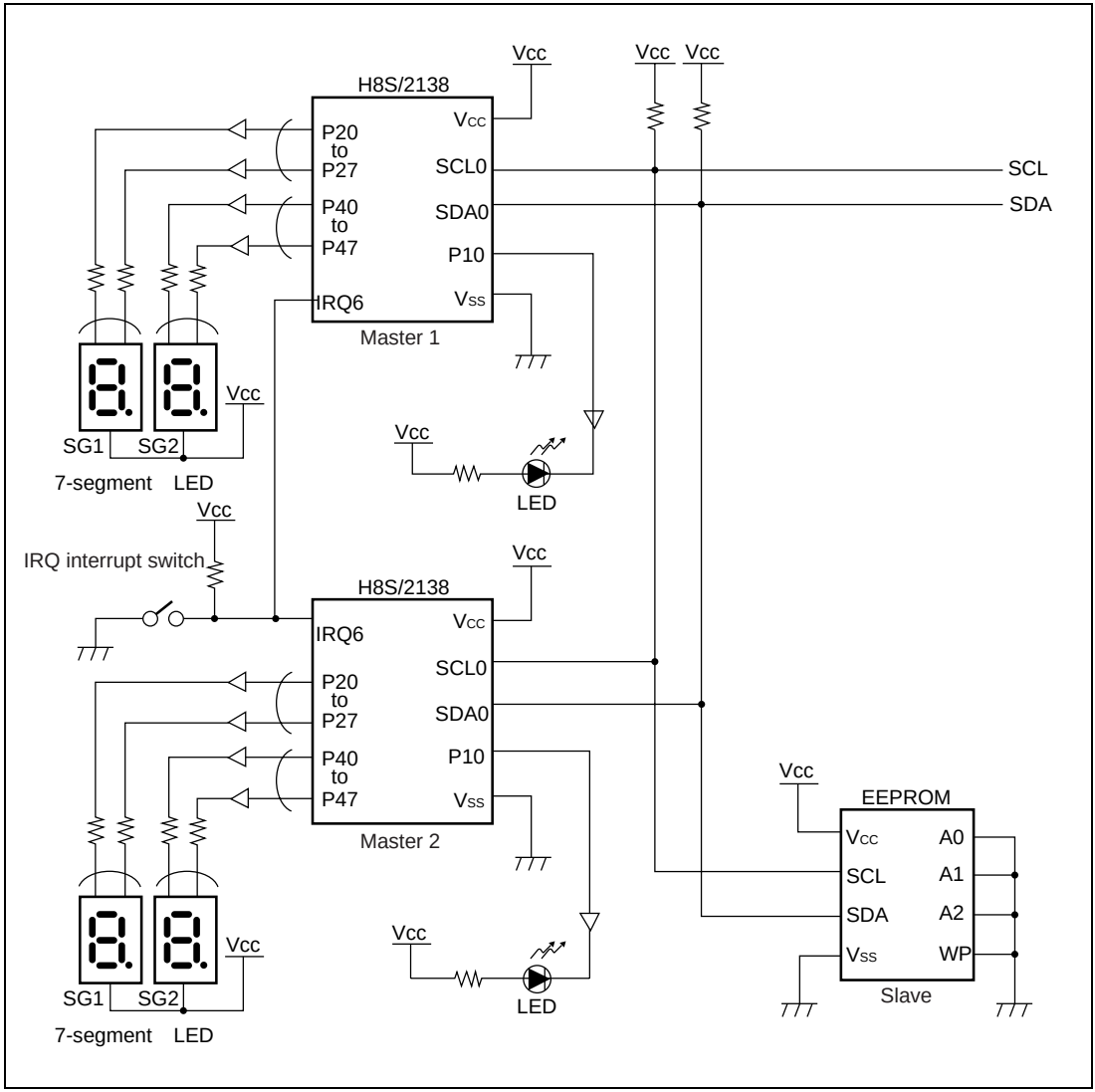


Figure 4.38 System Configuration

The I^{2}C bus format used in this example of a task is shown in figure 4.39.

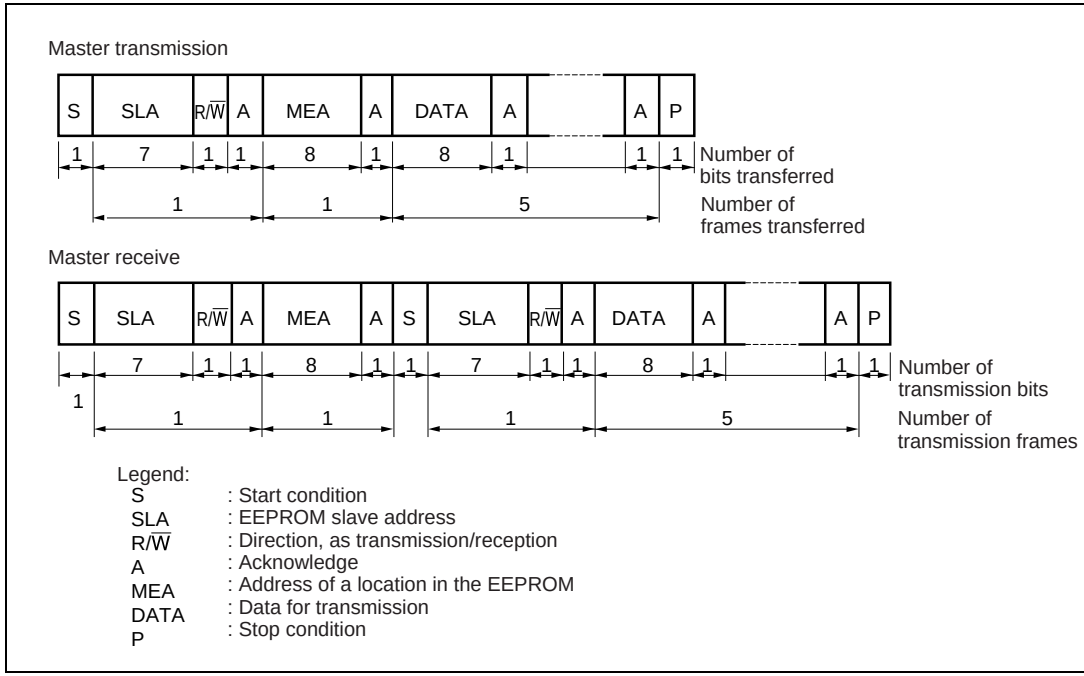


Figure 4.39 The Formats for Transfer Used in this Example of a Task

- The I^2C bus interface that is incorporated in H8S Series products includes the procedure for adjusting communications shown in figure 4.40 as well as the procedures described in section 1.4, Procedures for Adjusting Communications. Each master device monitors the bus line on the falling edge of SCL. When the level on the bus line does not match the level a master is attempting to put out, that master's output stage is cutoff.

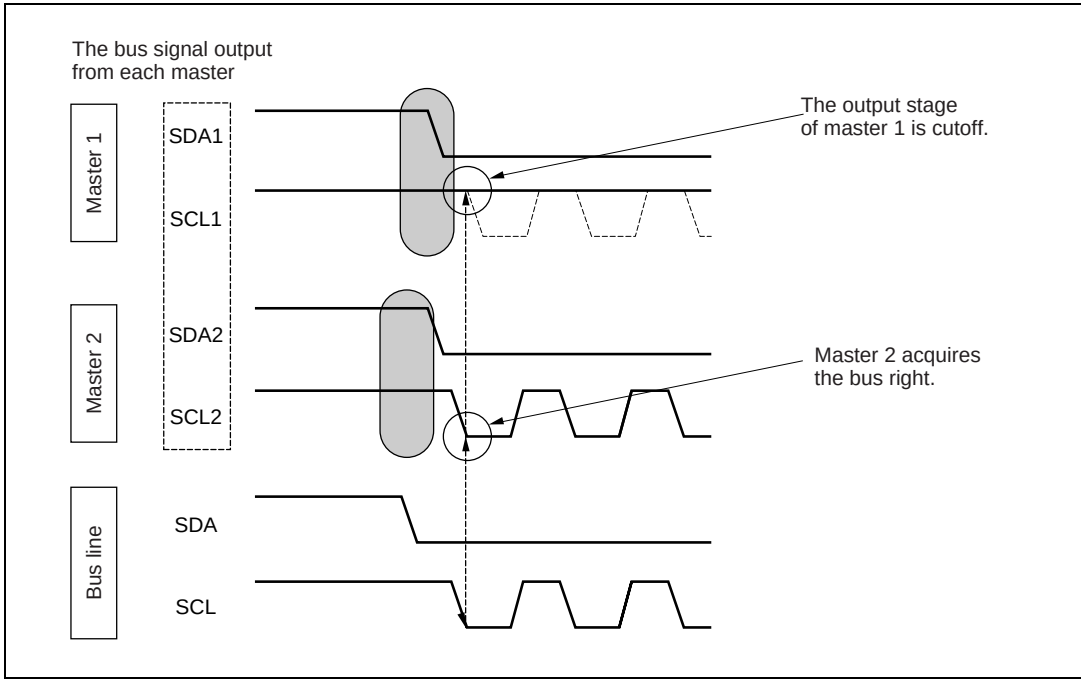


Figure 4.40 Method of Detecting Bus Arbitration

In this example, masters 1and 2 are attempting to simultaneously transmit data to the same slave device. Since the first lot of data sent (first frame) is the slave address plus the W bit, and this is immediately followed by the address of a location within the EEPROM's memory, both masters are initially sending the same data. The conflict thus does not arise until the third frame, the data to be stored at the first location in the EEPROM, is sent. The third frame of the data for transmission (the first byte is a transmission data) by master 1 is H'F9 while the third from master 2 is H'A4, so the first difference that appears is when master 2 sets SDA to its low level. For reasons that are explained in more detail in section 1.4, master 2 thus acquires bus mastership (see figure 4.41).

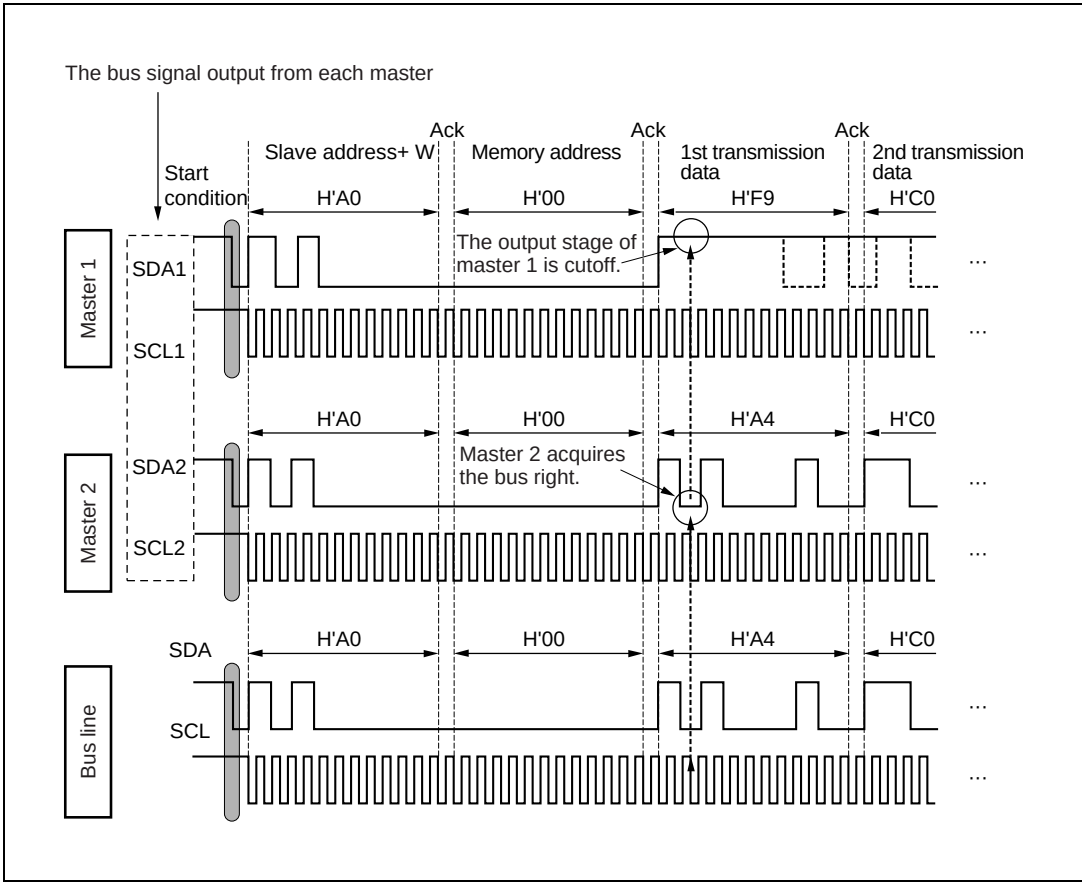
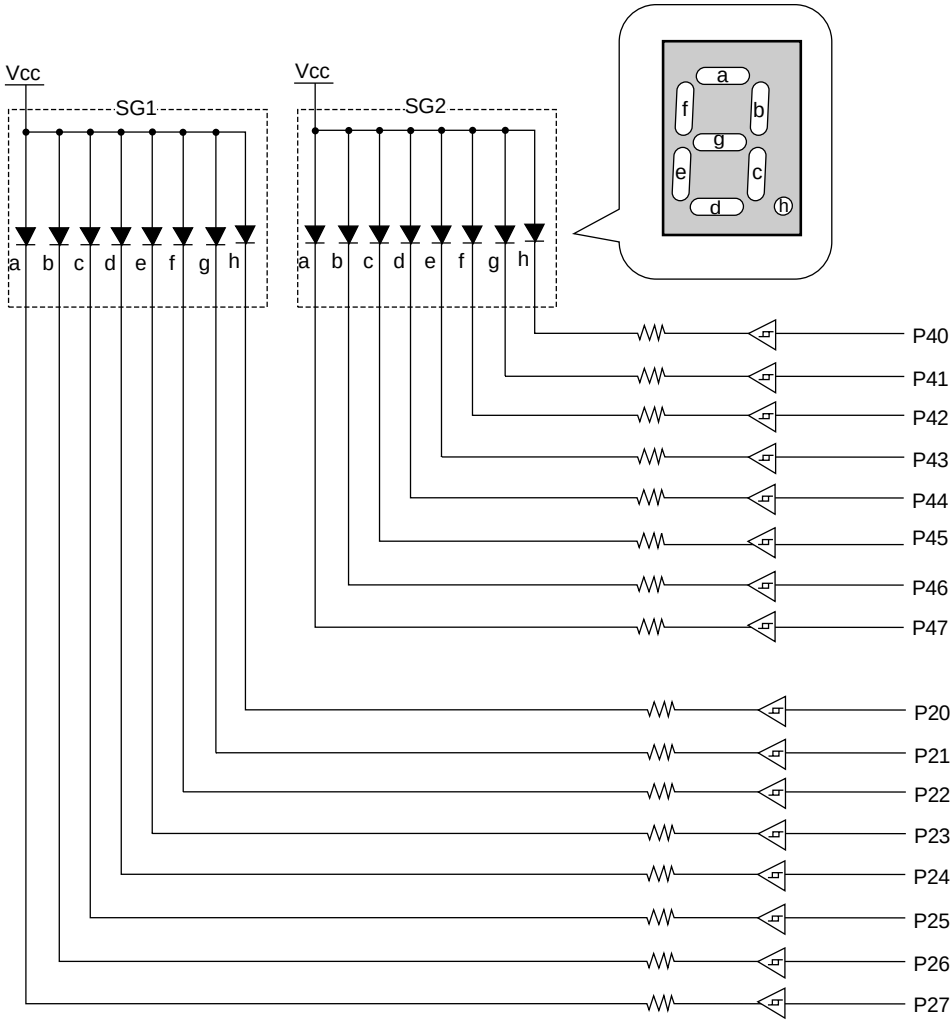


Figure 4.41 How Master 2 Becomes the Bus Master

- The connections between the 7-segment LED and the H8S/2138 used in this example of a task is shown in figure 4.42. The segments of each of the LEDs are lit by the output of low levels from ports 2 or 4.



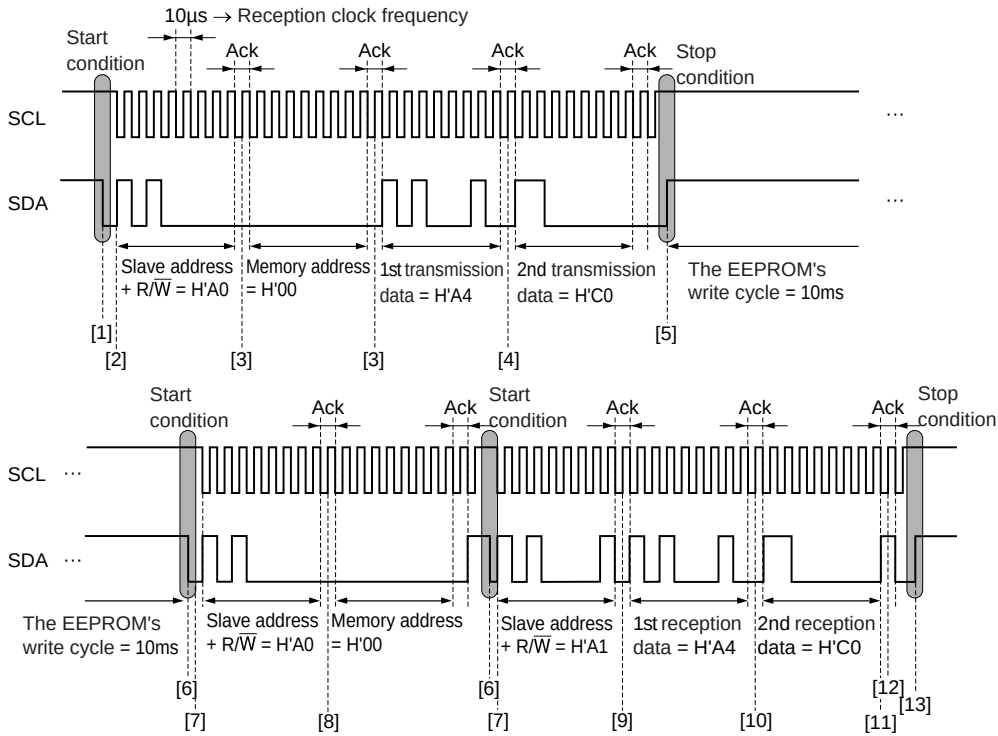
The display on SG1 and the data output from port 2

Display	P20	P21	P22	P23	P24	P25	P26	P27	Display	P20	P21	P22	P23	P24	P25	P26	P27
0	1	1	0	0	0	0	0	0	8	1	0	0	0	0	0	0	0
1	1	1	1	1	1	0	0	1	9	1	0	0	1	0	0	0	0
2	1	0	1	0	0	1	0	0	A	1	0	0	0	1	0	0	0
3	1	0	1	1	0	0	0	0	B	1	0	0	0	0	0	1	1
4	1	0	0	1	1	0	0	1	C	1	1	0	0	0	1	1	0
5	1	0	0	1	0	0	1	0	D	1	0	1	0	0	0	0	1
6	1	0	0	0	0	0	1	0	E	1	0	0	0	0	1	1	0
7	1	1	0	1	1	0	0	0	F	1	0	0	0	1	1	1	0

Figure 4.42 7-segment LED Connection Diagram

4.11.2 Operation Description

Figure 4.43 shows this example's principle of operation.



	Processing by master 1	Processing by master 2
[1]	Sets the start condition	Sets the start condition
[2]	Detects the start condition (IRIC = 1), writes the data for transmission to ICDR0, and clears IRIC to 0.	Detects the start condition (IRIC = 1), writes the data for transmission to ICDR0, and clears IRIC to 0.
[3]	Writes the data for transmission to ICDR0 at rising of 9th clock (IRIC = 1) and clears IRIC to 0.	Writes the data for transmission to ICDR0 at rising of 9th clock (IRIC = 1) and clears IRIC to 0.
[4]	Fails to acquire the bus (AL = 1) and enters the slave-reception mode (MST = 0, TRS = 0)	Sets the start condition (BBSY = 0, SCP = 1)
[5]	No operation	No operation
[6]	Sets the start condition (BBSY = 0, SCP = 1)	No operation
[7]	Detects the start condition (IRIC = 1), writes the data for transmission to ICDR0, and clears IRIC to 0.	No operation
[8]	Writes the data for transmission to ICDR0 at rising of 9th clock (IRIC = 1) and clears IRIC to 0.	No operation
[9]	Detects the start condition (IRC = 1), dummy-reads ICDR0, and clears IRIC to 0.	No operation
[10]	Reads the 1st byte of the received data from ICDR0 at rising of 9th clock (IRIC = 1) ACKB = WAIT = 1, IRIC = 0	No operation
[11]	Moves to the master receive mode (TRS = 0) and clears IRIC to 0 at falling of 8th clock (IRIC = 1).	No operation
[12]	Reads the 2nd byte of received data from ICDR0 on the falling edge of a 1 clock cycle (IRIC = 1).	No operation
[13]	Sets the start condition (BBSY = 0, SCP = 1)	No operation

Figure 4.43 Operation in a Bus Conflict

4.11.3 Description of Software

(1) Description of Modules

Table 4.36 describes the modules used in this example of a task.

Table 4.36 Description of Modules

Name	Label	Function
Main routine	main	Sets the stack pointer, MCU mode, and IRQ6 interrupt, and enables interrupts. When this device has acquired bus mastership, transmits 2 bytes of data by master transmission and lights the LED. When it is unable to acquire the bus, it acts as a master receiver to receive 2 bytes of data and displays the data on the 7-segment LED to which this master is attached.
Initial setting	initialize	Makes initial settings for the RAM, ports, and IIC0.
Start-condition setting	set_start	Sets the start condition.
Stop-condition setting	set_stop	Sets the stop condition.
Transmission of slave address + W	trs_slvadr_a0	Transmits the EEPROM's slave address + W bit as data (H'A0).
Transmission of slave address + R	trs_slvadr_a1	Transmits the EEPROM's slave address + R bit as data (H'A1).
Transmission of location in EEPROM	trs_memadr	Transmits the address of a location within the EEPROM as data (H'00).
Wait	wait_1	Waits for completion of the EEPROM-write cycle (10 ms: in 20-MHz operation)

(2) Description of the On-chip Registers

Table 4.37 describes the usage of on-chip registers in this example of a task.

Table 4.37 On-chip Registers

Register		Function	Address	Setting
ICDR0		Stores the data for transmission/received data	H'FFDE	—
SAR0	FS	Along with the settings in the FSX bit of SARX0 and the SW bit of DDCSWR, sets the format for transfer.	H'FFDF bit0	0
SARX0	FSX	Along with the settings in the FS bit of SAR0 and the SW bit of DDSWR, sets the format for transfer.	H'FFDE bit0	1

Table 4.37 On-chip Registers (cont)

Register		Function	Address	Setting
ICMR0	MLS	Sets data transfer as MSB first.	H'FFDF bit7	0
	WAIT	Sets whether to insert wait cycles between the data bits and the acknowledge bit.	H'FFDF bit6	0/1
	CKS2 to CKS0	Along with the setting in the IICX0 bit of STCR, set the frequency of the transfer clock to 100 kHz.	H'FFDF bit5 to bit3	CKS2=1 CKS1=0 CKS0=1
	BC2 to BC0	Set the number of bits for the next transfer in the I^2C bus format to 9 (9 bits/frame).	H'FFDF bit2 to bit0	BC2=0 BC1=0 BC0=0
ICCR0	ICE	Controls access to the ICMR0, ICDR0/SAR, and SARX registers, and selects the operation (the port function for the SCL0/SDA0 pin) or non-operation (bus-drive state for the SCL/SDA pin) of the I^2C bus interface.	H'FFD8 bit7	0/1
	IEIC	Disables the generation of interrupt requests by the I^2C bus interface.	H'FFD8 bit6	0
	MST	Uses the I^2C bus interface in its master mode.	H'FFD8 bit5	1
	TRS	Uses the I^2C bus interface in its transmission or reception mode.	H'FFD8 bit4	0/1
	ACKE	Suspends the continuous transfer of data when the acknowledge bit is 1.	H'FFD8 bit3	1
	BBSY	Confirms whether or not the I^2C bus is occupied, and, in combination with the SCP bit, sets the start and stop conditions.	H'FFD8 bit2	0/1
	IRIC	Detects the start condition, determines the end of data transfer, and detects acknowledge = 1.	H'FFD8 bit1	0/1
	SCP	Along with the BBSY bit, sets the start/stop conditions.	H'FFD8 bit0	0
ICSR0	ACKB	Stores the acknowledgement that is transmitted from the EEPROM during transmission, and sets the acknowledge data for output to the EEPROM during a receive operation.	H'FFD9 bit0	—
STCR	IICX0	Sets the frequency of the transfer clock to 100 kHz with the CKS2 to CKS0 of ICMR0.	H'FFC3 bit5	1
	IICE	Enables CPU access to the data register and the control register of the I^2C bus interface.	H'FFC3 bit4	1
	FLSHE	Sets the control register of the flash memory to non-selected.	H'FFC3 bit3	0

Table 4.37 On-chip Registers (cont)

Register		Function	Address	Setting
DDCSWR	SWE	Prohibits automatic changeover from format-less transfer to transfer in the I^2C bus format on the channel 0 I^2C interface.	H'FEE6 bit7	0
	SW	Uses the channel 0 I^2C interface in the I^2C bus format.	H'FEE6 bit6	0
	IE	Prohibits interrupts during automatic changes of format.	H'FEE6 bit5	0
	CLR3 to CLR0	Control the initialization of the internal state of the channel 0 I^2C interface.	H'FEE6 bit3 to bit0	CLR3=1 CLR2=1 CLR1=1 CLR0=1
MSTPCRL	MSTP7	Cancels the module-stopped mode for SCI channel 0.	H'FF87 bit7	0
	MSTP4	Cancels the module-stopped mode for I^2C channel 0.	H'FF87 bit4	0
SCR0	CKE1, 0	Make the I/O port setting for the P52/SCK0/SCL0 pin.	H'FFDA bit1, 0	CKE1=0 CKE0=0
SMR0	C/$\overline{A}$	Sets the mode for SCI transfer on channel 0 as asynchronous.	H'FFD8 bit7	0
SYSCR	INTM1, 0	Set the interrupt-control mode of the interrupt controller to 1-bit control.	H'FFC4 bit5, 4	INTM1=0 INTM0=0
MDCR	MDS1, 0	Set the MCU's operating mode to mode 3 by latching the input levels on the MD1 and 0 pins.	H'FFC5 bit1, 0	MDS1=1 MDS0=1
P1DDR	P10DDR	Sets the P10 pin to act as an output pin.	H'FFB0 bit0	1
P1DR	P10DDR	Sets the data for output on the P10 pin.	H'FFB2 bit0	0/1
P2DDR		Sets port 2 to act as an output port.	H'FFB1	H'FF
P2DR		Sets the data for output on port 2.	H'FFB3	—
P4DDR		Sets port 4 to act as an output pin.	H'FFB5	H'FF
P4DR		Sets the data for output on port 4.	H'FFB7	—
ISCRH		Generates an interrupt request at the falling edge of the IRQ6 input.	H'FEEC	H'10
ISR	IRQ6F	Displays the state of the IRQ6-interrupt request.	H'FEEB bit6	0/1

(3) Description of Variables

Table 4.38 describes the variables used in this example of a task.

Table 4.38 Description of Variables

Variable	Function	Size	Initial Value	Name of Using Module
dummy	MDCR read value	1 byte	—	main
dt_trs[0]	1st byte of data for transmission	1 byte	H'F9/A4	main
dt_trs[1]	2nd byte of data for transmission	1 byte	H'C0	main

(4) Description of RAM Usage

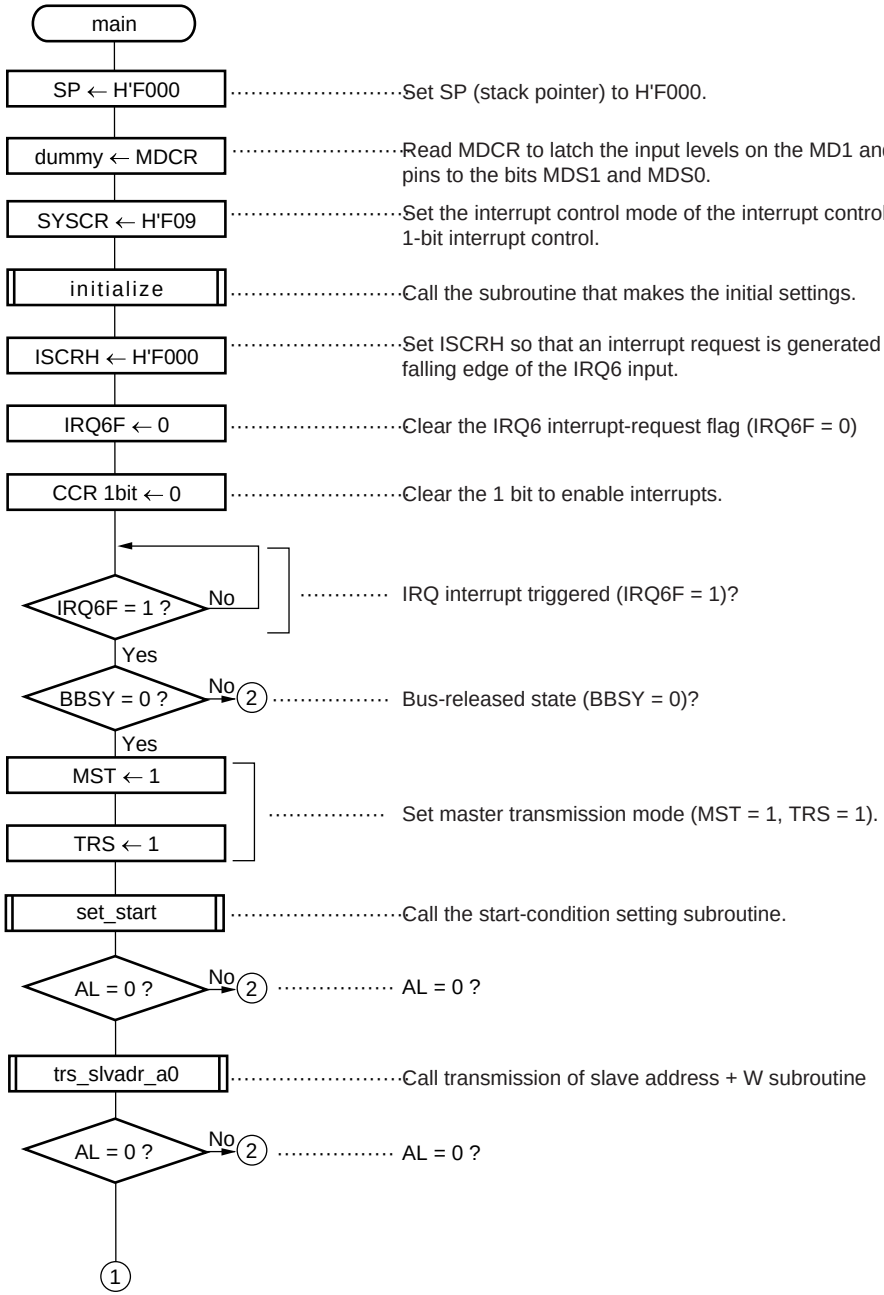
Table 4.39 describes the usage of RAM in this example of a task.

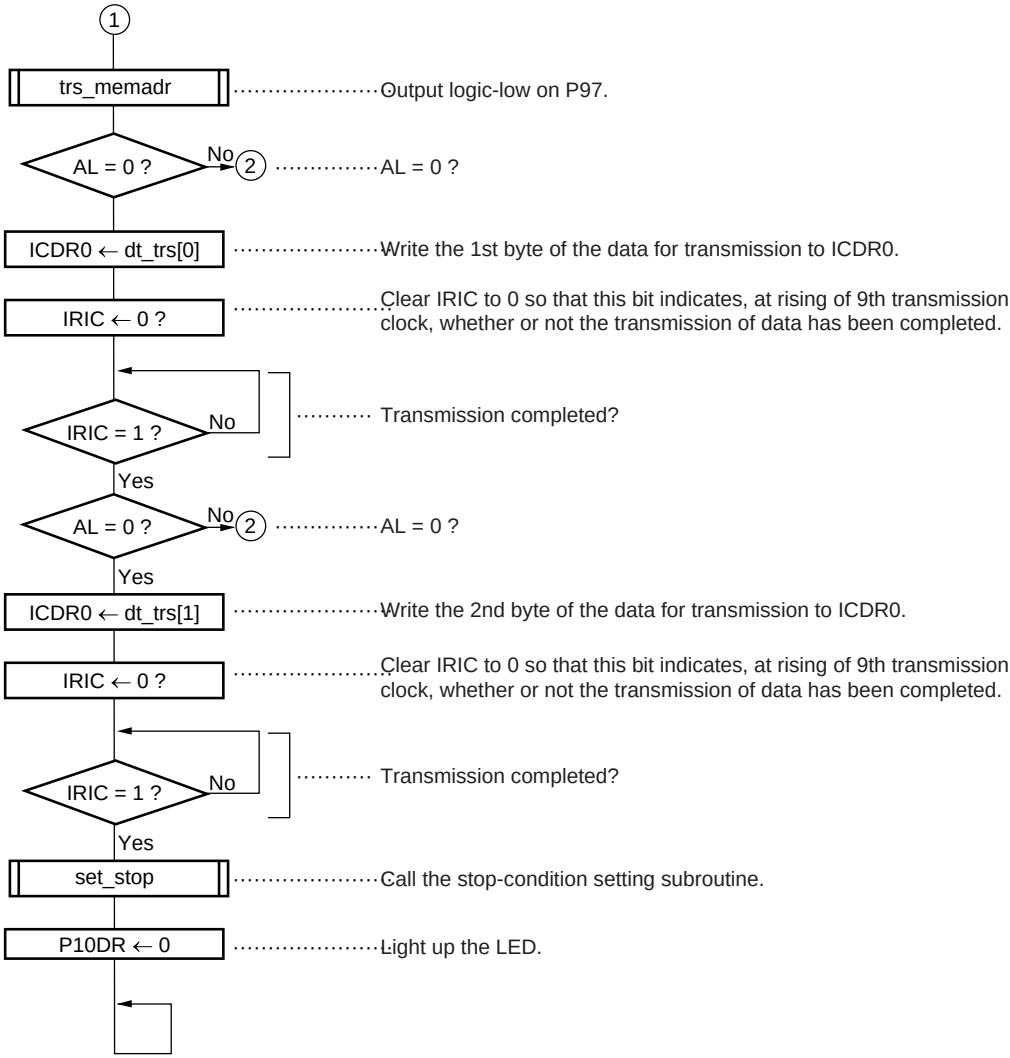
Table 4.39 Description of RAM Usage

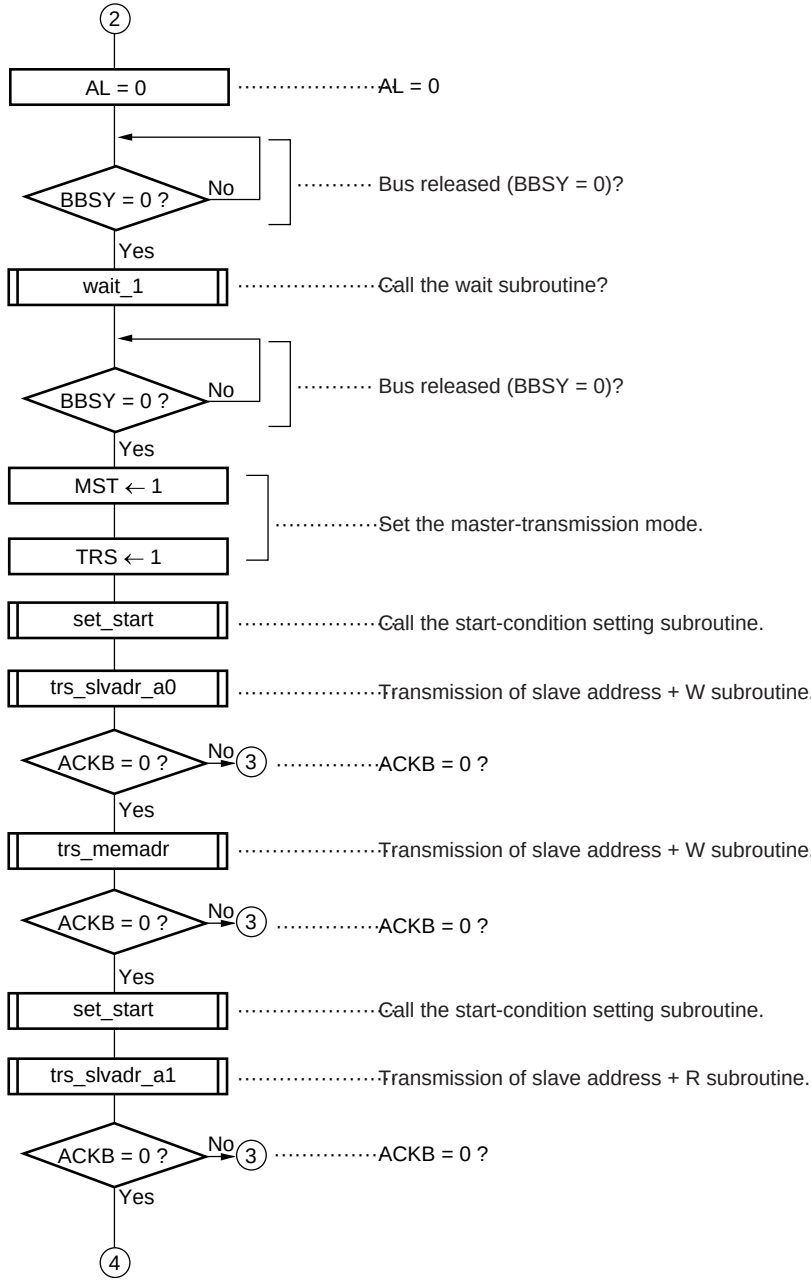
Label	Function	Size	Address	Name of Using Module
dt_rec[0]	Stores the 1st byte of received data	1 byte	H'E100	main, initialize
dt_rec[1]	Stores the 2nd byte of received data	1 byte	H'E101	main, initialize

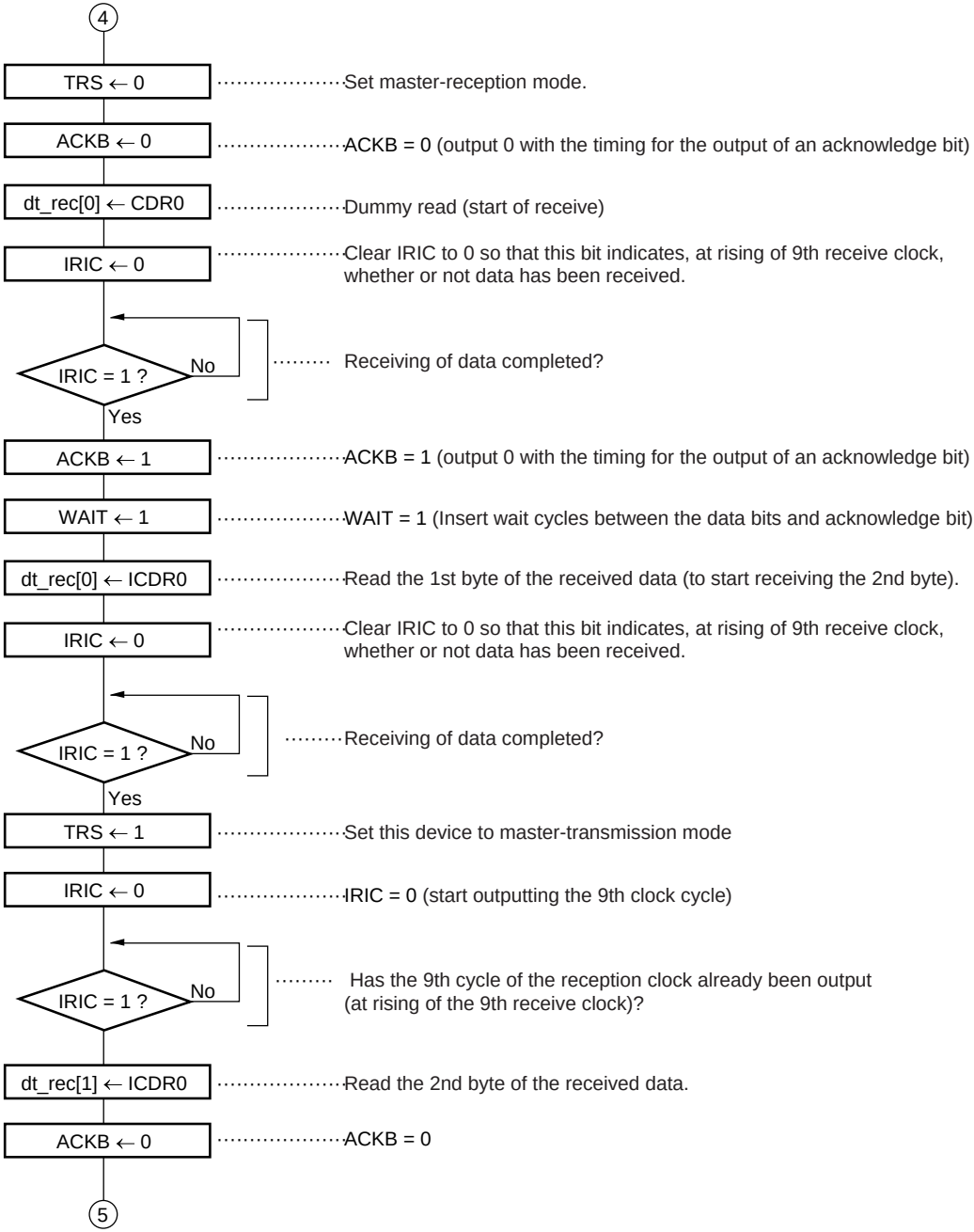
4.11.4 Flowchart

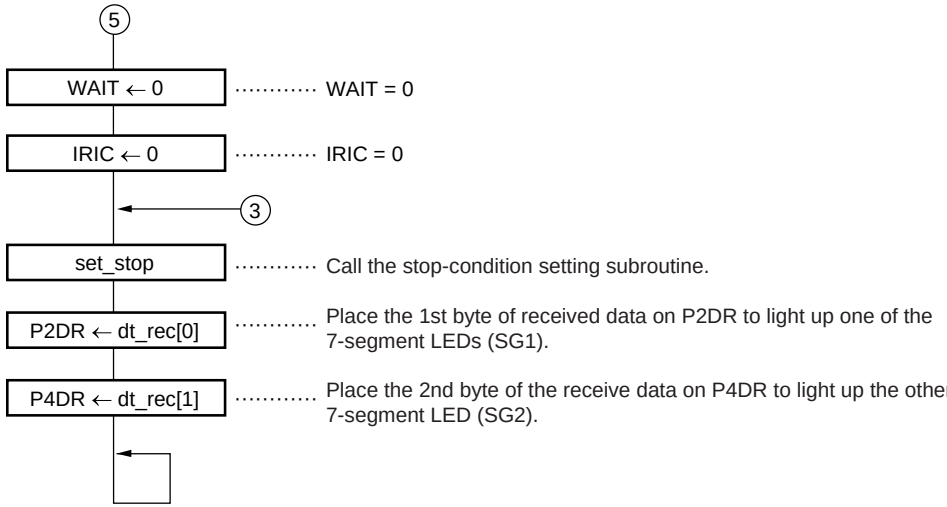
(1) Main routine



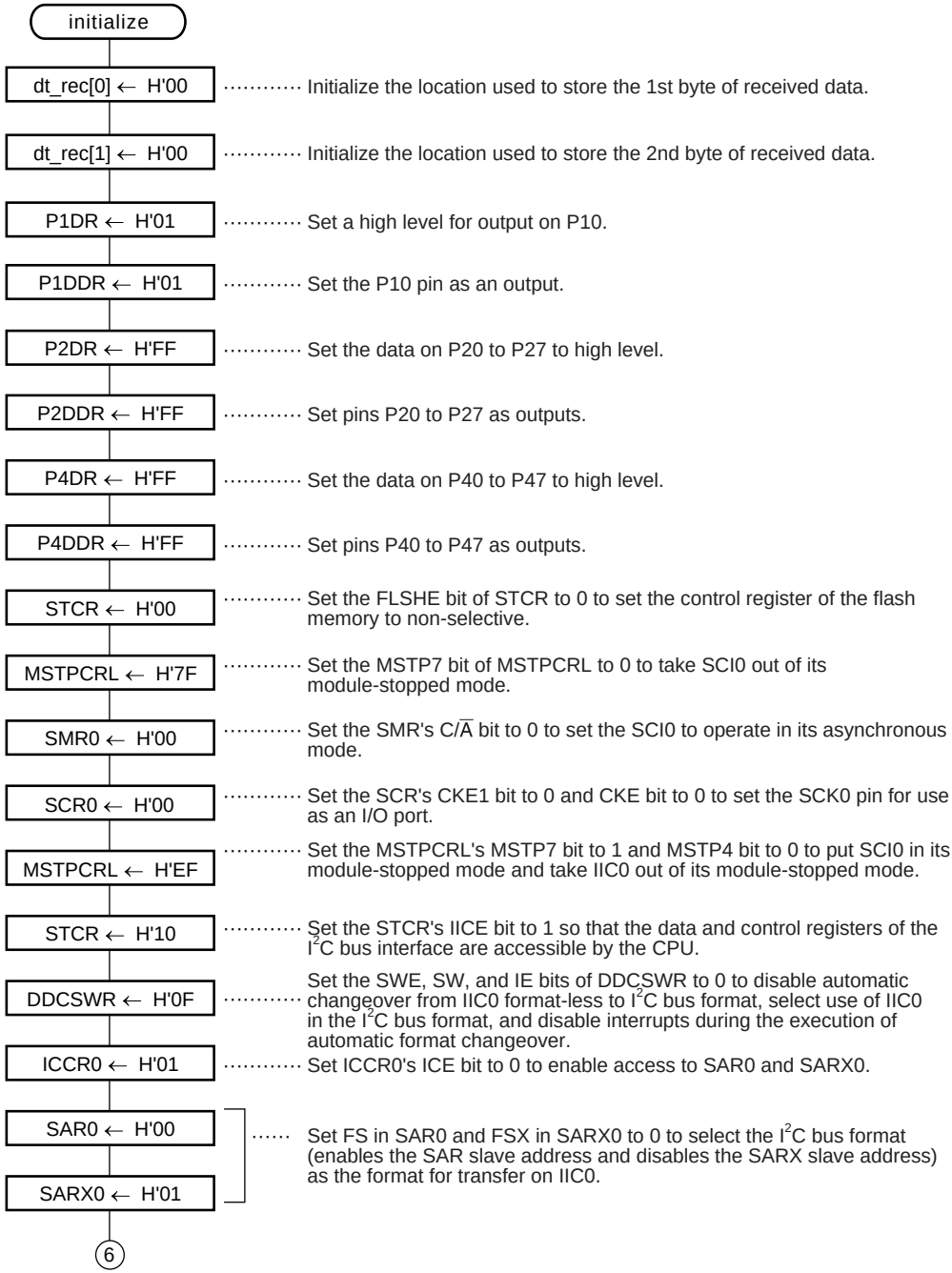

1
trs_memadr
Output logic-low on P97.
AL = 0 ?
No
2
AL = 0 ?
ICDR0 ← dt_trs[0]
Write the 1st byte of the data for transmission to ICDR0.
IRIC ← 0 ?
Clear IRIC to 0 so that this bit indicates, at rising of 9th transmission clock, whether or not the transmission of data has been completed.
IRIC = 1 ?
No
Transmission completed?
Yes
AL = 0 ?
No
2
AL = 0 ?
Yes
ICDR0 ← dt_trs[1]
Write the 2nd byte of the data for transmission to ICDR0.
IRIC ← 0 ?
Clear IRIC to 0 so that this bit indicates, at rising of 9th transmission clock, whether or not the transmission of data has been completed.
IRIC = 1 ?
No
Transmission completed?
Yes
set_stop
Call the stop-condition setting subroutine.
P10DR ← 0
Light up the LED.

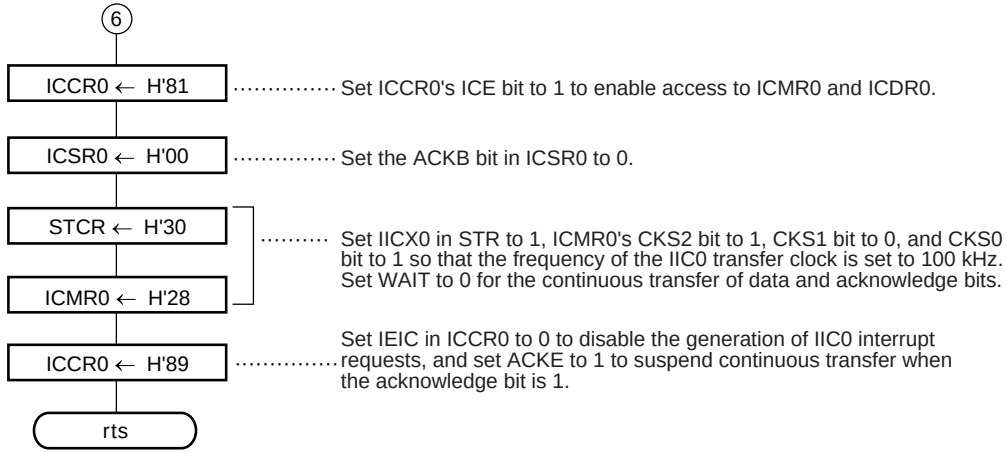

2
AL = 0
AL = 0
BBSY = 0 ?
No
Bus released (BBSY = 0)?
Yes
wait_1
Call the wait subroutine?
BBSY = 0 ?
No
Bus released (BBSY = 0)?
Yes
MST ← 1
TRS ← 1
Set the master-transmission mode.
set_start
Call the start-condition setting subroutine.
trs_slvadr_a0
Transmission of slave address + W subroutine.
ACKB = 0 ?
No
3
ACKB = 0 ?
Yes
trs_memadr
Transmission of slave address + W subroutine.
ACKB = 0 ?
No
3
ACKB = 0 ?
Yes
set_start
Call the start-condition setting subroutine.
trs_slvadr_a1
Transmission of slave address + R subroutine.
ACKB = 0 ?
No
3
ACKB = 0 ?
Yes
4


4
TRS ← 0
Set master-reception mode.
ACKB ← 0
ACKB = 0 (output 0 with the timing for the output of an acknowledge bit)
dt_rec[0] ← CDR0
Dummy read (start of receive)
IRIC ← 0
Clear IRIC to 0 so that this bit indicates, at rising of 9th receive clock, whether or not data has been received.
IRIC = 1 ?
No
Yes
Receiving of data completed?
ACKB ← 1
ACKB = 1 (output 0 with the timing for the output of an acknowledge bit)
WAIT ← 1
WAIT = 1 (Insert wait cycles between the data bits and acknowledge bit)
dt_rec[0] ← ICDR0
Read the 1st byte of the received data (to start receiving the 2nd byte).
IRIC ← 0
Clear IRIC to 0 so that this bit indicates, at rising of 9th receive clock, whether or not data has been received.
IRIC = 1 ?
No
Yes
Receiving of data completed?
TRS ← 1
Set this device to master-transmission mode
IRIC ← 0
IRIC = 0 (start outputting the 9th clock cycle)
IRIC = 1 ?
No
Has the 9th cycle of the reception clock already been output (at rising of the 9th receive clock)?
dt_rec[1] ← ICDR0
Read the 2nd byte of the received data.
ACKB ← 0
ACKB = 0
5

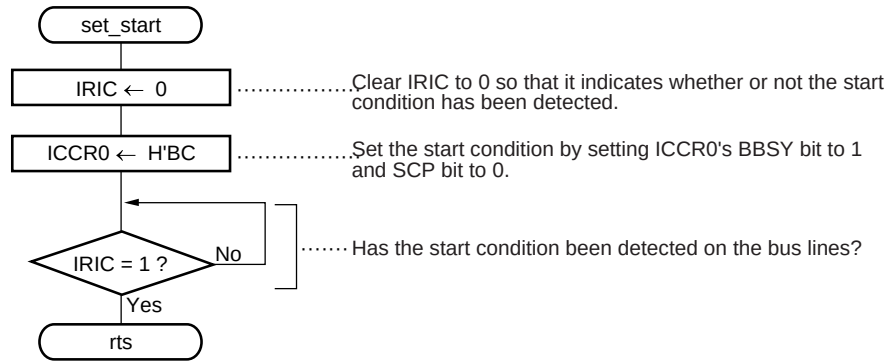

5
WAIT ← 0
WAIT = 0
IRIC ← 0
IRIC = 0
3
set_stop
Call the stop-condition setting subroutine.
P2DR ← dt_rec[0]
Place the 1st byte of received data on P2DR to light up one of the 7-segment LEDs (SG1).
P4DR ← dt_rec[1]
Place the 2nd byte of the receive data on P4DR to light up the other 7-segment LED (SG2).

(2) Initial Setting Subroutine

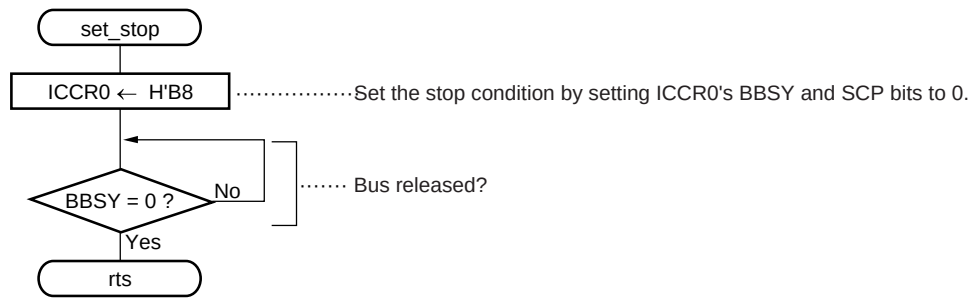




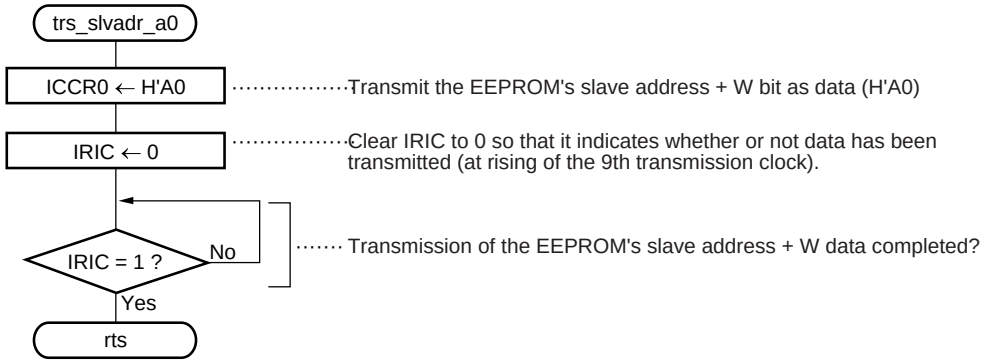
(3) Subroutine for Setting the Start Condition



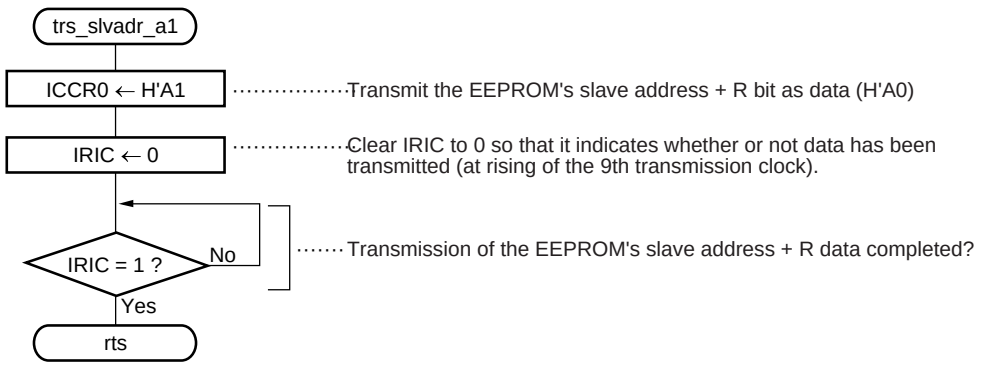
(4) Subroutine for Setting the Stop Condition



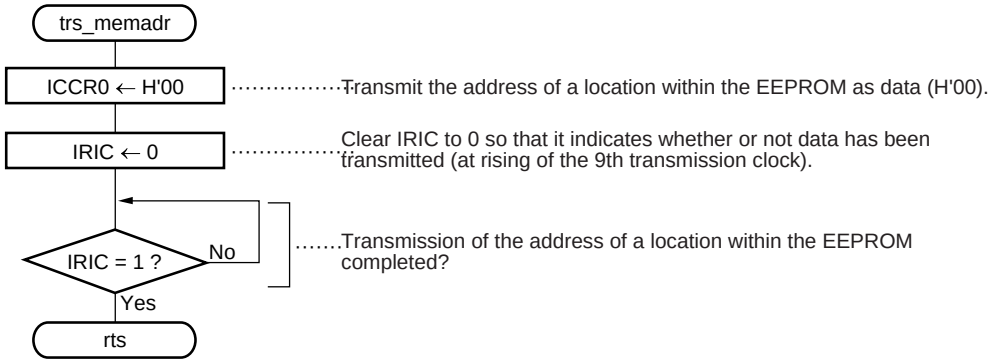
(5) Subroutine for transmitting the slave address + W bit



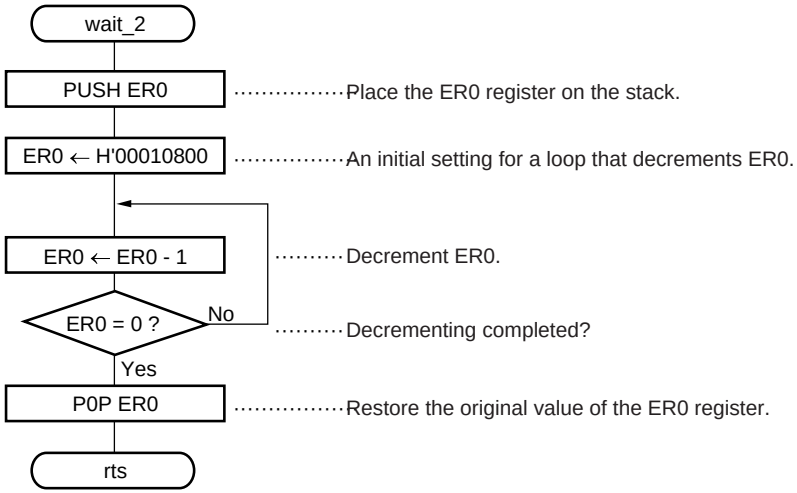
(6) Subroutine for transmitting the slave address + R bit



(7) Subroutine for transmitting the location within the EEPROM



(8) WAIT2 subroutine



4.11.5 Master-1 program List

```
/*****************************************************
*  H8S/2138 IIC bus application note                 *
*    10.Multi master transmit/receive 1              *
*                       File name   :   Mltx1.c       *
*                       Fai         :   20MHz         *
*                       Mode        :   3             *
*****************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*****************************************************
*  Prototype                                         *
*****************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* RAM & port & IIC0 initialize */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_slvadr_a1(void);                           /* Slave address + R data transmit */
void trs_memadr(void);                              /* EEPROM memry address data transmit */
void wait_1(void);                                  /* EEPROM write cycle(10ms) wait */

/*****************************************************
*  Data table                                        *
*****************************************************/
const unsigned char dt_trs[2] =                     /* Transmit data (2 byte) */
{
    0xf9,                                           /* Master 1 1st transmit data */
    0xc0                                            /* Master 1 2nd transmit data */
};

/*****************************************************
*  RAM allocation                                    *
*****************************************************/
```

```
#pragma section ramarea
unsigned char dt_rec[2];                        /* Receive data store area (2 byte) */
#pragma section

/*****************************************************
*  main : Main routine                               *
*****************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                     ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                         /* MCU mode set */

     SYSCR.BYTE = 0x09;                         /* Interrupt control mode set */
     initialize();                              /* Initialize */

     INTC.ISCR.BYTE.H = 0x10;                   /* IRQ6 edge sense set (faling edge) */
     INTC.ISR.BIT.IRQ6F = 0;                    /* IRQ6 interrupt request flag clear */

     set_imask_ccr(0);                          /* Interrupt enable */

     while(INTC.ISR.BIT.IRQ6F == 0);            /* IRQ interrupt switch on ? */
     INTC.ISR.BIT.IRQ6F = 0;                    /* IRQ6F = 0 */

     if(IIC0.ICCR.BIT.BBSY == 0)                /* Bus empty (BBSY=0) ? */
     {
          IIC0.ICCR.BIT.MST = 1;                /* Master transmit mode set */
          IIC0.ICCR.BIT.TRS = 1;                /* MST = 1, TRS = 1 */
          set_start();                          /* Start condition set */
          if(IIC0.ICSR.BIT.AL == 0)             /* AL = 0 ? */
          {
               trs_slvadr_a0();                 /* Slave address + W data transmit */
               if(IIC0.ICSR.BIT.AL == 0)        /* AL = 0 ? */
               {
                    trs_memadr();               /* EEPROM memory address transmit */
```

```
                if(IIC0.ICSR.BIT.AL == 0)          /* AL = 0 ? */
                {
                     IIC0.ICDR = dt_trs[0];        /* 1st transmit data write */
                     IIC0.ICCR.BIT.IRIC = 0;       /* IRIC = 0 */
                     while(IIC0.ICCR.BIT.IRIC == 0); /* Transmit end (IRIC=1) ? */
                     if(IIC0.ICSR.BIT.AL == 0)     /* AL = 0 ? */
                     {
                          IIC0.ICDR = dt_trs[1];   /* 2nd transmit data write */
                          IIC0.ICCR.BIT.IRIC = 0;  /* IRIC = 0 */
                          while(IIC0.ICCR.BIT.IRIC == 0); /* Transmit end (IRIC=1) ? */

                          set_stop();              /* Stop condition set */
                          P1.DR.BIT.B0 = 0;        /* LED on */
                          while(1);                /* End */
                     }
                }
           }
      }
}

IIC0.ICSR.BIT.AL = 0;                            /* AL= 0 */
while(IIC0.ICCR.BIT.BBSY == 1);                  /* Transmit end (BBSY=0) ? */
wait_1();                                        /* 10ms wait */

while(IIC0.ICCR.BIT.BBSY == 1);                  /* Bus empty (BBSY=0) ? */
IIC0.ICCR.BIT.MST = 1;                           /* Master transmit mode set */
IIC0.ICCR.BIT.TRS = 1;                           /* MST = 1, TRS = 1 */

set_start();                                     /* Start condition set */
trs_slvadr_a0();                                 /* Slave address + W data transmit */
if(IIC0.ICSR.BIT.ACKB == 0)                      /* ACKB = 0 ? */
{
     trs_memadr();                               /* EEPROM memory address data transmit */
     if(IIC0.ICSR.BIT.ACKB == 0)                 /* ACKB = 0 ? */
     {
          set_start();                           /* Re-start condition set */
          trs_slvadr_a1();                       /* Slave address + R data transmit */
```

```
            if(IIC0.ICSR.BIT.ACKB == 0)            /* ACKB = 0 ? */
            {
                IIC0.ICCR.BIT.TRS = 0;             /* Master receive mode set (TRS=0) */
                IIC0.ICSR.BIT.ACKB = 0;            /* ACKB = 0 */
                dt_rec[0] = IIC0.ICDR;             /* Dummy read */
                IIC0.ICCR.BIT.IRIC = 0;            /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);    /* Receive end (IRIC=1) ? */

                IIC0.ICSR.BIT.ACKB = 1;            /* ACKB = 1 */
                IIC0.ICMR.BIT.WAIT = 1;            /* WAIT = 1 */
                dt_rec[0] = IIC0.ICDR;             /* 1st receive data read */
                IIC0.ICCR.BIT.IRIC = 0;            /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);    /* Receive end (IRIC=1) ? */

                IIC0.ICCR.BIT.TRS = 1;             /* Master transmit mode set (TRS=1) */
                IIC0.ICCR.BIT.IRIC = 0;            /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);    /* IRIC = 1 ? */

                dt_rec[1] = IIC0.ICDR;             /* 2nd receive data read */
                IIC0.ICSR.BIT.ACKB = 0;            /* ACKB = 0 */
                IIC0.ICMR.BIT.WAIT = 0;            /* WAIT = 0 */
                IIC0.ICCR.BIT.IRIC = 0;            /* IRIC = 0 */
            }
        }
    }
    set_stop();                                    /* Stop condition set */

    P2.DR.BYTE = dt_rec[0];                        /* SG1 on */
    P4.DR.BYTE = dt_rec[1];                        /* SG2 on */

    while(1);                                      /* End */
}

/*****************************************************
*  initialize : RAM & Port & IIC0 Initialize         *
*****************************************************/
void initialize(void)
```

```
{
    dt_rec[0] = 0x00;                           /* Receive data store area initialize */
    dt_rec[1] = 0x00;

    P1.DR.BYTE = 0x01;                          /* Port 1 initialize */
    P1.DDR = 0x01;
    P2.DR.BYTE = 0xff;                          /* Port 2 initialize */
    P2.DDR = 0xff;
    P4.DR.BYTE = 0xff;                          /* Port 4 initialize */
    P4.DDR = 0xff;

    STCR.BYTE = 0x00;                           /* FLSHE = 0 */
    MSTPCR.BYTE.L = 0x7f;                       /* SCI0 module stop mode reset */
    SCI0.SMR.BYTE = 0x00;                       /* SCL0 pin function set */
    SCI0.SCR.BYTE = 0x00;
    MSTPCR.BYTE.L = 0xef;                       /* IIC0 module stop mode reset */
    STCR.BYTE = 0x10;                           /* IICE = 1 */
    DDCSWR.BYTE = 0x0f;                         /* IIC bus format initialize */
    IIC0.ICCR.BYTE = 0x01;                      /* ICE = 0 */
    IIC0.SAR.BYTE = 0x38;                       /* FS = 0 */
    IIC0.SARX.BYTE = 0x01;                      /* FSX = 1 */
    IIC0.ICCR.BYTE = 0x81;                      /* ICE = 1 */
    IIC0.ICSR.BYTE = 0x00;                      /* ACKB = 0 */
    STCR.BYTE = 0x30;                           /* IICX0 = 1 */
    IIC0.ICMR.BYTE = 0x28;                      /* Transfer rate = 100kHz */
    IIC0.ICCR.BYTE = 0x89;                      /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
void set_start(void)
{
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    IIC0.ICCR.BYTE = 0xbc;                      /* Start condition set (BBSY=1,SCP=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Start condition set (IRIC=1) ? */
}
```

```
/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
     IIC0.ICCR.BYTE = 0xb8;                          /* Stop condition set (BBSY=0,SCP=0) */
     while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
}

/*****************************************************
*  trs_slvadr_a0 : Slave addres + W data transmit    *
*****************************************************/
void trs_slvadr_a0(void)
{
     IIC0.ICDR = 0xa0;                               /* Slave address + W data(H'A0) write */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_slvadr_a1 : Slave addres + R data transmit    *
*****************************************************/
void trs_slvadr_a1(void)
{
     IIC0.ICDR = 0xa1;                               /* Slave address + R data(H'A1) write */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}
/*****************************************************
* trs_memadr : EEPROM memory address data transmit   *
*****************************************************/
void trs_memadr(void)
{
     IIC0.ICDR = 0x00;                               /* EEPROM memory address data(H'00) write */
     IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
     while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}
```

```
/*****************************************************
*   wait_1 : 10ms wait                                *
*****************************************************/
void wait_1(void)
{
#pragma asm
         push.l  er0                                ;Push ER0
         mov.l   #h'00010800,er0                    ;Decrement data set
wait1_1:
         dec.l   #1,er0                             ;Decrement
         bne     wait1_1                            ;Decrement end ?
         pop.l   er0                                ;Pop ER0
#pragma endasm
}
```

4.11.6 Master-2 program List

```
/*****************************************************
*  H8S/2138 IIC bus application note                 *
*    10.Multi master transmit/receive 2              *
*                       File name   :   Mltx2.c       *
*                       Fai         :   20MHz         *
*                       Mode        :   3             *
*****************************************************/
#include <stdio.h>
#include <machine.h>
#include "2138s.h"

/*****************************************************
*  Prototype                                         *
*****************************************************/
void main(void);                                    /* Main routine */
void initialize(void);                              /* RAM & port & IIC0 initialize */
void set_start(void);                               /* Start condition set */
void set_stop(void);                                /* Stop condition set */
void trs_slvadr_a0(void);                           /* Slave address + W data transmit */
void trs_slvadr_a1(void);                           /* Slave address + R data transmit */
void trs_memadr(void);                              /* EEPROM memry address data transmit */
void wait_1(void);                                  /* EEPROM write cycle(10ms) wait */

/*****************************************************
*  Data table                                        *
*****************************************************/
const unsigned char dt_trs[2] =                     /* Transmit data (2 byte) */
{
    0xa4,                                           /* Master 2 1st transmit data */
    0xc0                                            /* Master 2 2nd transmit data */
};
```

```
/*****************************************************
*  RAM allocation                                    *
*****************************************************/
#pragma section ramarea
unsigned char dt_rec[2];                             /* Receive data store area (2 byte) */
#pragma section

/*****************************************************
*  main : Main routine                               *
*****************************************************/
void main(void)
#pragma asm
         mov.l   #h'f000,sp                          ;Stack pointer initialize
#pragma endasm
{
     unsigned char dummy;
     dummy = MDCR.BYTE;                              /* MCU mode set */

     SYSCR.BYTE = 0x09;                              /* Interrupt control mode set */
     initialize();                                   /* Initialize */

     INTC.ISCR.BYTE.H = 0x10;                        /* IRQ6 edge sense set (faling edge) */
     INTC.ISR.BIT.IRQ6F = 0;                         /* IRQ6 interrupt request flag clear */

     set_imask_ccr(0);                               /* Interrupt enable */

     while(INTC.ISR.BIT.IRQ6F == 0);                 /* IRQ interrupt switch on ? */
     INTC.ISR.BIT.IRQ6F = 0;                         /* IRQ6F = 0 */

     if(IIC0.ICCR.BIT.BBSY == 0)                     /* Bus empty (BBSY=0) ? */
     {
          IIC0.ICCR.BIT.MST = 1;                     /* Master transmit mode set */
          IIC0.ICCR.BIT.TRS = 1;                     /* MST = 1, TRS = 1 */
          set_start();                               /* Start condition set */
          if(IIC0.ICSR.BIT.AL == 0)                  /* AL = 0 ? */
          {
               trs_slvadr_a0();                      /* Slave address + W data transmit */
```

```
        if(IIC0.ICSR.BIT.AL == 0)              /* AL = 0 ? */
        {
            trs_memadr();                      /* EEPROM memory address transmit */
            if(IIC0.ICSR.BIT.AL == 0)          /* AL = 0 ? */
            {
                IIC0.ICDR = dt_trs[0];         /* 1st transmit data write */
                IIC0.ICCR.BIT.IRIC = 0;        /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0); /* Transmit end (IRIC=1) ? */
                if(IIC0.ICSR.BIT.AL == 0)      /* AL = 0 ? */
                {
                    IIC0.ICDR = dt_trs[1];     /* 2nd transmit data write */
                    IIC0.ICCR.BIT.IRIC = 0;    /* IRIC = 0 */
                    while(IIC0.ICCR.BIT.IRIC == 0); /* Transmit end (IRIC=1) ? */

                    set_stop();                /* Stop condition set */
                    P1.DR.BIT.B0 = 0;          /* LED on */
                    while(1);                  /* End */
                }
            }
        }
    }
}

IIC0.ICSR.BIT.AL = 0;                          /* AL= 0 */
while(IIC0.ICCR.BIT.BBSY == 1);                /* Transmit end (BBSY=0) ? */
wait_1();                                      /* 10ms wait */

while(IIC0.ICCR.BIT.BBSY == 1);                /* Bus empty (BBSY=0) ? */
IIC0.ICCR.BIT.MST = 1;                         /* Master transmit mode set */
IIC0.ICCR.BIT.TRS = 1;                         /* MST = 1, TRS = 1 */

set_start();                                   /* Start condition set */
trs_slvadr_a0();                               /* Slave address + W data transmit */
if(IIC0.ICSR.BIT.ACKB == 0)                    /* ACKB = 0 ? */
{
    trs_memadr();                              /* EEPROM memory address data transmit */
    if(IIC0.ICSR.BIT.ACKB == 0)                /* ACKB = 0 ? */
```

```
        {
            set_start();                            /* Re-start condition set */
            trs_slvadr_a1();                        /* Slave address + R data transmit */
            if(IIC0.ICSR.BIT.ACKB == 0)             /* ACKB = 0 ? */
            {
                IIC0.ICCR.BIT.TRS = 0;              /* Master receive mode set (TRS=0) */
                IIC0.ICSR.BIT.ACKB = 0;             /* ACKB = 0 */
                dt_rec[0] = IIC0.ICDR;              /* Dummy read */
                IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);     /* Receive end (IRIC=1) ? */

                IIC0.ICSR.BIT.ACKB = 1;             /* ACKB = 1 */
                IIC0.ICMR.BIT.WAIT = 1;             /* WAIT = 1 */
                dt_rec[0] = IIC0.ICDR;              /* 1st receive data read */
                IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);     /* Receive end (IRIC=1) ? */

                IIC0.ICCR.BIT.TRS = 1;              /* Master transmit mode set (TRS=1) */
                IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
                while(IIC0.ICCR.BIT.IRIC == 0);     /* IRIC = 1 ? */

                dt_rec[1] = IIC0.ICDR;              /* 2nd receive data read */
                IIC0.ICSR.BIT.ACKB = 0;             /* ACKB = 0 */
                IIC0.ICMR.BIT.WAIT = 0;             /* WAIT = 0 */
                IIC0.ICCR.BIT.IRIC = 0;             /* IRIC = 0 */
            }
        }
    }
    set_stop();                                     /* Stop condition set */

    P2.DR.BYTE = dt_rec[0];                         /* SG1 on */
    P4.DR.BYTE = dt_rec[1];                         /* SG2 on */

    while(1);                                       /* End */
}
```

```
/*****************************************************
*  initialize : RAM & Port & IIC0 Initialize         *
*****************************************************/
void initialize(void)
{
     dt_rec[0] = 0x00;                              /* Receive data store area initialize */
     dt_rec[1] = 0x00;

     P1.DR.BYTE = 0x01;                             /* Port 1 initialize */
     P1.DDR = 0x01;
     P2.DR.BYTE = 0xff;                             /* Port 2 initialize */
     P2.DDR = 0xff;
     P4.DR.BYTE = 0xff;                             /* Port 4 initialize */
     P4.DDR = 0xff;

     STCR.BYTE = 0x00;                              /* FLSHE = 0 */
     MSTPCR.BYTE.L = 0x7f;                          /* SCI0 module stop mode reset */
     SCI0.SMR.BYTE = 0x00;                          /* SCL0 pin function set */
     SCI0.SCR.BYTE = 0x00;
     MSTPCR.BYTE.L = 0xef;                          /* IIC0 module stop mode reset */
     STCR.BYTE = 0x10;                              /* IICE = 1 */
     DDCSWR.BYTE = 0x0f;                            /* IIC bus format initialize */
     IIC0.ICCR.BYTE = 0x01;                         /* ICE = 0 */
     IIC0.SAR.BYTE = 0x38;                          /* FS = 0 */
     IIC0.SARX.BYTE = 0x01;                         /* FSX = 1 */
     IIC0.ICCR.BYTE = 0x81;                         /* ICE = 1 */
     IIC0.ICSR.BYTE = 0x00;                         /* ACKB = 0 */
     STCR.BYTE = 0x30;                              /* IICX0 = 1 */
     IIC0.ICMR.BYTE = 0x28;                         /* Transfer rate = 100kHz */
     IIC0.ICCR.BYTE = 0x89;                         /* IEIC = 0, ACKE = 1 */
}

/*****************************************************
*  set_start : Start condition set                   *
*****************************************************/
void set_start(void)
{
```

```
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    IIC0.ICCR.BYTE = 0xbc;                          /* Start condition set (BBSY=1,SCP=0) */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Start condition set (IRIC=1) ? */
}
/*****************************************************
*  set_stop : Stop condition set                     *
*****************************************************/
void set_stop(void)
{
    IIC0.ICCR.BYTE = 0xb8;                          /* Stop condition set (BBSY=0,SCP=0) */
    while(IIC0.ICCR.BIT.BBSY == 1);                 /* Bus empty (BBSY=0) ? */
}

/*****************************************************
*  trs_slvadr_a0 : Slave addres + W data transmit    *
*****************************************************/
void trs_slvadr_a0(void)
{
    IIC0.ICDR = 0xa0;                               /* Slave address + W data(H'A0) write */
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}

/*****************************************************
*  trs_slvadr_a1 : Slave addres + R data transmit    *
*****************************************************/
void trs_slvadr_a1(void)
{
    IIC0.ICDR = 0xa1;                               /* Slave address + R data(H'A1) write */
    IIC0.ICCR.BIT.IRIC = 0;                         /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);                 /* Transmit end (IRIC=1) ? */
}
/*****************************************************
* trs_memadr : EEPROM memory address data transmit   *
*****************************************************/
void trs_memadr(void)
{
```

```
    IIC0.ICDR = 0x00;                           /* EEPROM memory address data(H'00) write */
    IIC0.ICCR.BIT.IRIC = 0;                     /* IRIC = 0 */
    while(IIC0.ICCR.BIT.IRIC == 0);             /* Transmit end (IRIC=1) ? */
}
/*****************************************************
*  wait_1 : 10ms wait                                *
*****************************************************/
void wait_1(void)
{
#pragma asm
        push.l  er0                             ;Push ER0
        mov.l   #h'00010800,er0                 ;Decrement data set
wait1_1:
        dec.l   #1,er0                          ;Decrement
        bne     wait1_1                         ;Decrement end ?
        pop.l   er0                             ;Pop ER0
#pragma endasm
}
```

I^2C Bus Interface Application Note

Publication Date: 1st Edition, March 1994
2nd Edition, November 2001
Published by: Business Planning Division
Semiconductor & Integrated Circuits
Hitachi, Ltd.
Edited by: Technical Documentation Group
Hitachi Kodaira Semiconductor Co., Ltd.
Copyright © Hitachi, Ltd., 1994. All rights reserved. Printed in Japan.