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.

SH7145 Group

I²C Bus Interface in Combined Use with DTC

Introduction

This application note describes how to implement automatic execution of transmission and reception of data via the I²C bus (Inter IC Bus) interface through the use of DTC (Data Transfer Controller) of the SH7145F. The master device is the SH7145F, to which EEPROM is connected as a slave device.

Target Device

SH7145F

Contents

1. I²C Bus Single-Master Transmission in Combined Use with DTC

1.1 Specifications

Data transmission is performed via an I²C bus (Inter IC Bus) in combined use with the Data Transfer Controller (DTC) of the SH7145F.

As shown in figure 1.1, the master device is the SH7145F, and an EEPROM (HN58X2464, 64 kbits, 8 words × 8 bits) is connected as the slave device. The I²C bus interface of the SH7145F is used to write 10 bytes of data stored in the on-chip RAM to EEPROM. In this process, the DTC is used to transfer the write data from the on-chip RAM to the data register of the I²C module.

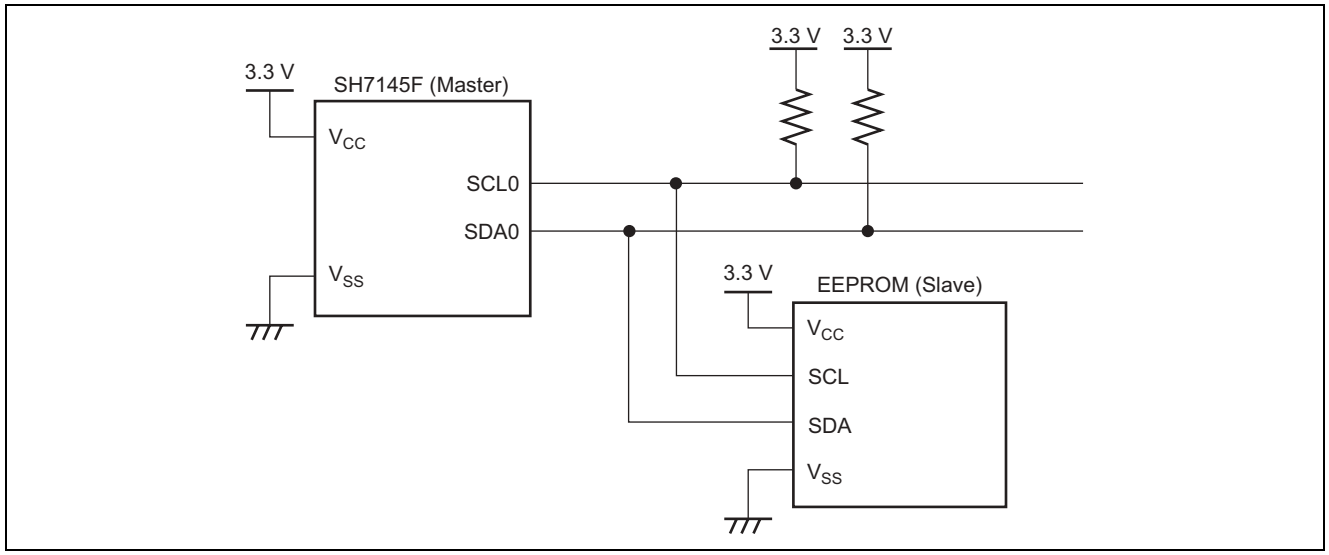


Figure 1.1 Connection of EEPROM to SH7145F

Figure 1.2 shows the data transfer by the DTC. Table 1.1 shows the DTC settings, and table 1.2 shows the I²C bus interface settings.

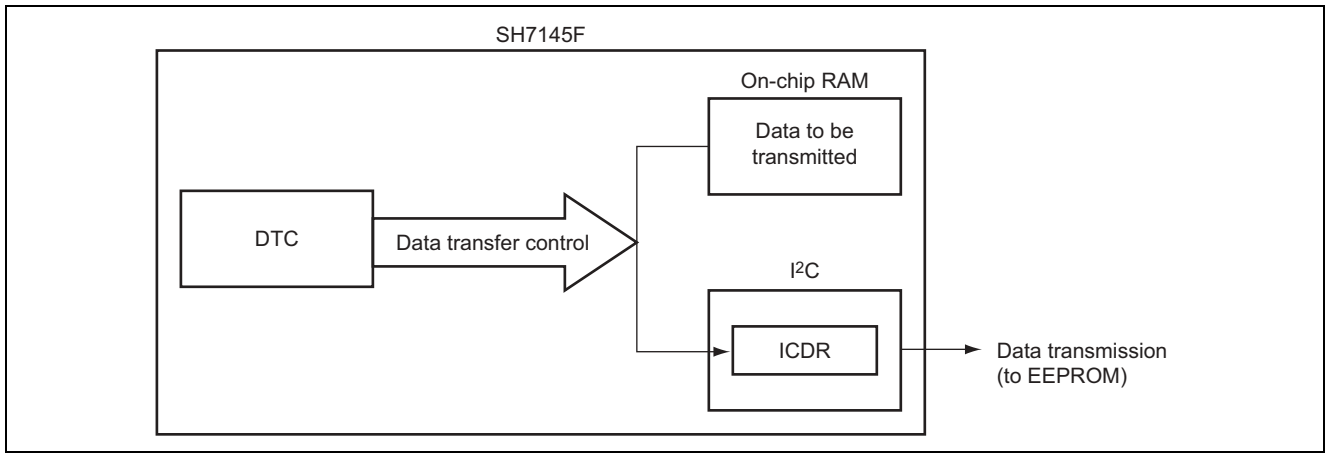


Figure 1.2 Data Transfer by DTC

Table 1.1 DTC Settings

Condition	Setting
Transfer mode	Normal mode
Transfer data size	1 byte
Number of transfers (DTCRA)	13
Transfer source	On-chip RAM
Transfer destination	ICDR (I^2C bus data register: H'FFFF880E)
Transfer source address	Incremented after transfer.
Transfer destination address	Fixed address
Transfer information storage address	H'FFFFF000
Transfer start source	Started by an I^2C interrupt (ICI).
Interrupt	CPU interrupts enabled only when the specified data transfer has ended

Table 1.2 I^2C Settings

Item	Setting
Operation	Master transmission
Transfer clock	156 kHz (Pϕ = 40 MHz)
Number of bits in data	9 bits (including ACK)
Wait between data and ACK	None
Interrupt	Enabled
ACK judgment	Transfer is discontinued when ACK = 1 received

1.2 Description of Functions

In this sample task, the I²C bus (Inter IC Bus) is used to write data to an EEPROM. Write data are transferred from the on-chip RAM to the data register of the I²C bus interface using the DTC (Data Transfer Controller).

1.2.1 I²C Bus (Inter IC Bus) Interface

This interface conforms to the I²C bus interface format advocated by Philips Corp., and is provided with subset functions. Figure 1.3 is a block diagram of the I²C module, which is explained below.

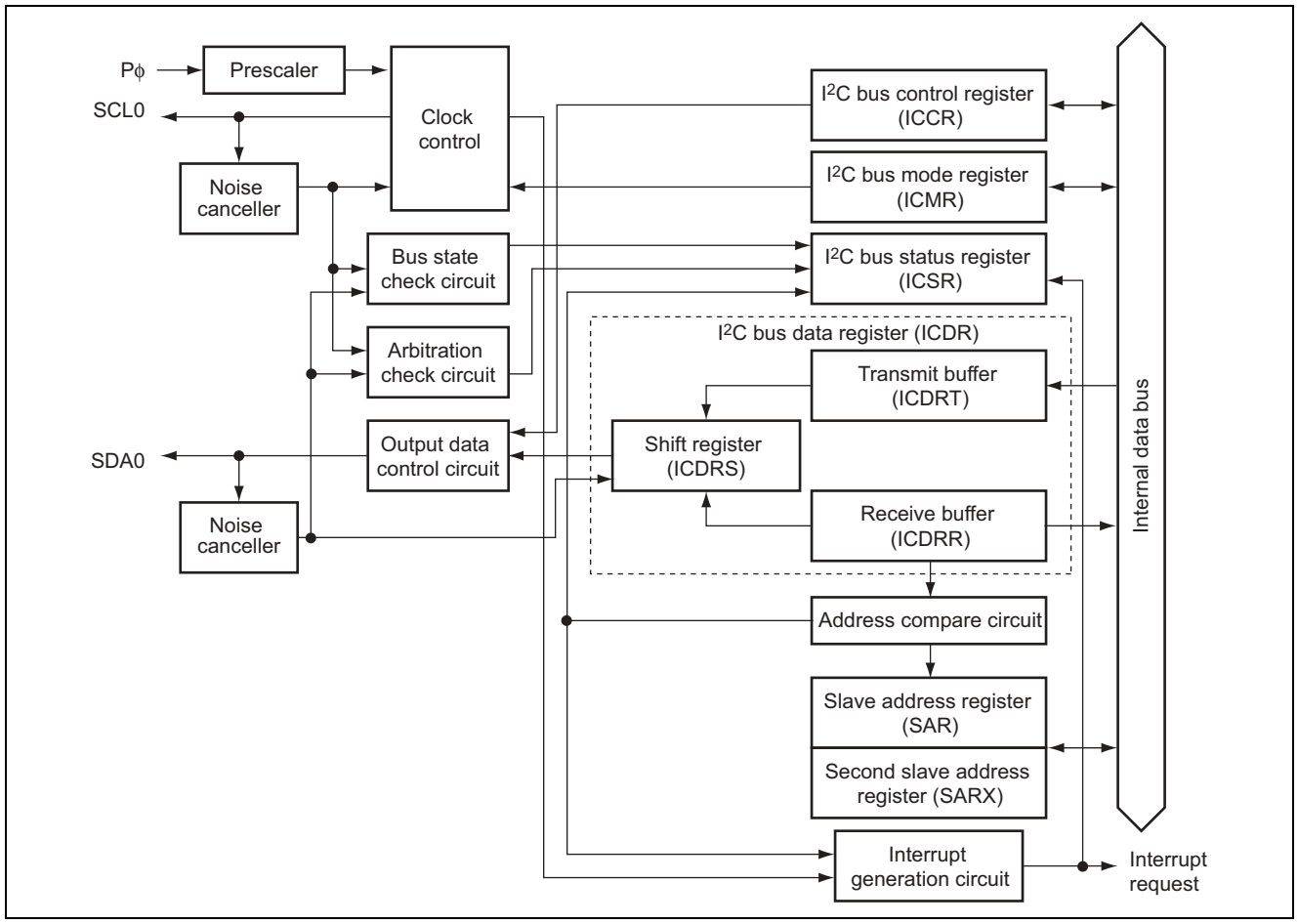


Figure 1.3 I²C Module Block Diagram

- The I²C bus control register (ICCR) sets operation parameters for the I²C bus interface.
- The I²C bus mode register (ICMR) selects MSB- or LSB-first, and selects wait insertion, transfer clock, and the number of bits for transfer. ICMR can be accessed only when the ICE bit in ICCR is set to 1.
- The I²C bus status register (ICSR) is used to check flags and to check and control acknowledgement.
- The I²C bus data register (ICDR) is a data register used for both transmission and reception. ICDR is internally divided into a shift register (ICDRS), receive buffer (ICDRR), and transmit buffer (ICDRT). During transmission, when transmit data is written to ICDR, the data is stored to ICDRT, and it is automatically transferred to ICDRS after transmission of the preceding data. When ICDRS receives data, the received data is automatically transferred to ICDRR and can be read by reading ICDR. ICDR can be accessed only when the ICE bit in ICCR is set to 1.
- The slave address register (SAR) sets the format, together with the FSX bit in SARX. It also stores the slave address during slave operation. SAR can be accessed only when the ICE bit in ICCR is cleared to 0.
- The second slave address register (SARX) sets the format, together with the FS bit in SAR. It also stores the second slave address during slave operation. SARX can be accessed only when the ICE bit in ICCR is cleared to 0.

1.2.2 DTC (Data Transfer Controller)

In this sample task, the DTC is used to transfer the transmit data from the on-chip RAM to the I²C module's data register. Figure 1.4 is a block diagram of the DTC module, which is explained below.

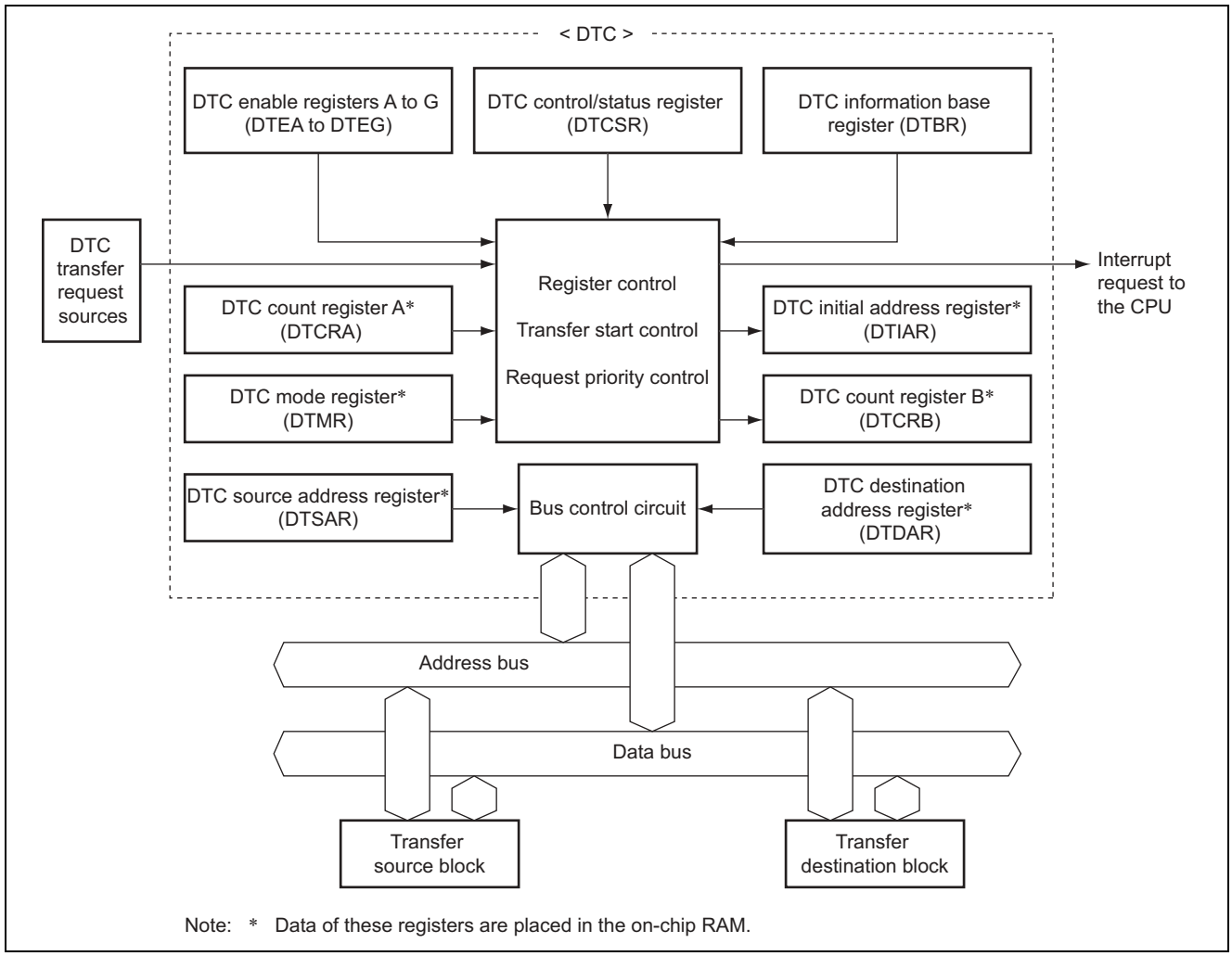


Figure 1.4 DTC Module Block Diagram

- In order to use the DTC, a DTC vector table is necessary. For details on vector addresses and related information, refer to the hardware manual.
- Registers denoted with "*" in figure 1.4 cannot be accessed directly from the CPU. This register information is placed in the on-chip RAM, with the upper 16 bits of the start address set in DTBR, and the lower 16 bits set in the DTC vector table. During DTC operation, register information is automatically read and transferred from the on-chip RAM.
- The DTC mode register (DTMR) sets the transfer data size and the DTC operating mode.
- The DTC source address register (DTSAR) specifies the transfer source address for DTC transfer.
- The DTC destination address register (DTDAR) specifies the transfer destination address for DTC transfer.
- The DTC initial address register (DTIAR) specifies the initial address of the transfer source or transfer destination in repeat mode.
- The DTC transfer count register A (DTCRA) specifies the number of DTC transfers. The number of transfers is 65,536 when the setting is H'0000.
- The DTC transfer count register B (DTCRB) specifies the block length in block transfer mode. The block length is 65,536 when the setting is H'0000.
- The DTC enable register (DTER) is a register used to select interrupt sources that activate the DTC. For details, refer to the hardware manual.
- The DTC control/status register (DTCSR) sets values to enable or disable DTC activation by software, and sets the DTC vector address for activation by software.
- The DTC information base register (DTBR) specifies the upper 16 bits of the memory address in which DTC transfer information is stored. By means of the DTBR and the DTC vector table, the storage address for DTC transfer information is specified. DTBR should always be accessed in word or long word units.

1.3 Description of Operation

Data contents when writing to the EEPROM in this sample task appear in figure 1.5.

After issue of a start condition, the slave address and R/W bit cleared to 0 (specifying writing) are transmitted. Next, the upper byte and lower byte of the start address of EEPROM to which data is to be written are transmitted. Then, the data which will be written are transmitted in sequence. When ACK from the EEPROM is 0, the next data is transmitted. Finally, a stop condition is issued.

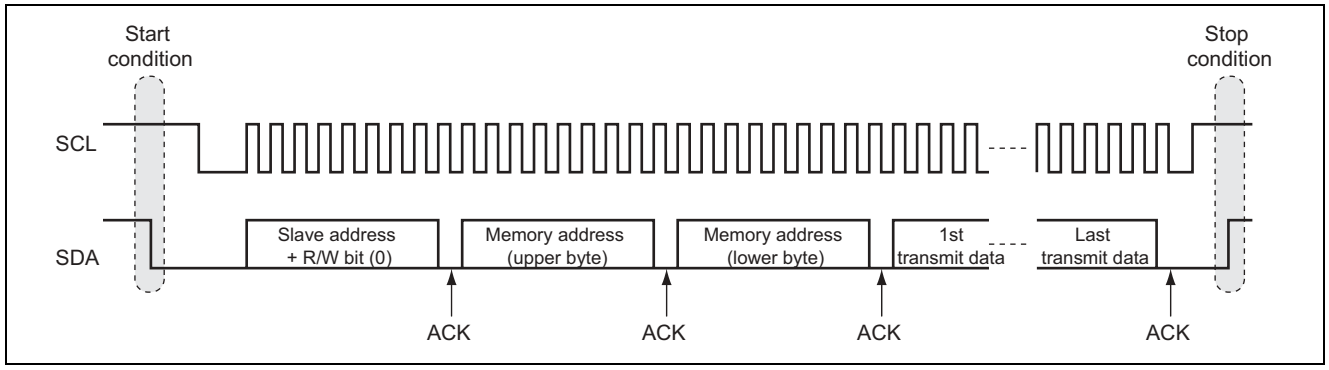


Figure 1.5 Data Contents When Writing to EEPROM

Figure 1.6 shows the details of operation when data writing to the EEPROM is started, and figure 1.7 describes the operation when writing ends. As explanations of figure 1.6 and figure 1.7, the contents of software and hardware processing are shown in table 1.3 and table 1.4 respectively.

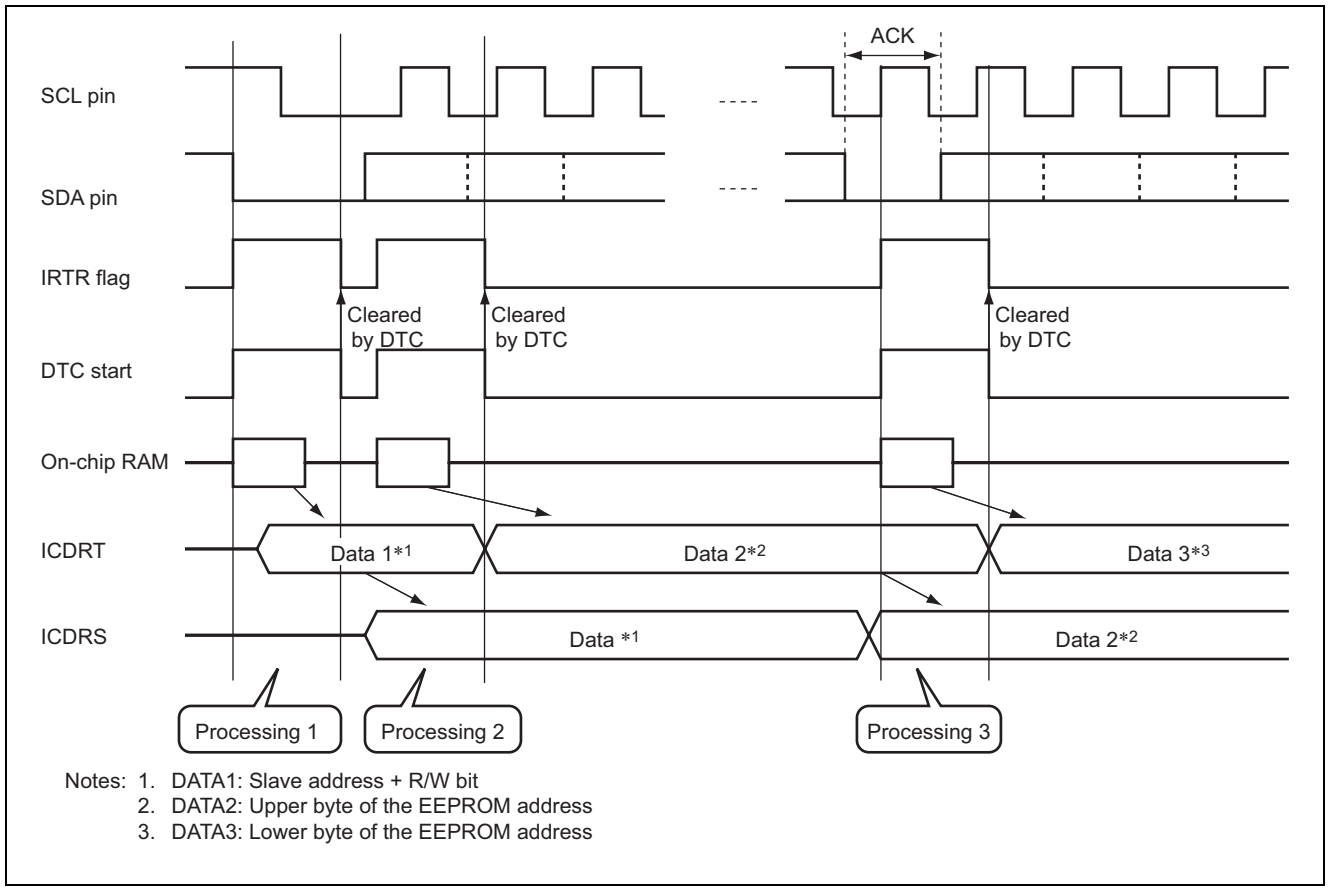
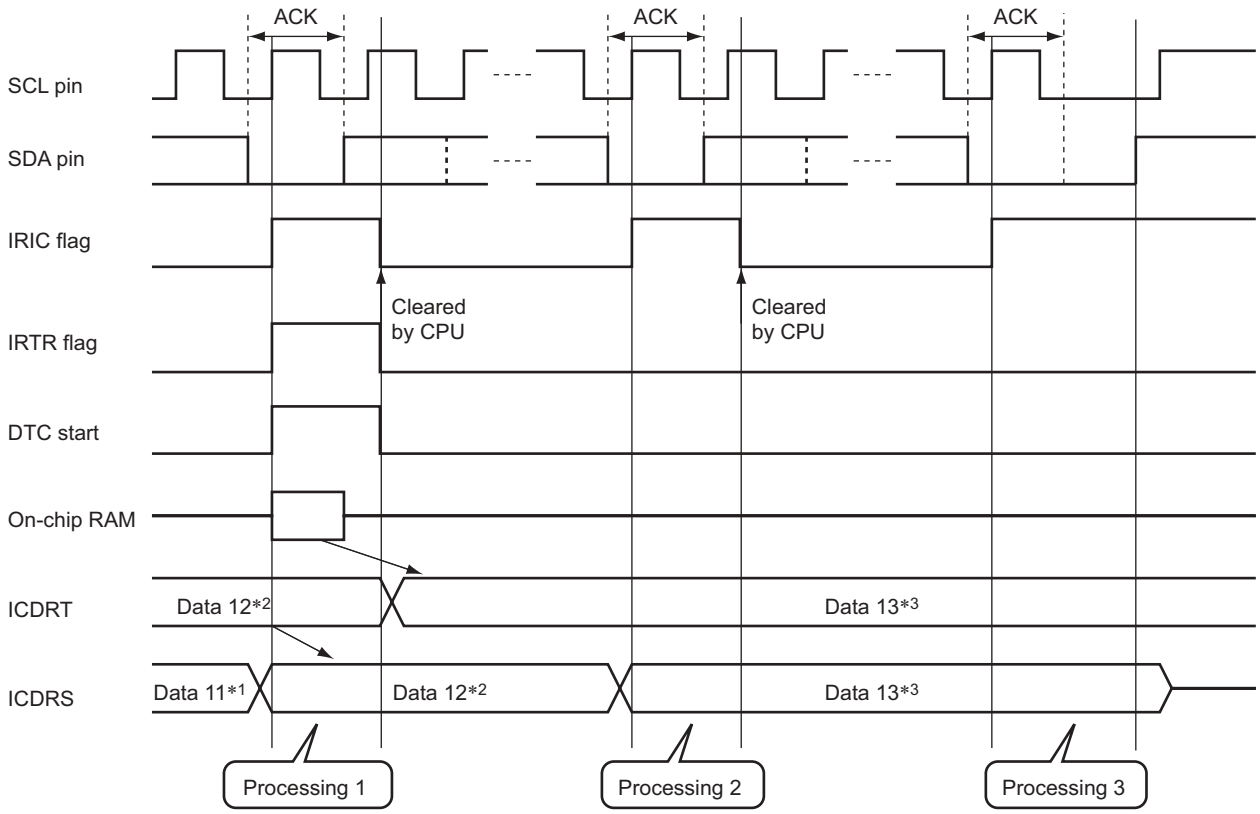


Figure 1.6 Operation When Data Write to EEPROM Starts

Table 1.3 Software and Hardware Processing When Data Write to EEPROM Starts

	Software Processing	Hardware Processing
Processing 1	• Set BBSY to 1 and clear SCP to 0 to issue a start condition.	• Set the IRTR flag to 1 when a start condition is detected. • Start DTC operation by the IRTR flag. • Transfer the transmit data (DATA1) in the on-chip RAM to ICDRT (transmit buffer in ICDR) by the DTC • Clear the IRTR flag to 0 by the DTC.
Processing 2	—	• Transfer data (DATA1) in ICDRT to ICDRS (shift register in ICDR). • Transmit data (DATA1) in ICDRS and set the IRTR flag to 1. • Start DTC operation by the IRTR flag. • Transfer transmit data (DATA2) in the on-chip RAM to ICDRT by the DTC. • Clear the IRTR flag to 0 by the DTC.
Processing 3	—	• After end of DATA1 transmission, transfer data (DATA2) in ICDRT to ICDRS. • Transmit data (DATA2) in ICDRS and set the IRTR flag to 1. • Start DTC operation by the IRTR flag. • Transfer transmit data (DATA3) in the on-chip RAM to ICDRT by the DTC. • Clear the IRTR flag by the DTC.



Notes: 1. DATA11: 8th byte of data to be written to the EEPROM
2. DATA12: 9th byte of data to be written to the EEPROM
3. DATA13: 10th byte of data to be written to the EEPROM

Figure 1.7 Operation When Data Write to EEPROM Ends

Table 1.4 Software and Hardware Processing When Data Write to EEPROM Ends

	Software Processing	Hardware Processing
Processing 1	• Disable the ICI interrupt. • Clear the IRIC flag. • Wait until the IRIC flag is set to 1.	• After DATA11 has been transmitted, set the IRTR and IRIC flags to 1. • Start DTC operation by the IRTR flag. • Transfer transmit data (DATA13) in on-chip RAM to ICDRT by the DTC • When data transfer by the DTC ends, request an interrupt to the CPU.
Processing 2	• Clear the IRIC flag. • Wait until the IRIC flag is set to 1.	• After DATA12 has been transmitted, set the IRIC flag to 1. • Transfer data (DATA13) in ICDRT to ICDRS and transmit it.
Processing 3	• Clear the ACKE bit in ICCR to 0. • Clear BBSY and SCP to 0 to issue a stop condition.	• After DATA13 has been transmitted, set the IRIC flag to 1.

1.4 Description of Software

1.4.1 Modules

Table 1.5 shows the modules used in this sample task.

Table 1.5 Description of Modules

Module Name	Label Name	Function
Main routine	main	Cancels the I²C module standby state and sets pin function and write data.
I²C master transmission routine	iic_mast_trans	Writes data to the EEPROM.
I²C interrupt routine	int_iic	Performs the last frame processing and issues a stop condition.

1.4.2 Internal Registers

Tables 1.6 through 1.8 describe internal registers used in this sample task. The settings are the values used in this sample task, and differ from their initial values.

Table 1.6 Description of Internal Registers (1)

Register Name	Bit	Bit Name	Setting	Function
MSTCR1				Module standby control register 1
	9	MSTP25	0	DTC Standby Control
	8	MSTP24	0	When MSTP25 = 0 and MSTP24 = 0, DTC standby state is cancelled.
	5	MSTP21	0	I²C Standby Control When MSTP21 = 0, I²C standby state is cancelled.
SCRX				Serial control register X
	7	—	0	Reserved
	6			
	5	IICX	1	I²C Transfer Rate Select Selects transfer rate in combination with CKS2 to CKS0 in ICMR
	4	IICE	1	I²C Master Enable When IICE = 1, the CPU is enabled to access ICDR and ICMR.
	3	HNDS	1	Handshake Reception When HNDS = 1, continuous receive operation is disabled.
	2	—	0	Reserved
	1	ICDRF	0	ICDRF = 0 indicates that there is no valid receive data in ICDR.
	0	STOPIM	1	Stop Condition Detection Interrupt Mask When STOPIM = 1, even when a stop condition is detected in slave mode, issue of an IRIC interrupt is disabled.

Table 1.7 Description of Internal Registers (2)

Register Name	Bit	Bit Name	Setting	Function
ICMR				I²C bus mode register
	7	MLS	0	MSB-First or LSB-First Selection When MSL = 0, MSB first is selected.
	6	WAIT	0	Wait Insertion When WAIT = 0, data and ACK are transferred continuously.
	5	CKS2	1	Transfer Clock Select 2 to 0 These bits select the frequency of the transfer clock in combination with IICX in SCRX. (In this sample task, the frequency is specified as 156 kHz.)
	4	CKS1	1	
	3	CKS0	1	
	2	BC2	0	Bit Counter These bits specify the number of bits of the data that is to be transferred next. (In this sample task, 9 bits/frame.)
	1	BC1	0	
	0	BC0	0	
ICCR			—	I²C bus control register
	7	ICE	1	I²C Bus Interface Enable When ICE = 1, the I²C module is placed in a transfer enabled state.
	6	IEIC	1	I²C Bus Interface Interrupt Enable When IEIC = 1, interrupts to the CPU are enabled.
	5	MST	1	Master/Slave Select When MST = 1, the I²C bus is used in master mode.
	4	TRS	1	Transmission/Reception Select When TRS = 1, the transmission mode is selected.
	3	ACKE	1	Acknowledge Bit Check Select When ACKE = 1, if ACK=1 is detected, continuous transfer is discontinued.
	2	BBSY	*1	Bus Busy Flag BBSY = 0 indicates a bus released state. BBSY = 1 indicates a bus occupied state.
	1	IRIC	*2	I²C Bus Interface Interrupt Request Flag IRIC = 0 indicates a transfer wait state or transfer in progress. IRIC = 1 indicates that the I²C bus interface has generated an interrupt.
	0	SCP	*1	Start/Stop Condition Issue Disable When SCP = 0, combined with BBSY, a start condition or a stop condition is issued. Invalid when SCP = 1.
PBCR1			H'0C00	Port B control register Sets port B pin functions in combination with PBCR2. In this sample task, the functions are set to SCL0 (clock I/O pin) and SDA0 (data I/O pin).

Notes: 1. When BBSY=1 and SCP=0, a start condition is issued; when BBSY = 0 and SCP = 0, a stop condition is issued.
2. Set to 1 by hardware.

Table 1.8 Description of Internal Registers (3)

Register Name	Bit	Bit Name	Setting	Function
PBCR2			H'0000	Port B control register Sets port B pin functions in combination with PBCR1. In this sample task, the functions are set to SCL0 (clock I/O pin) and SDA0 (data I/O pin).
DTCRA			13	DTC transfer count register A Specifies the number of DTC data transfers.
DTSAR			&WR_DATA[0]	DTC source address register Specifies a transfer source address for DTC transfer. The address of on-chip RAM that stores the data to be transmitted by I²C is set.
DTDAR			H'FFFF880E	DTC destination address register Specifies a transfer destination address for DTC transfer. The address of the I²C module's ICDR register of is set.
DTBR			H'FFFF	DTC information base register Specifies the upper 16 bits of the address for storing DTC transfer information.
DTEG			H'80	DTC enable register G Specifies the ICI interrupt as the interrupt source for DTC activation.
DTMR			—	DTC mode register
	15	SM1	1	Source Address Mode
	14	SM0	0	When SM[1,0] = B'10, DTSAR is incremented.
	13	DM1	0	Destination Address Mode
	12	DM0	0	When DM[1,0] = B'0X, DTDAR is fixed.
	11	MD1	0	DTC Mode
	10	MD0	0	When MD[1,0] = B'00, normal mode transfer is selected.
	9	Sz1	0	DTC Data Transfer Size
	8	Sz0	0	When Sz[1,0] = B'00, byte-size transfer is selected.
	7	DTS	0	DTC Transfer Mode Select No effect because normal mode transfer is selected in this sample task.
	6	CHNE	0	DTC Chain Transfer Enable When CHNE = 0, chain transfer is disabled.
	5	DISEL	0	DTC Interrupt Select When DISEL = 0, an interrupt request to the CPU is generated after transfer of all specified data has ended.
	4	NMIM	0	DTC NMI Mode When NMIM = 0, DTC transfer is interrupted by NMI.
	3 to 0	—	0	Reserved

1.4.3 RAM Usage

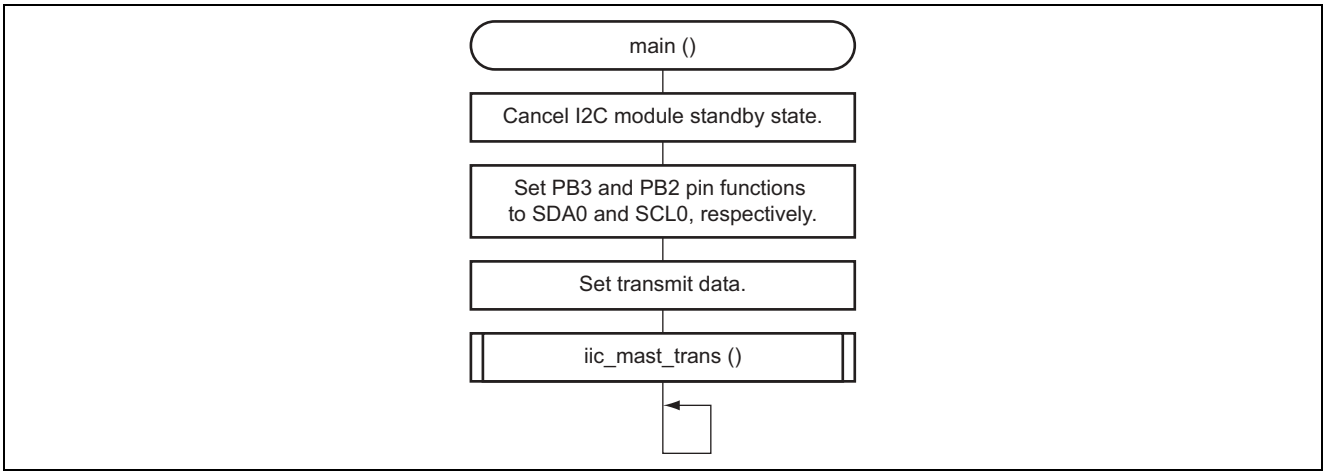
Table 1.9 describes the RAM usage in this sample task.

Table 1.9 Description of RAM

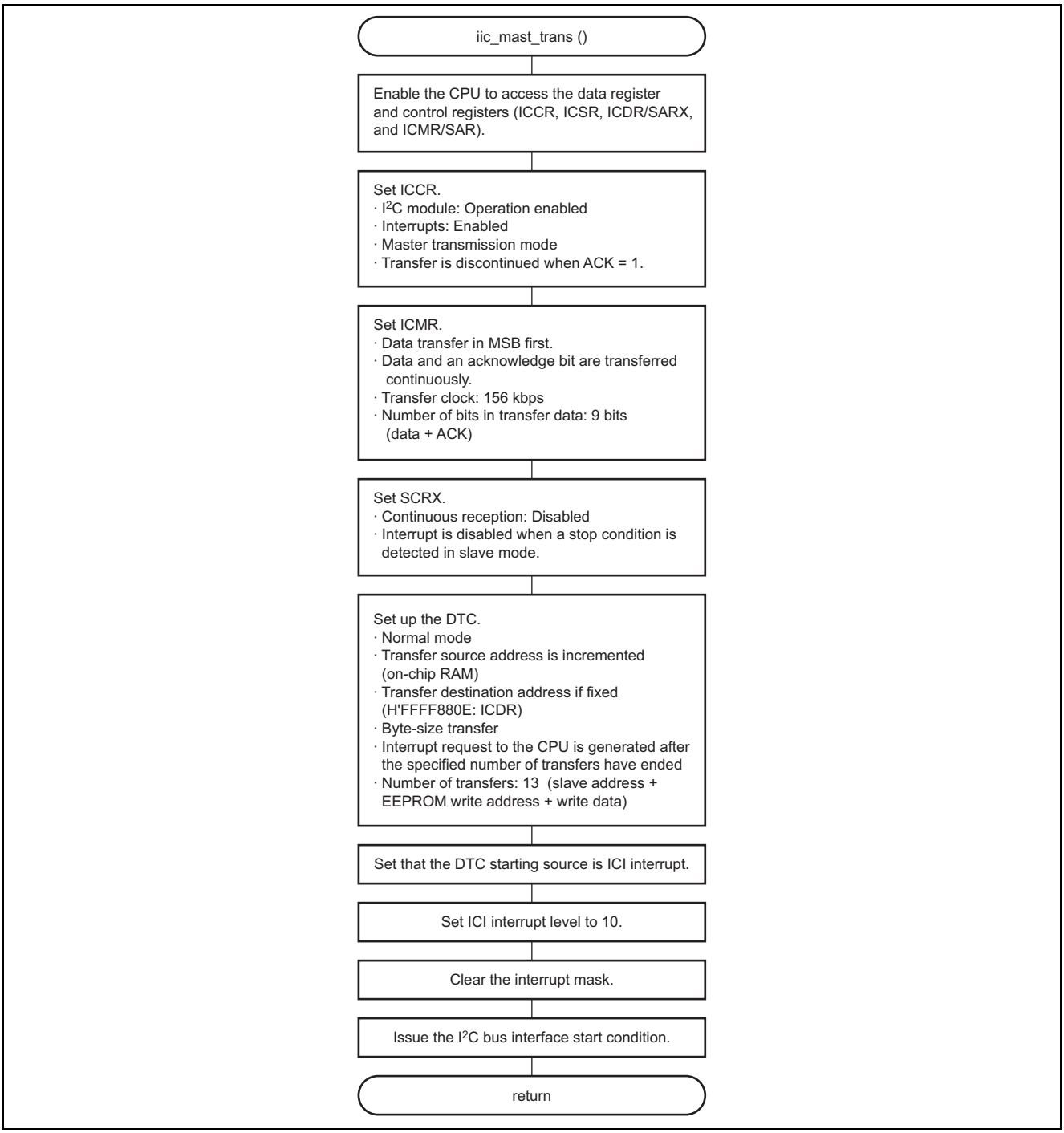
Label Name	Function	Address	Used in
WR_DATA[0-12]	Stores data to be written to the EEPROM (DTC transfer source address during transmission)	On-chip RAM	Main routine, I²C master transmission routine

1.5 Flowchart

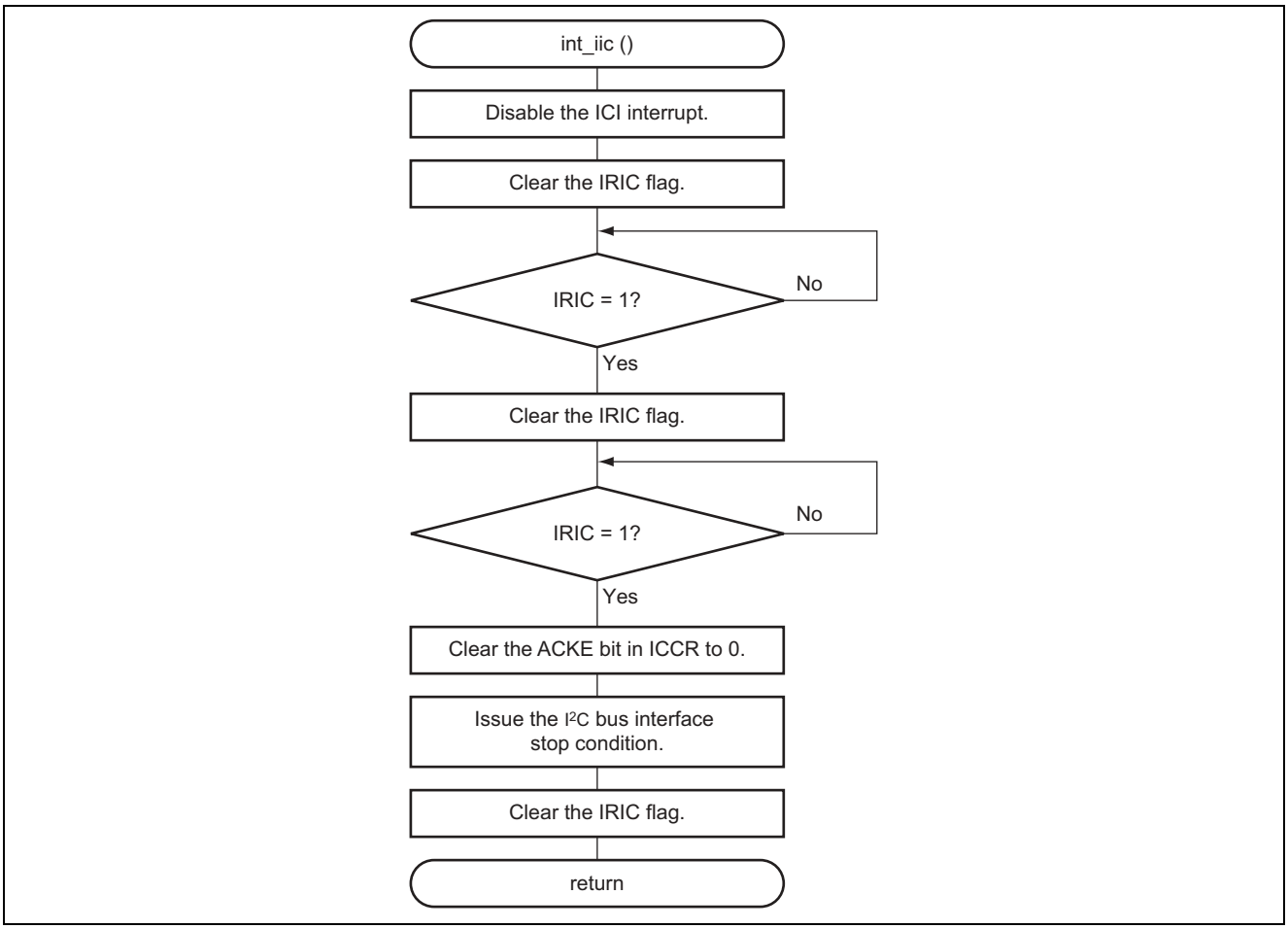
1.5.1 Main Routine



1.5.2 I²C Master Transmission Routine



1.5.3 I²C Interrupt Routine



1.6 Program Listing

```
/************************************************************************************************/
/* SH7145F    Application Note                                                                  */
/*                                                                                              */
/* Function                                                                                     */
/*    :I2C,DTC(EEPROM;HN58X2464,64k bit,8word x 8bit)                                           */
/*    :Single Master Transmit                                                                   */
/*                                                                                              */
/* External input clock         :10MHz                                                          */
/* Internal CPU clock           :40MHz                                                          */
/* Internal peripheral clock    :40MHz                                                          */
/*                                                                                              */
/* Written        :2004/1       Rev.1.0                                                         */
/************************************************************************************************/

#include "iodefine.h"
#include <machine.h>

/************************************************************************************************/
/*  Symbol Definition                                                                           */
/************************************************************************************************/
struct st_dtc_iic {
    unsigned short DTMR;
    unsigned short DTCRA;
    unsigned short dummy1;
    unsigned short dummy2;
    unsigned long DTSAR;
    unsigned long DTDAR;
};

#define D_CODE    0xA0                          /* Device code of EEPROM                  */
#define A_CODE    0x00                          /* Device address code of EEPROM          */
#define WR        0x00                          /* Write data B'0                         */

#define UP_ADDR   0x00                          /* Upper address of EEPROM                */
#define LO_ADDR   0x00                          /* Lower address of EEPROM                */

#define WR_NUM    13                            /* The number of write data to EEPROM     */

/************************************************************************************************/
/*  Function define                                                                             */
/************************************************************************************************/
void main(void);
void iic_mast_trans(void);

void int_iic(void);
void dummy_f(void);
```

```
/*****************************************************************************************************/
/*  RAM allocation Definition                                                                        */
/*****************************************************************************************************/
unsigned char WR_DATA[WR_NUM];                      /* Write data                                    */

#define DTC_ICI (*(volatile struct st_dtc_iic*)0xFFFFF000)
                                                    /* DTC information address                       */

/*****************************************************************************************************/
/*  Main Program                                                                                     */
/*****************************************************************************************************/
void main( void )
{
    P_STBY.MSTCR1.BIT.MSTP21 = 0;                   /* Disable IIC standby mode                      */

     /* Set of I2C Pin Function                                                                      */
    P_PORTB.PBCR1.WORD = 0x0C00;
    P_PORTB.PBCR2.WORD = 0x0000;                    /* SDA0(PB3-32pin@SH7145F),                      */
                                                    /* SCL0(PB2-31pin@SH7145F)                       */

    /* Set of Transmitting Data                                                                     */
    WR_DATA[0]  = (D_CODE|A_CODE|WR);               /* Device code + R/W bit(0;Write)                */
    WR_DATA[1]  = UP_ADDR;                          /* Set of upper address to EEPROM                */
    WR_DATA[2]  = LO_ADDR;                          /* Set of lower address to EEPROM                */
    WR_DATA[3]  = 0x11;                             /* Write data(H'0000:EEPROM address)             */
    WR_DATA[4]  = 0x22;                             /* Write data(H'0001:EEPROM address)             */
    WR_DATA[5]  = 0x33;                             /* Write data(H'0002:EEPROM address)             */
    WR_DATA[6]  = 0x44;                             /* Write data(H'0003:EEPROM address)             */
    WR_DATA[7]  = 0x55;                             /* Write data(H'0004:EEPROM address)             */
    WR_DATA[8]  = 0x66;                             /* Write data(H'0005:EEPROM address)             */
    WR_DATA[9]  = 0x77;                             /* Write data(H'0006:EEPROM address)             */
    WR_DATA[10] = 0x88;                             /* Write data(H'0007:EEPROM address)             */
    WR_DATA[11] = 0x99;                             /* Write data(H'0008:EEPROM address)             */
    WR_DATA[12] = 0xAA;                             /* Write data(H'0009:EEPROM address)             */

    iic_mast_trans();                               /* Write routine                                 */

    while(1);                                       /* LOOP                                          */
}
```

```
/*****************************************************************************************************/
/*  Function    : iic_mast_trans                                                                     */
/*  Operation   : Data write to EEPROM by the I2C interface                                          */
/*  Argument    : Non-argument                                                                       */
/*  Return      : Non-return                                                                         */
/*****************************************************************************************************/
void iic_mast_trans(void)
{
    P_IIC.SCRX.BIT.IICE = 1;                        /* Register access enable from CPU          */

    P_IIC.ICCR0.BYTE = 0xF9;
            // ICE      [7]=B'1      : I2C interface enable
            // IECE     [6]=B'1      : I2C interrupt enable
            // MST      [5]=B'1      : Master mode
            // TRS      [4]=B'1      : Transmit mode
            // ACKE     [3]=B'1      : Continuous data transfer is halted
            // BBSY     [2]=B'0
            // IRIC     [1]=B'0
            // SCP      [0]=B'1
    P_IIC.ICMR0.BYTE = 0x38;
            // MLS      [7]=B'0      : MSB first
            // WAIT     [6]=B'0      : Data and an acknowledge are transmitted continuously
            // CKS      [5-3]=B'111  : Transfer clock(with IICX0) = 156kHz(P phi=40MHz)
            // BC       [2-0]=B'000
    P_IIC.SCRX.BYTE = 0x39;
            // Reserve [7-6]=B'00
            // IICX     [5]=B'1      : Transfer rate select, reference CKS bit
            // IICE     [4]=B'1      : Register access enable from CPU
            // HNDS     [3]=B'1
            // Reserve [2]=B'0
            // ICDRF    [1]=B'0
            // STOPIM   [0]=B'1      : Disables interrupt requests

    /* DTC Initialize                                                                               */
    DTC_ICI.DTMR = 0x8000;                          /* Set DTC mode                             */
            // SM       [15-14]=B'10 : Source address increment
            // DM       [13-12]=B'00 : Destination address unchanging
            // MD       [11-10]=B'00 : Normal mode
            // Sz       [9-8]  =B'00 : Transfer data size = byte
            // DTS      [7]=B'0      : Not used
            // CHNE     [6]=B'0      : Not used
            // DISEL    [5]=B'0      : DTC interrupt disable
            // NMIM     [4]=B'0      : NMI->Terminate DTC transfer
            // Reserve [3-0]=B'0000
    DTC_ICI.DTCRA = WR_NUM;                         /* Transfer count                           */
    DTC_ICI.DTSAR = (unsigned long)&(WR_DATA[0]);
                                                    /* Set source address                       */
    DTC_ICI.DTDAR = (unsigned long)&(P_IIC.ICDR0.BYTE);
                                                    /* Set destination address                  */

    P_DTC.DTBR = 0xFFFF;                            /* DTC information base register            */
    P_DTC.DTEG.BYTE |= 0x80;                        /* Interrupt sources IIC                    */
```

```
    P_INTC.IPRJ.BIT.IIC = 10;                       /* Set of IIC interrupt level              */
    set_imask(0);                                   /* Clear interrupt mask level              */

    P_IIC.ICCR0.BYTE = ((P_IIC.ICCR0.BYTE & 0xFE) | 0x04);
                                                    /* Generate the start condition            */
}

/*******************************************************************************************/
/*  Interruption Program                                                                   */
/*******************************************************************************************/
/*******************************************************************************************/
/*  Function    : int_iic                                                                  */
/*  Operation   : Transmission end interruption                                            */
/*  Argument    : Non-argument                                                             */
/*  Return      : Non-return                                                               */
/*******************************************************************************************/
#pragma interrupt(int_iic)
void int_iic(void)
{
    P_IIC.ICCR0.BIT.IEIC = 0;                       /* IEIC interrupt disable                  */

    P_IIC.ICCR0.BIT.IRIC = 0;                       /* Clear IRIC flag                         */
    while(P_IIC.ICCR0.BIT.IRIC != 1);               /* Wait 1 byte transmitted                 */

    P_IIC.ICCR0.BIT.IRIC = 0;                       /* Clear IRIC flag                         */
    while(P_IIC.ICCR0.BIT.IRIC != 1);               /* Wait 1 byte transmitted                 */
    P_IIC.ICCR0.BIT.ACKE = 0;                       /* Clear ACKE bit                          */
    P_IIC.ICCR0.BYTE = P_IIC.ICCR0.BYTE & 0xFA;
                                                    /* Generate stop condition                 */
}

/*******************************************************************************************/
/*  Other Interruption Program                                                             */
/*******************************************************************************************/
#pragma interrupt(dummy_f)
void dummy_f(void)
{
    /* Other Interrupt */
}
```

2. I²C Bus Single-Master Reception in Combined Use with DTC

2.1 Specifications

Data reception is performed via an I²C bus (Inter IC Bus) in combined use with the Data Transfer Controller (DTC) of the SH7145F.

As shown in figure 2.1, the master device is the SH7145F, and an EEPROM (HN58X2464, 64 kbits, 8 words × 8 bits) is connected as the slave device. The I²C bus interface of the SH7145F is used to read 10 bytes of data from the EEPROM, which is then stored to the on-chip RAM. In this process, the DTC is used to transfer the read data to the on-chip RAM.

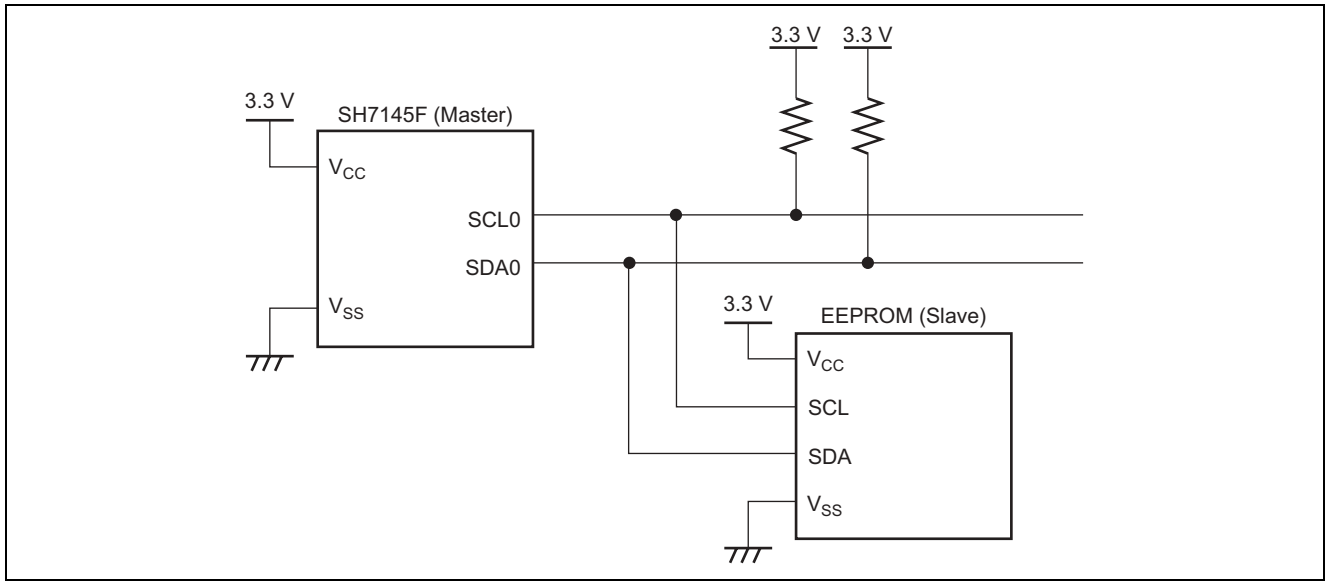


Figure 2.1 Connection of EEPROM to SH7145F

Figure 2.2 shows the data transfer by the DTC. Table 2.1 shows the DTC settings, and table 2.2 shows the I²C bus interface settings.

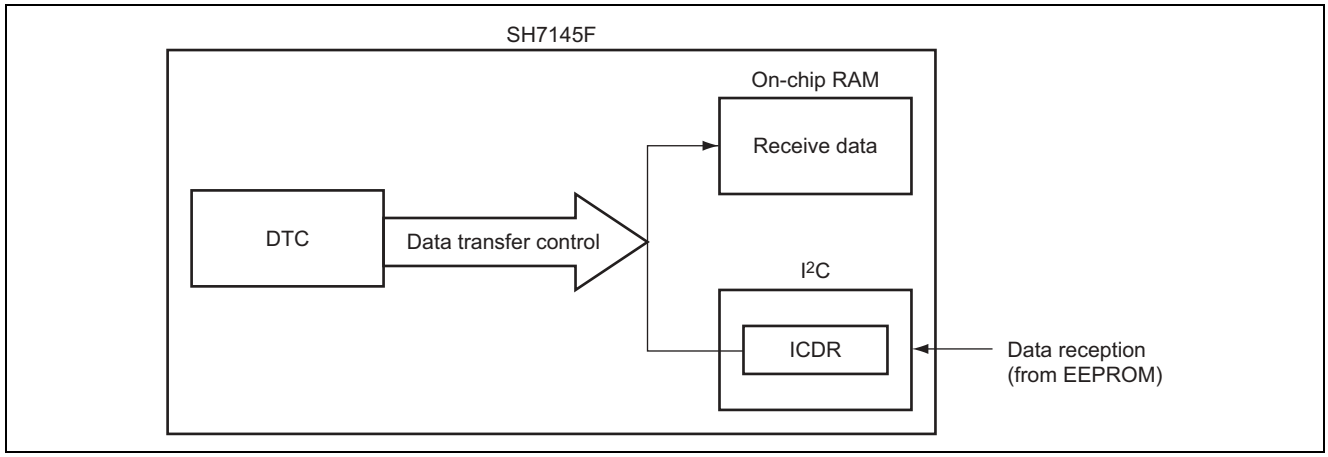


Figure 2.2 Data Transfer by DTC

Table 2.1 DTC Settings

Condition	Setting
Transfer mode	Normal mode
Transfer data size	1 byte
Number of transfers (DTCRA)	13
Transfer source	ICDR (I²C bus data register: H'FFFF880E)
Transfer destination	On-chip RAM
Transfer source address	Fixed address
Transfer destination address	Incremented after transfer.
Transfer information storage address	H'FFFFF000
Transfer start source	Started by an I²C interrupt (ICI).
Interrupt	Disabled

Table 2.2 I²C Settings

Item	Setting
Operation	Master reception
Transfer clock	156 kHz (Pϕ = 40 MHz)
Number of bits in data	9 bits (including ACK)
Wait between data and ACK	None
Interrupt	Enabled
ACK transmission	"1" is output only when the last data is received. During continuous reception, "0" is output.

2.2 Description of Functions

In this sample task, the I²C bus (Inter IC Bus) is used to read data from an EEPROM. The read data is transferred from the data register of the I²C bus interface to the on-chip RAM using the DTC (Data Transfer Controller).

2.2.1 I²C Bus (Inter IC Bus) Interface

This interface conforms to the I²C bus interface format set up by Philips Corp., and is provided with subset functions. Figure 2.3 is a block diagram of the I²C module, which is explained below.

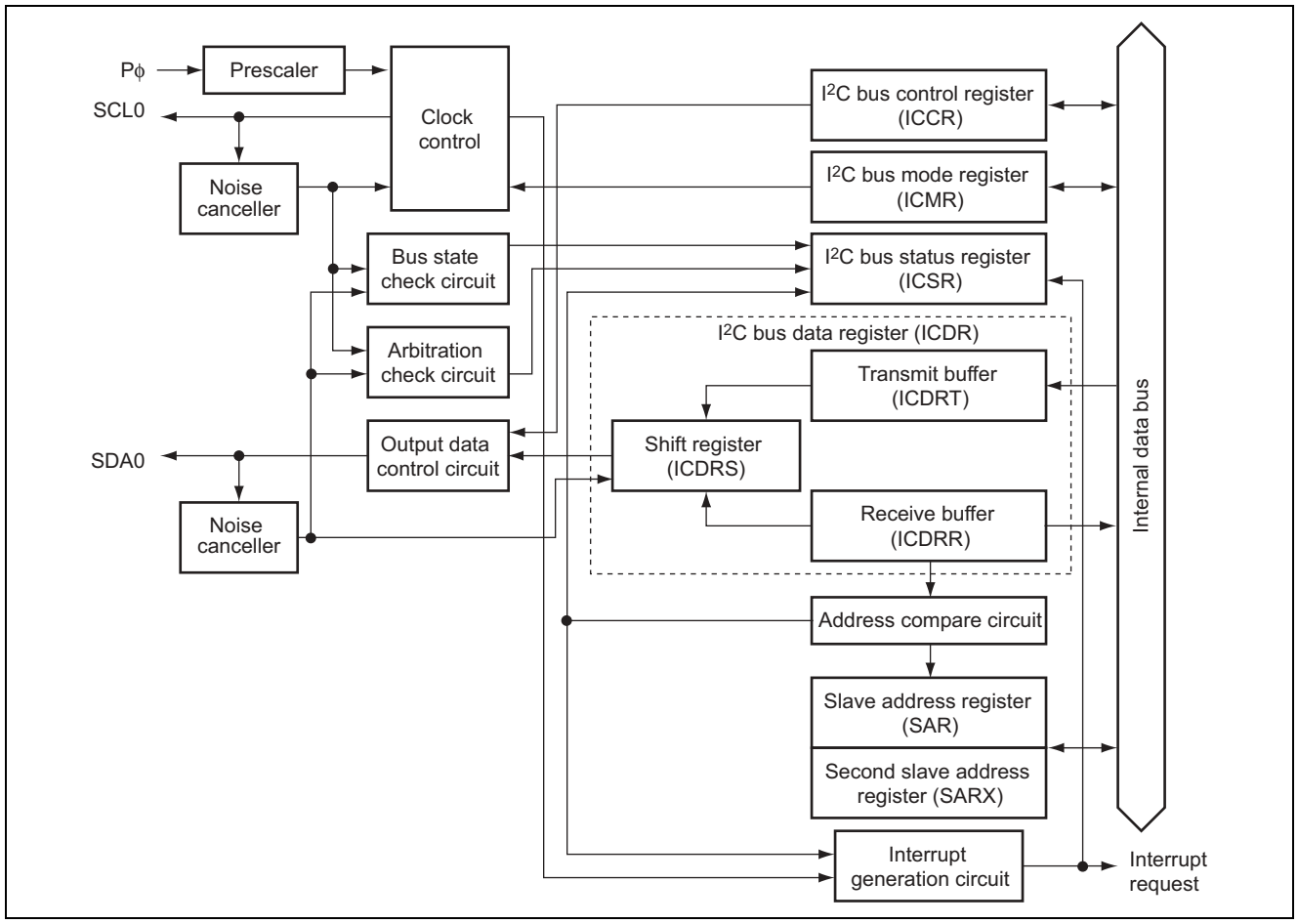


Figure 2.3 I²C Module Block Diagram

- The I²C bus control register (ICCR) sets operation parameters for the I²C bus interface.
- The I²C bus mode register (ICMR) selects MSB- or LSB-first, and selects wait insertion, transfer clock, and the number of bits for transfer. ICMR can be accessed only when the ICE bit in ICCR is set to 1.
- The I²C bus status register (ICSR) is used to check flags and to check and control acknowledgement.
- The I²C bus data register (ICDR) is a data register used for both transmission and reception. ICDR is internally divided into a shift register (ICDRS), receive buffer (ICDRR), and transmit buffer (ICDRT). During transmission, when transmit data is written to ICDR, the data is stored to ICDRT, and it is automatically transferred to ICDRS after transmission of the preceding data. When ICDRS receives data, the received data is automatically transferred to ICDRR and can be read by reading ICDR. ICDR can be accessed only when the ICE bit in ICCR is set to 1.
- The slave address register (SAR) sets the format, together with the FSX bit in SARX. It also stores the slave address during slave operation. SAR can be accessed only when the ICE bit in ICCR is cleared to 0.
- The second slave address register (SARX) sets the format, together with the FS bit in SAR. It also stores the second slave address during slave operation. SARX can be accessed only when the ICE bit in ICCR is cleared to 0.

2.2.2 DTC (Data Transfer Controller)

In this sample task, the DTC is used to transfer the received data from the I²C module data register to the on-chip RAM. Figure 2.4 is a block diagram of the DTC module, explained below.

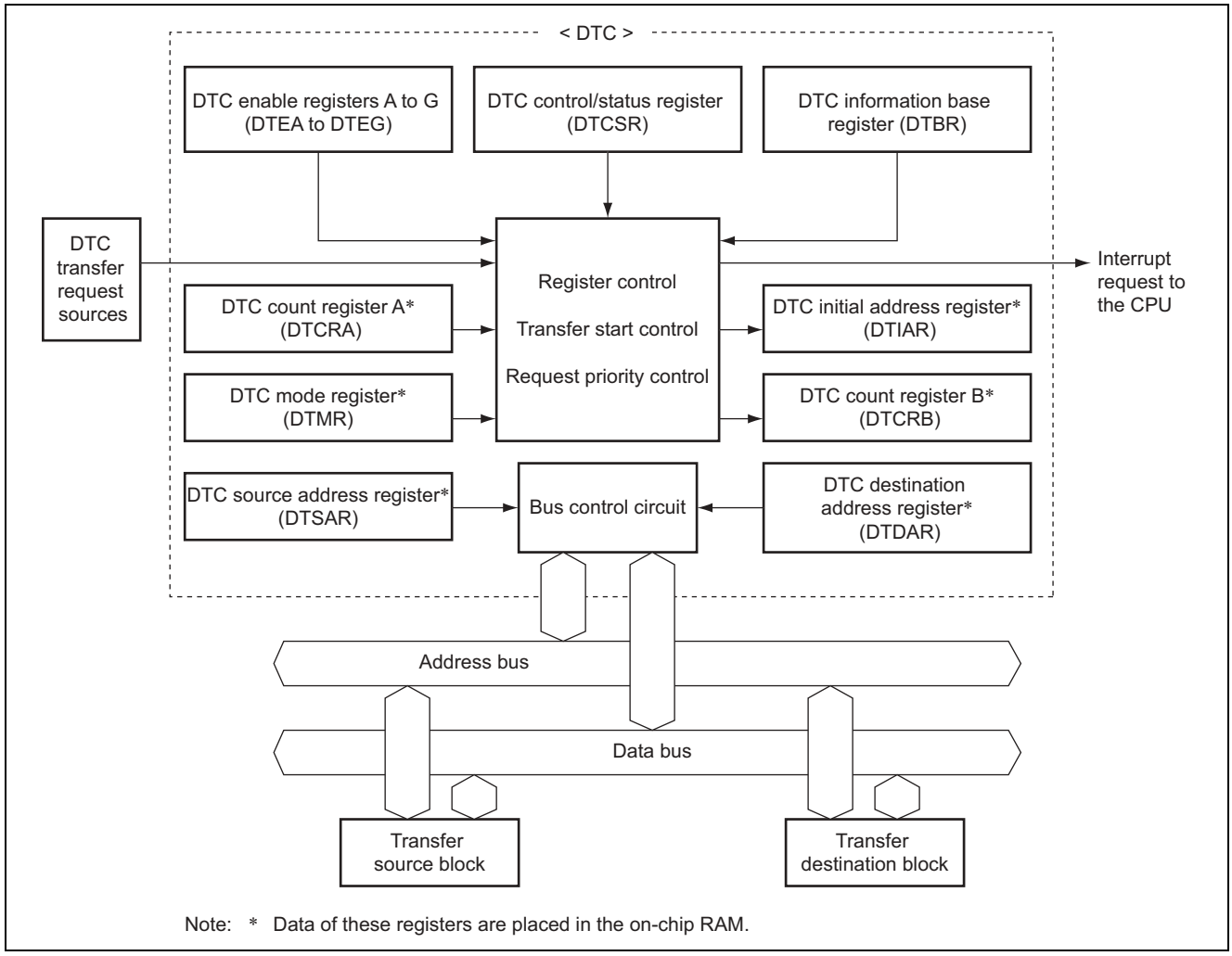


Figure 2.4 DTC Module Block Diagram

- In order to use the DTC, a DTC vector table is necessary. For details on vector addresses and related information, refer to the hardware manual.
- Registers denoted with "*" in figure 1.4 cannot be accessed directly from the CPU. This register information is placed in the on-chip RAM, with the upper 16 bits of the start address set in DTBR, and the lower 16 bits set in the DTC vector table. During DTC operation, register information is automatically read and transferred from the on-chip RAM.
- The DTC mode register (DTMR) sets the transfer data size and the DTC operating mode.
- The DTC source address register (DTSAR) specifies the transfer source address for DTC transfer.
- The DTC destination address register (DTDAR) specifies the transfer destination address for DTC transfer.
- The DTC initial address register (DTIAR) specifies the initial address of the transfer source or transfer destination in repeat mode.
- The DTC transfer count register A (DTCRA) specifies the number of DTC transfers. The number of transfers is 65,536 when the setting is H'0000.
- The DTC transfer count register B (DTCRB) specifies the block length in block transfer mode. The block length is 65,536 when the setting is H'0000.

- The DTC enable register (DTER) is a register used to select interrupt sources that activate the DTC. For details, refer to the hardware manual.
- The DTC control/status register (DTCSR) sets values to enable or disable DTC activation by software, and sets the DTC vector address for activation by software.
- The DTC information base register (DTBR) specifies the upper 16 bits of the memory address in which DTC transfer information is stored. By means of the DTBR and the DTC vector table, the storage address for DTC transfer information is specified. DTBR should always be accessed in word or long word units.

2.3 Description of Operation

Data contents when reading from the EEPROM in this sample task appear in figure 2.5.

After issue of a start condition, the slave address and R/W bit cleared to 0 (specifying writing) are transmitted. Next, the upper byte and lower byte of the start address of EEPROM from which data is to be read are transmitted. Then, a stop condition is issued.

After the second start condition is issued, the slave address and R/W bit set to 1 (specifying reading) are transmitted. Then, data is output in sequence from the EEPROM according to the I²C format. When the master receives the data, an acknowledge bit (ACK) is output. On receiving ACK = 0, the EEPROM outputs the next data. On receiving the last data, the master outputs ACK = 1 and issues a stop condition.

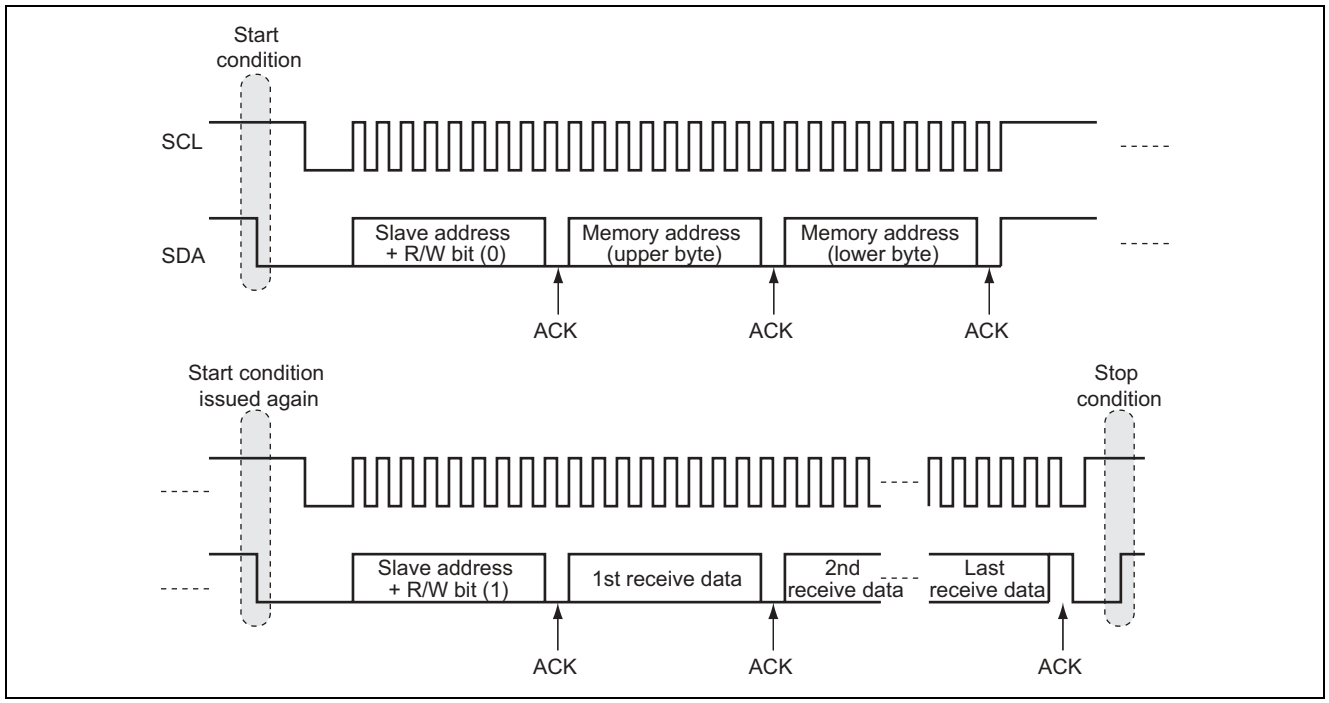
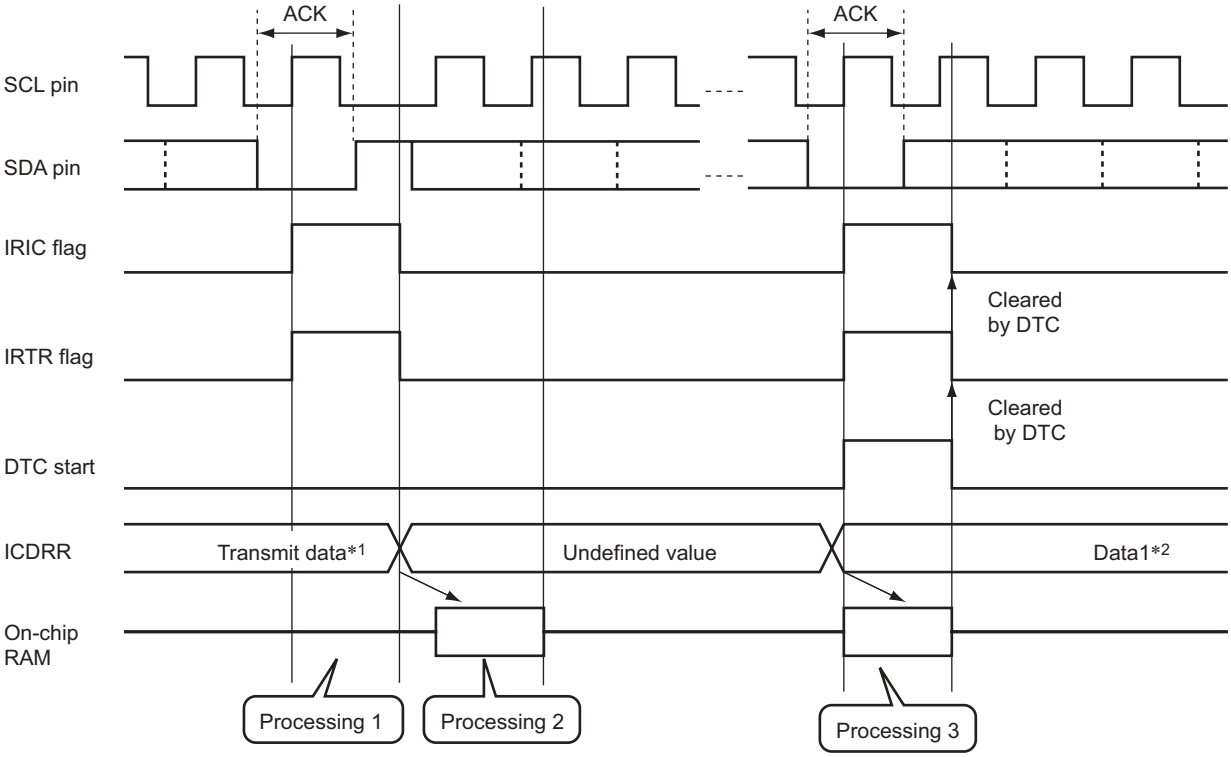


Figure 2.5 Data Contents When Reading from EEPROM

Figure 2.6 shows the details of operation when data reading from the EEPROM is started; figure 2.7 describes the operation when reading ends. As explanations of figure 2.6 and figure 2.7, the contents of software and hardware processing are shown in table 2.3 and table 2.4, respectively.

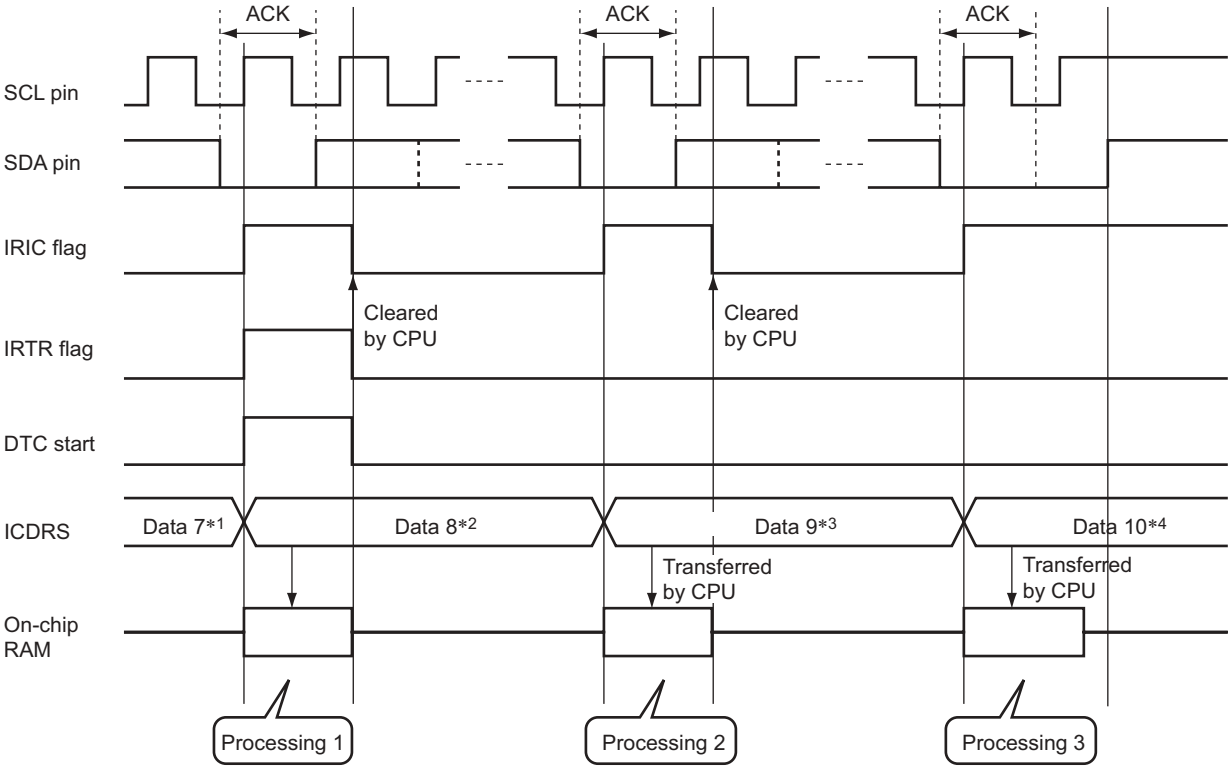


Notes: 1. Transmit data: Slave address + R/W bit
2. Data1: The first byte of data read from the EEPROM

Figure 2.6 Operation When Data Read from EEPROM Starts

Table 2.3 Software and Hardware Processing When Data Read from EEPROM Starts

	Software Processing	Hardware Processing
Processing 1	• Clear the IRIC flag.	• After transmission of the slave address + R/W bit, set the IRIC and IRTR flags to 1.
Processing 2	• Set the master reception mode and set ACK to 1. • Enable the ICI interrupt. • Set DTC transfer conditions. • Dummy-read the undefined value in ICDRR (receive buffer in ICDR)	• Start reception of data read from the EEPROM.
Processing 3	—	• After reception of data read from the EEPROM, set the IRIC and IRTR flags to 1. • Start DTC operation by the IRTR flag. • Transfer the received data in ICDRR to the on-chip RAM. • Clear the IRIC and IRTR flags.



Notes: 1. Data 7: 7th byte of data read from the EEPROM
2. Data 8: 8th byte of data read from the EEPROM
3. Data 9: 9th byte of data read from the EEPROM
4. Data 10: 10th byte of data read from the EEPROM

Figure 2.7 Operation When Data Read from EEPROM Ends

Table 2.4 Software and Hardware Processing When Data Read from EEPROM Ends

	Software Processing	Hardware Processing
Processing 1	—	• After reception of data read from the EEPROM, set the IRTR and IRIC flags to 1. • Start DTC operation by the IRTR flag. • Transfer the received data in ICDRR to the on-chip RAM • Clear the IRTR and IRIC flags to 0.
Processing 2	• Clear the IRIC flag. • Disable the ICI interrupt. • Set ACK to 1. • Read the 9th byte of the received data.	• After reception of data read from the EEPROM, set the IRTR and IRIC flags to 1.
Processing 3	• Set master transmission mode. • Clear the IRIC flag. • Read the 10th byte of the received data. • Clear BBSY and SCP to 0 to issue a stop condition.	• After reception of data read from the EEPROM, set the IRTR and IRIC flags to 1.

2.4 Description of Software

2.4.1 Modules

Table 2.5 shows the modules used in this sample task.

Table 2.5 Description of Modules

Module Name	Label Name	Function
Main routine	main	Cancels the I²C module standby state and sets pin functions.
I²C master transmission routine	iic_mast_trans	Reads data from the EEPROM.
I²C interrupt routine	int_iic	Performs the last frame processing and issues a stop condition.
Address transmission routine	addr_EEPROM	Sets the start address for reading from the EEPROM.

2.4.2 Internal Registers

Tables 2.6 through 2.8 describe internal registers used in this sample task. The settings are the values used in this sample task, and differ from their initial values.

Table 2.6 Description of Internal Registers (1)

Register Name	Bit	Bit Name	Setting	Function
MSTCR1				Module standby control register 1
	9	MSTP25	0	DTC Standby Control
	8	MSTP24	0	When MSTP25 = 0 and MSTP24 = 0, DTC standby state is cancelled.
	5	MSTP21	0	I²C Standby Control When MSTP21 = 0, I²C standby state is cancelled.
SCRX				Serial control register X
	7	—	0	Reserved
	6			
	5	IICX	1	I²C Transfer Rate Select Selects transfer rate in combination with CKS2 to CKS0 in ICMR
	4	IICE	1	I²C Master Enable When IICE = 1, the CPU is enabled to access ICDR and ICMR.
	3	HNDS	1	Handshake Reception When HNDS = 1, continuous receive operation is disabled.
	2	—	0	Reserved
	1	ICDRF	0	ICDRF = 0 indicates that there is no valid receive data in ICDR.
	0	STOPIM	1	Stop Condition Detection Interrupt Mask When STOPIM = 1, even when a stop condition is detected in slave mode, issue of an IRIC interrupt is disabled.

Table 2.7 Description of Internal Registers (2)

Register Name	Bit	Bit Name	Setting	Function
ICMR				I²C bus mode register
	7	MLS	0	MSB-First or LSB-First Selection When MSL = 0, MSB first is selected.
	6	WAIT	0	Wait Insertion When WAIT = 0, data and ACK are transferred continuously.
	5	CKS2	1	Transfer Clock Select 2 to 0 These bits select the frequency of the transfer clock in combination with IICX in SCRX. (In this sample task, the frequency is specified as 156 kHz.)
	4	CKS1	1	
	3	CKS0	1	
	2	BC2	0	Bit Counter These bits specify the number of bits of the data that is to be transferred next. (In this sample task, 9 bits/frame.)
	1	BC1	0	
	0	BC0	0	
ICCR			—	I²C bus control register
	7	ICE	1	I²C Bus Interface Enable When ICE = 1, the I²C module is placed in a transfer enabled state.
	6	IEIC	1	I²C Bus Interface Interrupt Enable When IEIC = 1, interrupts to the CPU are enabled.
	5	MST	1	Master/Slave Select When MST = 1, the I²C bus is used in master mode.
	4	TRS	0	Transmission/Reception Select When TRS = 0, the reception mode is selected.
	3	ACKE	1	Acknowledge Bit Check Select When ACKE = 1, if ACK=1 is detected, continuous transfer is discontinued.
	2	BBSY	*1	Bus Busy Flag BBSY = 0 indicates a bus released state. BBSY = 1 indicates a bus occupied state.
	1	IRIC	*2	I²C Bus Interface Interrupt Request Flag IRIC = 0 indicates a transfer wait state or transfer in progress. IRIC = 1 indicates that the I²C bus interface has generated an interrupt.
	0	SCP	*1	Start/Stop Condition Issue Disable When SCP = 0, combined with BBSY, a start condition or a stop condition is issued. Invalid when SCP = 1.
PBCR1			H'0C00	Port B control register Sets port B pin functions in combination with PBCR2. In this sample task, the functions are set to SCL0 (clock I/O pin) and SDA0 (data I/O pin).

Notes: 1. When BBSY=1 and SCP=0, a start condition is issued; when BBSY=0 and SCP=0, a stop condition is issued.
2. Set to 1 by hardware.

Table 2.8 Description of Internal Registers (3)

Register Name	Bit	Bit Name	Setting	Function
PBCR2			H'0000	Port B control register Sets port B pin functions in combination with PBCR1. In this sample task, the functions are set to SCL0 (clock I/O pin) and SDA0 (data I/O pin).
DTCRA			9	DTC transfer count register A Specifies the number of DTC data transfers.
DTSAR			H'FFFF880E	DTC source address register Specifies a transfer source address for DTC transfer. The address of the I²C module's ICDR register of is set.
DTDAR			&RD_DATA[0]	DTC destination address register Specifies a transfer destination address for DTC transfer. The address of on-chip RAM for storing the data read from the EEPROM is set.
DTBR			H'FFFF	DTC information base register Specifies the upper 16 bits of the address for storing DTC transfer information.
DTEG			H'80	DTC enable register G Specifies the ICI interrupt as the interrupt source for DTC activation.
DTMR			—	DTC mode register
	15	SM1	0	Source Address Mode
	14	SM0	0	When SM[1,0] = B'0X, DTSAR is fixed.
	13	DM1	1	Destination Address Mode
	12	DM0	0	When DM[1,0] = B'10, DTDAR is incremented.
	11	MD1	0	DTC Mode
	10	MD0	0	When MD[1,0] = B'00, normal mode transfer is selected.
	9	Sz1	0	DTC Data Transfer Size
	8	Sz0	0	When Sz[1,0] = B'00, byte-size transfer is selected.
	7	DTS	0	DTC Transfer Mode Select No effect because normal mode transfer is selected in this sample task.
	6	CHNE	0	DTC Chain Transfer Enable When CHNE = 0, chain transfer is disabled.
	5	DISEL	0	DTC Interrupt Select When DISEL = 0, an interrupt request to the CPU is generated after transfer of all specified data has ended.
	4	NMIM	0	DTC NMI Mode When NMIM = 0, DTC transfer is interrupted by NMI.
	3 to 0	—	0	Reserved

2.4.3 RAM Usage

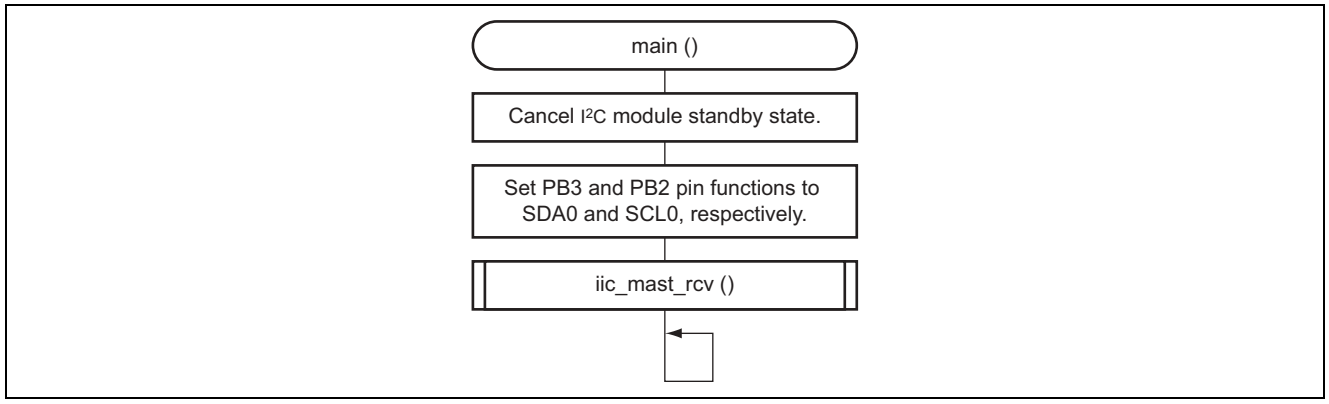
Table 2.9 describes the RAM usage in this sample task.

Table 2.9 Description of RAM

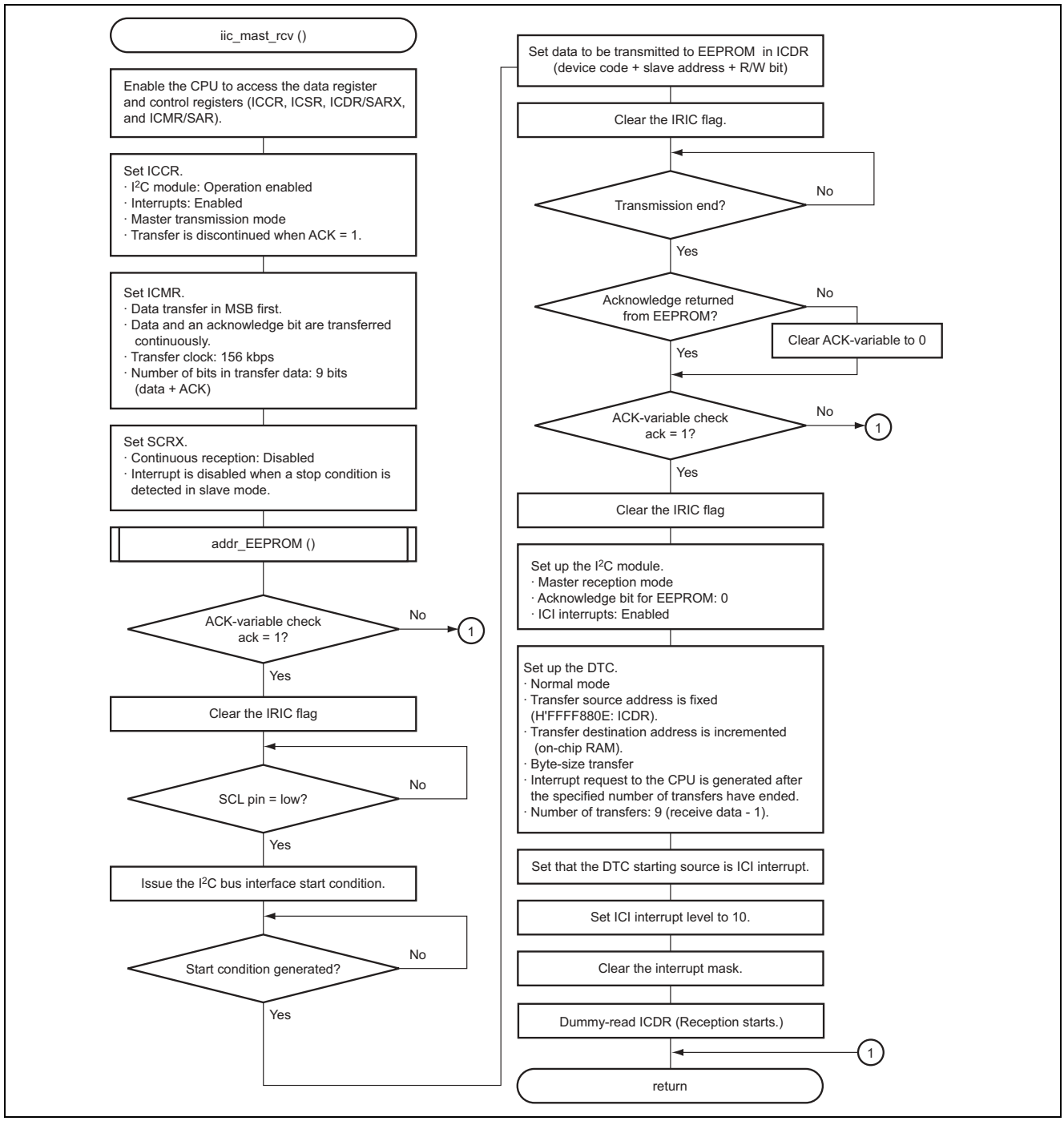
Label Name	Function	Address	Used in
RD_DATA[0-9]	Stores data to be read from the EEPROM (DTC transfer destination address during reception)	On-chip RAM	I²C master reception routine, I²C interrupt routine
Dummy	Stores dummy-read data	On-chip RAM	I²C master reception routine

2.5 Flowchart

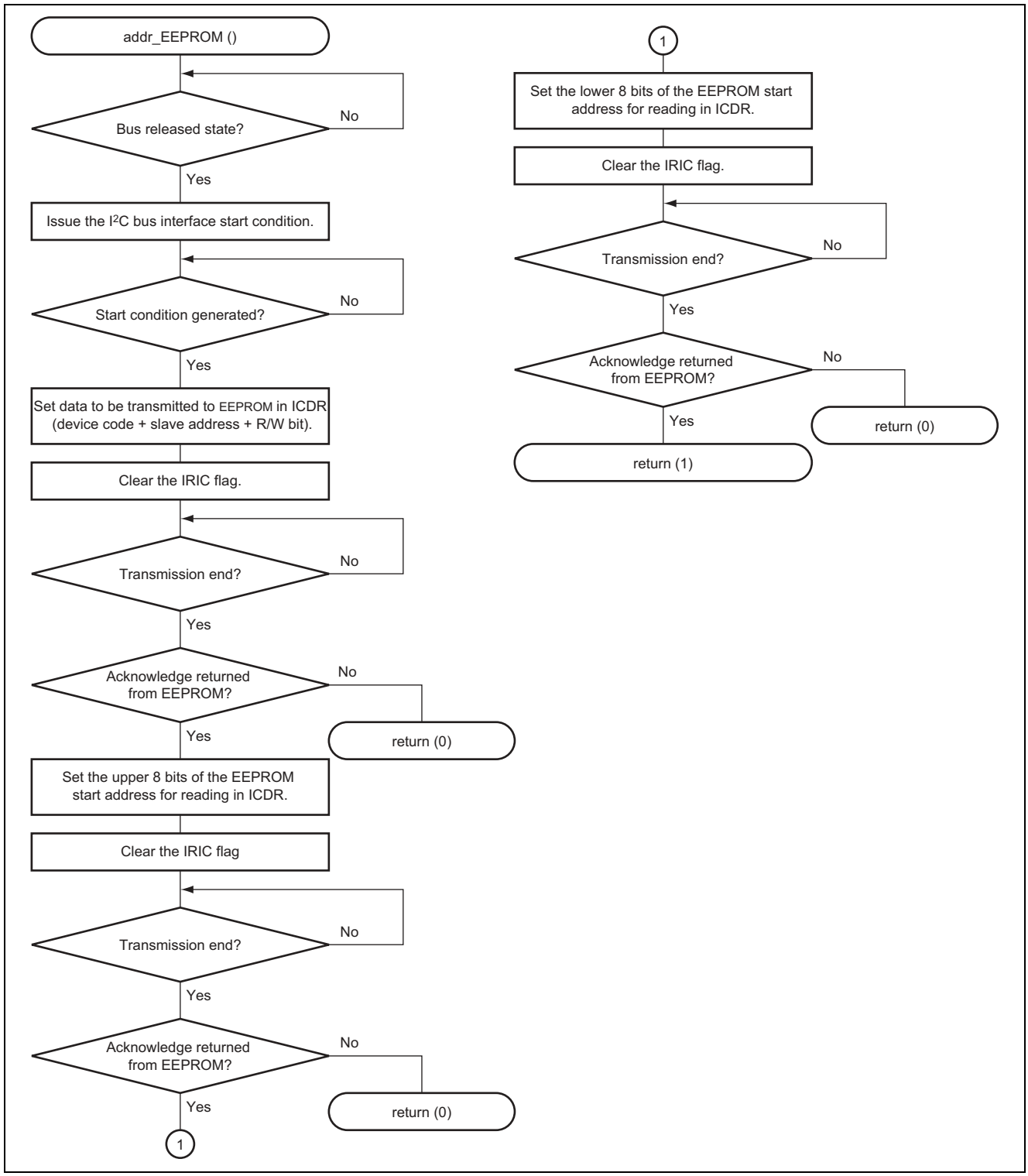
2.5.1 Main Routine



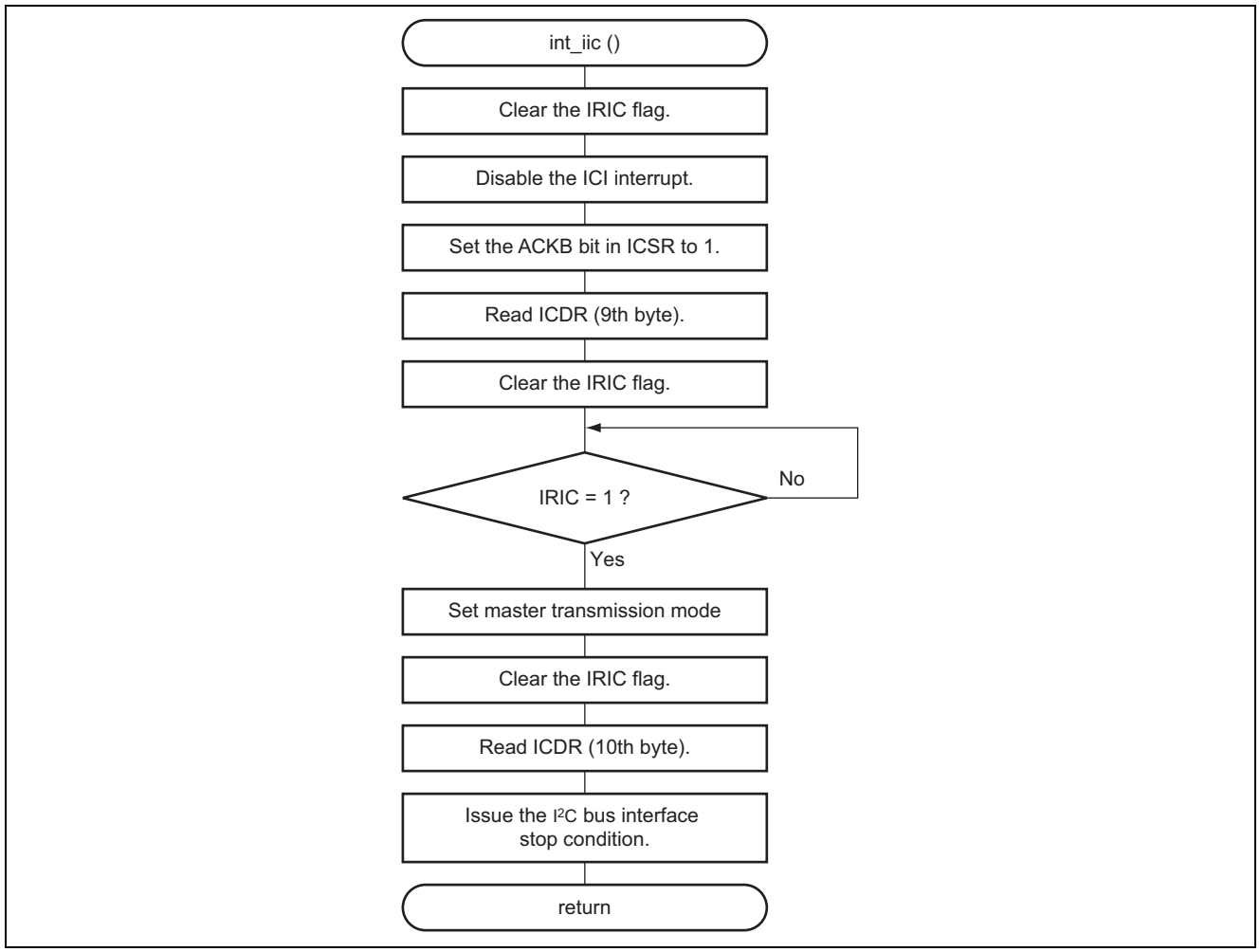
2.5.2 I²C Master Reception Routine



2.5.3 Address Transmission Routine



2.5.4 I²C Interrupt Routine



2.6 Program Listing

```
/************************************************************************************************/
/* SH7145F    Application Note                                                                  */
/*                                                                                              */
/* Function                                                                                     */
/*    :I2C,DTC(EEPROM;HN58X2464,64k bit,8word x 8bit)                                           */
/*    :Single Master Receive                                                                    */
/*                                                                                              */
/* External input clock         :10MHz                                                          */
/* Internal CPU clock           :40MHz                                                          */
/* Internal peripheral clock    :40MHz                                                          */
/*                                                                                              */
/* Written        :2004/1       Rev.1.0                                                         */
/************************************************************************************************/

#include "iodefine.h"
#include <machine.h>

/************************************************************************************************/
/*  Symbol Definition                                                                           */
/************************************************************************************************/
struct st_dtc_iic {
    unsigned short DTMR;
    unsigned short DTCRA;
    unsigned short dummy1;
    unsigned short dummy2;
    unsigned long  DTSAR;
    unsigned long  DTDAR;
};

#define D_CODE     0xA0                         /* Device code of EEPROM                   */
#define A_CODE     0x00                         /* Device address code of EEPROM           */
#define RD         0x01                         /* Read data B'1                           */
#define WR         0x00                         /* Write data B'0                          */

#define UP_ADDR    0x00                         /* Upper address of EEPROM                 */
#define LO_ADDR    0x00                         /* Lower address of EEPROM                 */

#define RD_NUM     10                           /* The number of read data from EEPROM     */

/************************************************************************************************/
/*  Function define                                                                             */
/************************************************************************************************/
void main(void);
void iic_mast_rcv(void);
unsigned char addr_EEPROM(void);

void int_iic(void);
void dummy_f(void);
```

```
/************************************************************************************************/
/*  RAM allocation Definition                                                                   */
/************************************************************************************************/
unsigned char RD_DATA[RD_NUM];                      /* Read data                                */
unsigned char Dummy;                                /* Dummy read data                          */

#define DTC_ICI (*(volatile struct st_dtc_iic*)0xFFFFF000)
                                                    /* DTC information address                  */
/************************************************************************************************/
/*  Main Program                                                                                */
/************************************************************************************************/
void main(void)
{
    P_STBY.MSTCR1.BIT.MSTP21 = 0;                   /* Disable IIC standby mode                 */

    /* Set of I2C pin function                                                                  */
    P_PORTB.PBCR1.WORD = 0x0C00;
    P_PORTB.PBCR2.WORD = 0x0000;                    /* SDA0(PB2-31pin@SH7145F),                 */
                                                    /* SCL0(PB2-31pin@SH7145F)                  */

    iic_mast_rcv();                                 /* Read routine                             */

    while(1);                                       /* LOOP                                     */
}
/************************************************************************************************/
/*  Function    : iic_mast_rcv                                                                  */
/*  Operation   : Data read from EEPROM by the I2C interface                                    */
/*  Argument    : Non-argument                                                                  */
/*  Return      : Non-return                                                                    */
/************************************************************************************************/
void iic_mast_rcv(void)
{
    unsigned short d_count,data;
    unsigned char ack;

    P_IIC.SCRX.BIT.IICE = 1;                        /* Register access enable from CPU          */

    P_IIC.ICCR0.BYTE = 0xB9;
            // ICE      [7]=B'1       : I2C interface enable
            // IECE     [6]=B'0       : I2C interrupt enable
            // MST      [5]=B'1       : Master mode
            // TRS      [4]=B'1       : Transmit mode
            // ACKE     [3]=B'1       : Continuous data transfer is halted
            // BBSY     [2]=B'0
            // IRIC     [1]=B'0
            // SCP      [0]=B'1
    P_IIC.ICMR0.BYTE = 0x38;
            // MLS      [7]=B'0       : MSB first
            // WAIT     [6]=B'0       : A wait state is inserted between DATA and ACK
            // CKS      [5-3]=B'111   : Transfer clock(with IICX0) = 156kHz(P phi=40MHz)
            // BC       [2-0]=B'000
```

```
P_IIC.SCRX.BYTE = 0x39;
        // Reserve [7-6]=B'00
        // IICX    [5]=B'1       : Transfer rate select, reference CKS bit
        // IICE    [4]=B'1       : Register access enable from CPU
        // HNDS    [3]=B'1
        // Reserve [2]=B'0
        // ICDRF   [1]=B'0
        // STOPIM  [0]=B'1       : Disables interrupt requests

ack = addr_EEPROM();

if(ack == 1){                                   /* ACK OK                          */
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                 */
    while(P_PORTB.PBDR.BIT.PB2DR != 0);         /* Check SCL0 pin                  */

    P_IIC.ICCR0.BYTE = ((P_IIC.ICCR0.BYTE & 0xFE) | 0x04);
                                                /* Generate start condition        */
    while(P_IIC.ICCR0.BIT.IRIC != 1);           /* Wait 1byte transmitted          */

    P_IIC.ICDR0.BYTE = (D_CODE|A_CODE|RD);
                                                /* Device code + R/W bit(1;Read)   */
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                 */
    while(P_IIC.ICCR0.BIT.IRIC != 1);           /* Wait 1byte transmitted          */
    if(P_IIC.ICSR0.BIT.ACKB != 0){              /* ACK = 1                         */
        ack = 0;                                /* EEPROM write error              */
    }
}

if(ack == 1){
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                 */

    P_IIC.ICCR0.BIT.TRS = 0;                    /* Receive mode                    */
    P_IIC.ICSR0.BIT.ACKB = 0;                   /* Set ACK=0                       */
    P_IIC.ICCR0.BIT.IEIC = 1;                   /* IIC interrupt enable            */

 /* DTC initialize*/
    DTC_ICI.DTMR = 0x2000;                      /* Set DTC mode                    */
        // SM      [15-14]=B'00 : Source address unchanging
        // DM      [13-12]=B'10 : Destination address increment
        // MD      [11-10]=B'00 : Normal mode
        // Sz      [9-8]=B'00   : Transfer data size = byte
        // DTS     [7]=B'0      : Not used
        // CHNE    [6]=B'0      : Not used
        // DISEL   [5]=B'0      : DTC interrupt disable
        // NMIM    [4]=B'0      : NMI->Terminate DTC transfer
        // Reserve [3-0]=B'0000
    DTC_ICI.DTCRA = (RD_NUM - 1);               /* Transfer count                  */
    DTC_ICI.DTSAR = (unsigned long)&(P_IIC.ICDR0.BYTE);
                                                /* Set source address              */
    DTC_ICI.DTDAR = (unsigned long)&(RD_DATA[0]);
                                                /* Set destination address         */
```

```
        P_DTC.DTBR = 0xFFFF;                    /* DTC information base register        */
        P_DTC.DTEG.BYTE |= 0x80;                /* Interrupt sources IIC                */

        P_INTC.IPRJ.BIT.IIC = 10;               /* Set IIC interrupt level              */
        set_imask(0);                           /* Clear interrupt mask level           */

        Dummy = P_IIC.ICDR0.BYTE;               /* Dummy read                           */
    }
}
/*****************************************************************************************/
/*  Function    : addr_EEPROM                                                            */
/*  Operation   : Transmission of a slave address and a memory-address                   */
/*  Argument    : Non-argument                                                           */
/*  Return      : ack                                                                    */
/*****************************************************************************************/
unsigned char addr_EEPROM(void)
{
    while(P_IIC.ICCR0.BIT.BBSY != 0);           /* Judging bus condition                */

    P_IIC.ICCR0.BYTE = ((P_IIC.ICCR0.BYTE & 0xFE) | 0x04);
                                                /* Generate start condition             */
    while(P_IIC.ICCR0.BIT.IRIC != 1);           /* Wait 1 byte transmitted              */

    // Transmission of a slave address
    P_IIC.ICDR0.BYTE = (D_CODE|A_CODE|WR);      /* Set of transmission data             */
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                      */
    while(P_IIC.ICCR0.BIT.IRIC != 1);           /* Wait 1 byte transmitted              */
    if(P_IIC.ICSR0.BIT.ACKB != 0){              /* Judging ACK bit                      */
        return(0);                              /* No ACK bit                           */
    }

    // Transmission of a upper-byte memory-address
    P_IIC.ICDR0.BYTE = UP_ADDR;                 /* Set of transmission data             */
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                      */
    while(P_IIC.ICCR0.BIT.IRIC != 1);           /* Wait 1 byte transmitted              */
    if(P_IIC.ICSR0.BIT.ACKB != 0){              /* Judging ACK bit                      */
        return(0);                              /* No ACK bit                           */
    }

    // Transmission of a lower-byte memory-address
    P_IIC.ICDR0.BYTE = LO_ADDR;                 /* Set of transmission data             */
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                      */
    while(P_IIC.ICCR0.BIT.IRIC != 1);           /* Wait 1byte transmitted               */
    if(P_IIC.ICSR0.BIT.ACKB != 0){              /* Judging ACK bit                      */
        return(0);                              /* No ACK bit                           */
    }
    return(1);                                  /* ACK OK                               */
}
```

```
/*****************************************************************************************************/
/*  Interruption Program                                                                             */
/*****************************************************************************************************/
/*****************************************************************************************************/
/*  Function   : int_iic                                                                             */
/*  Operation  : Reception end interruption                                                          */
/*  Argument   : Non-argument                                                                        */
/*  Return     : Non-return                                                                          */
/*****************************************************************************************************/
#pragma interrupt(int_iic)
void int_iic(void)
{
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                          */
    P_IIC.ICCR0.BIT.IEIC = 0;                   /* IEIC interrupt disable                   */

    P_IIC.ICSR0.BIT.ACKB = 1;                   /* Set of ACK(=1)                           */

    RD_DATA[8] = P_IIC.ICDR0.BYTE;              /* Read ICDR0 register                      */
    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                          */
    while(P_IIC.ICCR0.BIT.IRIC != 1);           /* Wait 1 byte to be received               */

    P_IIC.ICCR0.BIT.TRS = 1;                    /* IIC transmit mode                        */

    P_IIC.ICCR0.BIT.IRIC = 0;                   /* Clear IRIC flag                          */

    RD_DATA[9] = P_IIC.ICDR0.BYTE;              /* Read ICDR0 register                      */

    P_IIC.ICSR0.BIT.ACKB = 0;                   /* Set of ACK(=0)                           */

    P_IIC.ICCR0.BYTE = P_IIC.ICCR0.BYTE & 0xFA;
                                                /* Generate stop condition                  */
}
/*****************************************************************************************************/
/*  Other Interruption Program                                                                       */
/*****************************************************************************************************/
#pragma interrupt(dummy_f)
void dummy_f(void)
{
    /* Other Interrupt */
}
```

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Sep.16.04	—	First edition issued

Keep safety first in your circuit designs!

1. Renesas Technology Corp. 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 Corp. 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 Corp. or a third party.
2. Renesas Technology Corp. 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 Corp. without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. by various means, including the Renesas Technology Corp. 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 Corp. assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corp. 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 Corp. or an authorized Renesas Technology Corp. 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 Corp. 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 Corp. for further details on these materials or the products contained therein.