RENESAS APPLICATION NOTE

R8C/35C Group

I^{2}C bus Single Master Control Program (Slave Transmit/Receive)

R01AN0075EJ0100
Rev.1.00
Aug. 31, 2010

1. Abstract

This document describes the slave transmit/receive processes in the I^{2}C bus single master control program using the R8C/35C Group I^{2}C bus interface.

2. Introduction

The application example described in this document applies to the following microcomputer (MCU) and parameter:

- MCU: R8C/35C Group
- XIN Clock: 20 MHz

This application note can be used with other R8C Family MCUs which have the same special function registers (SFRs) as the above group. Check the manual for any modifications to functions. Careful evaluation is recommended before using the program described in this application note.

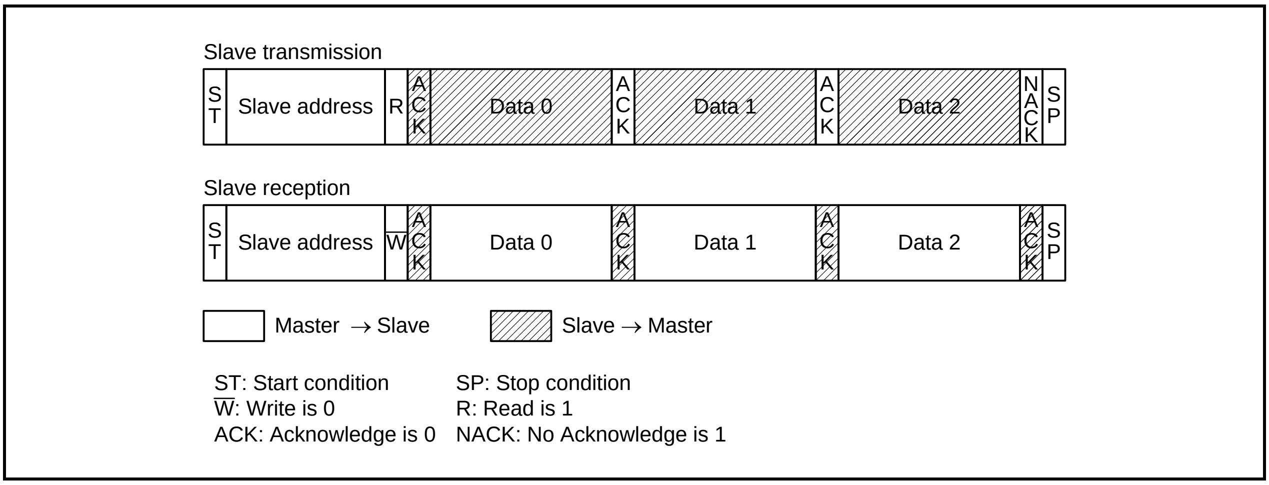
3. Application Example

3.1 Program Outline

Use the I2C bus interface to perform serial communication. A maximum of 255 bytes of data can be transmitted and received. This communication procedure conforms to the I2C bus communication protocol when used under the following conditions:

- Slave address: 7 bits
- Standard-mode and Fast-mode are supported.
- Transfer data length: 1 to 255 bytes (not including the slave address)
- Restart condition detection is not supported.

Figure 3.1 shows the Communication Format, Figure 3.2 shows the Block Diagram, Figure 3.3 shows the Outline Flowchart, and Figure 3.4 to Figure 3.6 show Timing Diagrams.



ST: Start condition
W: Write is 0
ACK: Acknowledge is 0

SP: Stop condition
R: Read is 1
NACK: No Acknowledge is 1

Figure 3.1 Communication Format

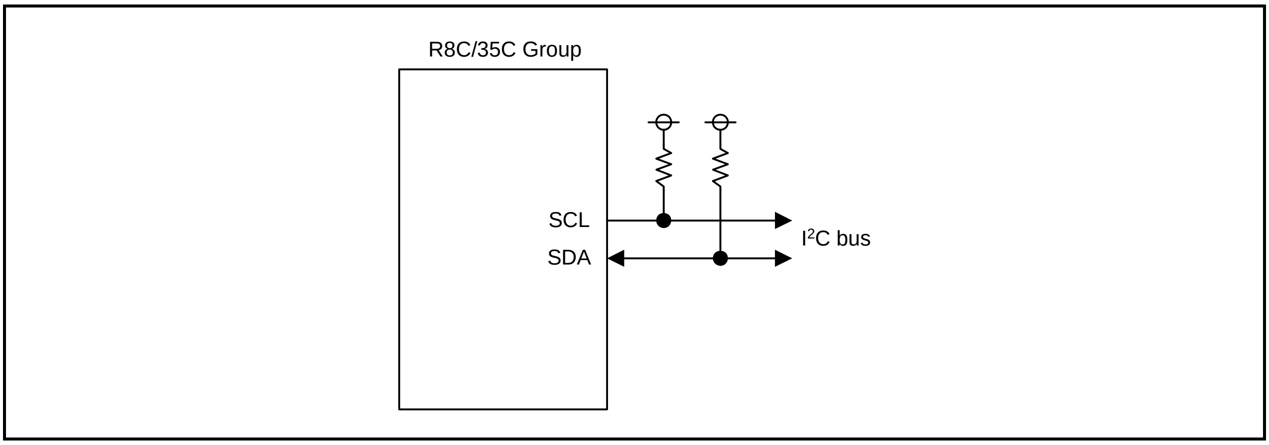


Figure 3.2 Block Diagram

The numbers in Figure 3.3 correspond to the numbers indicated in the program processing in the operating timing charts in Figure 3.4 to Figure 3.6.

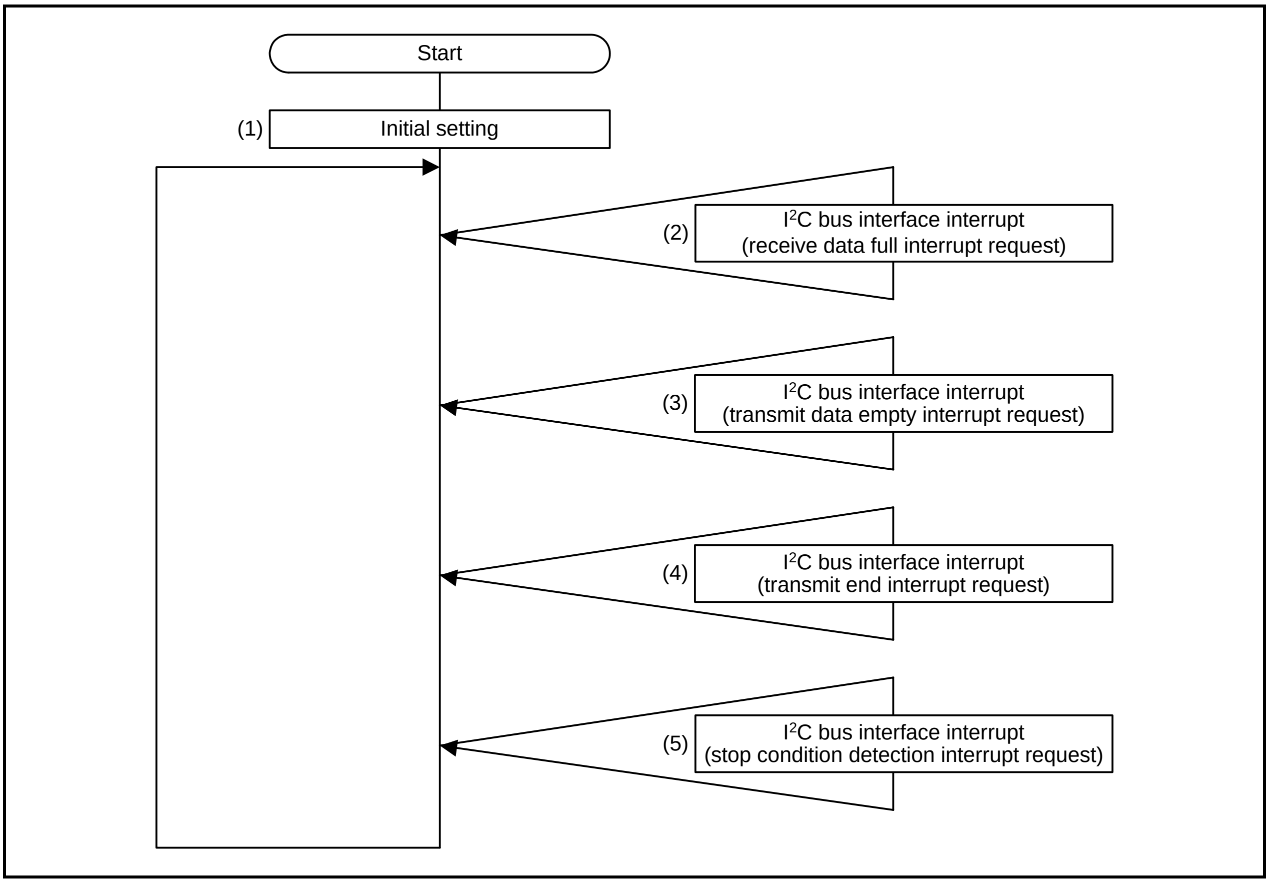

Start
(1)
Initial setting
(2)
I2C bus interface interrupt
(receive data full interrupt request)
(3)
I2C bus interface interrupt
(transmit data empty interrupt request)
(4)
I2C bus interface interrupt
(transmit end interrupt request)
(5)
I2C bus interface interrupt
(stop condition detection interrupt request)

Figure 3.3 Outline Flowchart

A description of the process outline is as follows:

(1) Initial setting
Initialize the system clock, I2C bus interface associated SFRs, and variables used.

(2) I2C bus interface interrupt (receive data full interrupt request)
Before the slave address is matched:
When the following conditions are met, an interrupt is generated at the rising edge of the ninth bit of the SCL clock.
• The slave address is matched at the first byte after a start condition is detected.
• The data at the eighth byte (R/$\overline{W}$) is 0.
After the slave address is matched:
An interrupt is generated at the rising edge of the ninth bit of the SCL clock. The received data is stored at reception.

(3) I2C bus interface interrupt (transmit data empty interrupt request)
When the following conditions are met, an interrupt is generated at the rising edge of the ninth bit of the SCL clock.
• The slave address is matched at the first byte after a start condition is detected.
• The data at the eighth byte (R/$\overline{W}$) is 1.
Disable the transmit data empty interrupt request and receive data full interrupt request. Enable the transmit end interrupt request.

(4) I2C bus interface interrupt (transmit end interrupt request)
An interrupt is generated at the rising edge of the ninth bit of the SCL clock. Determine ACK/NACK and set the transmit data for the next byte.
When NACK is detected:
• Set the TRS bit in the ICCR1 register to receive mode.
• Disable the transmit end interrupt request and enable the receive data full interrupt request.

(5) I2C bus interface interrupt (stop condition detection interrupt request)
An interrupt is generated when a stop condition is detected. Disable the stop condition detection interrupt request. Set the TRS bit to receive mode, enable the transmit data empty interrupt request, and receive the data full interrupt request.

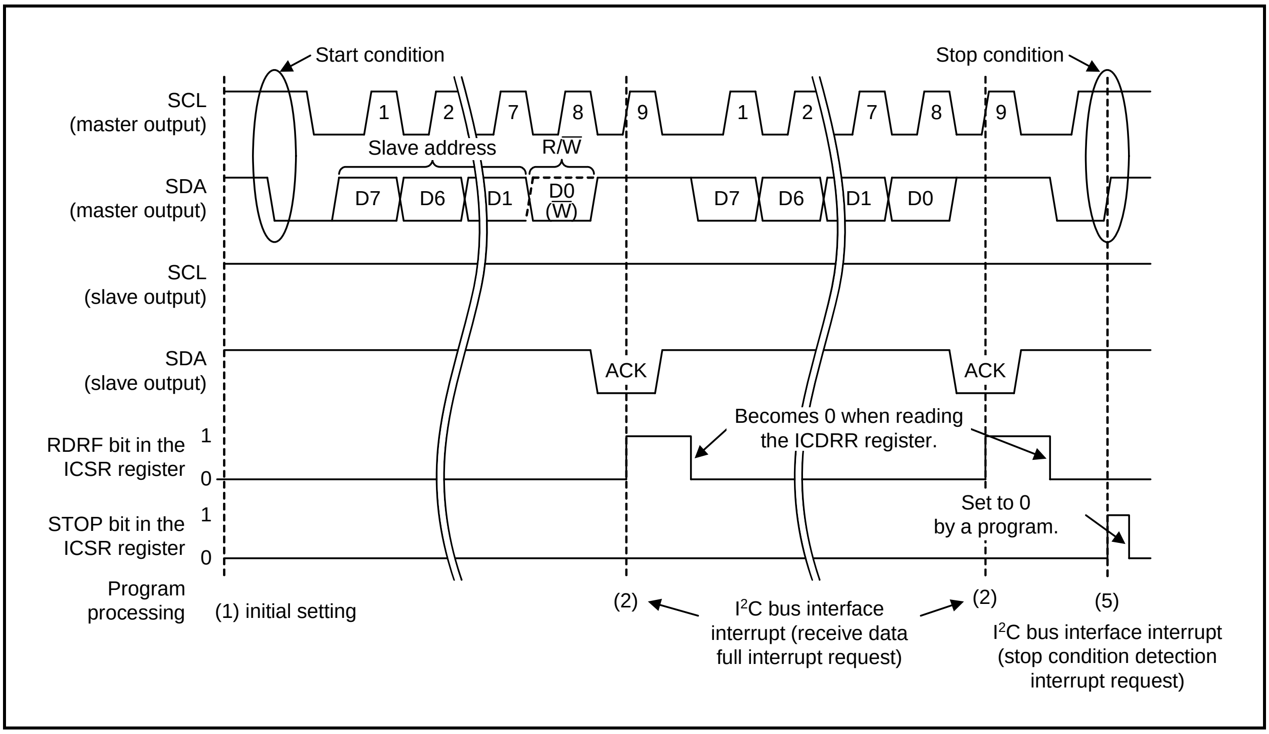


Figure 3.4 Slave Receive Timing

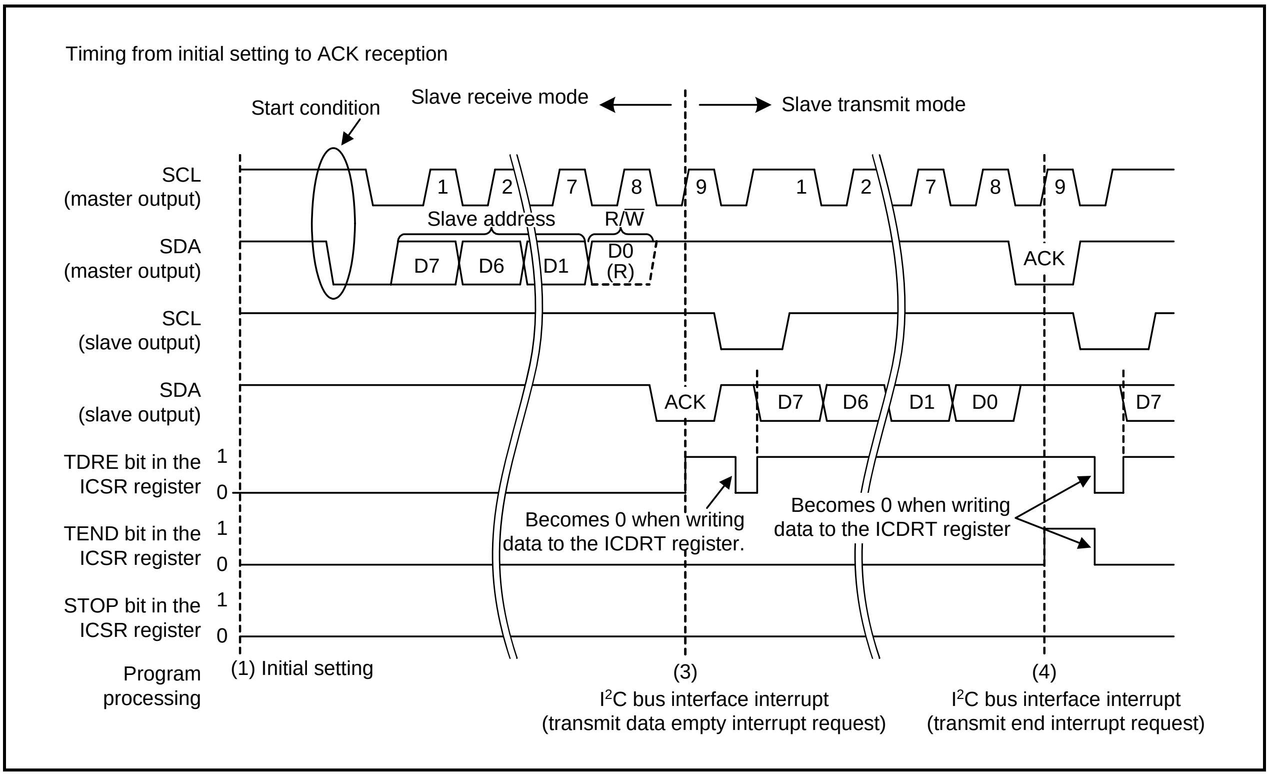

Timing from initial setting to ACK reception
Start condition
Slave receive mode
Slave transmit mode
SCL (master output)
SDA (master output)
SCL (slave output)
SDA (slave output)
TDRE bit in the ICSR register
TEND bit in the ICSR register
STOP bit in the ICSR register
Program processing
1
2
7
8
9
Slave address
R/W
D7
D6
D1
D0 (R)
ACK
D0
1
0
Becomes 0 when writing data to the ICDRT register.
Becomes 0 when writing data to the ICDRT register
(1) Initial setting
(3) I2C bus interface interrupt (transmit data empty interrupt request)
(4) I2C bus interface interrupt (transmit end interrupt request)

Figure 3.5 Slave Transmit Timing (1)

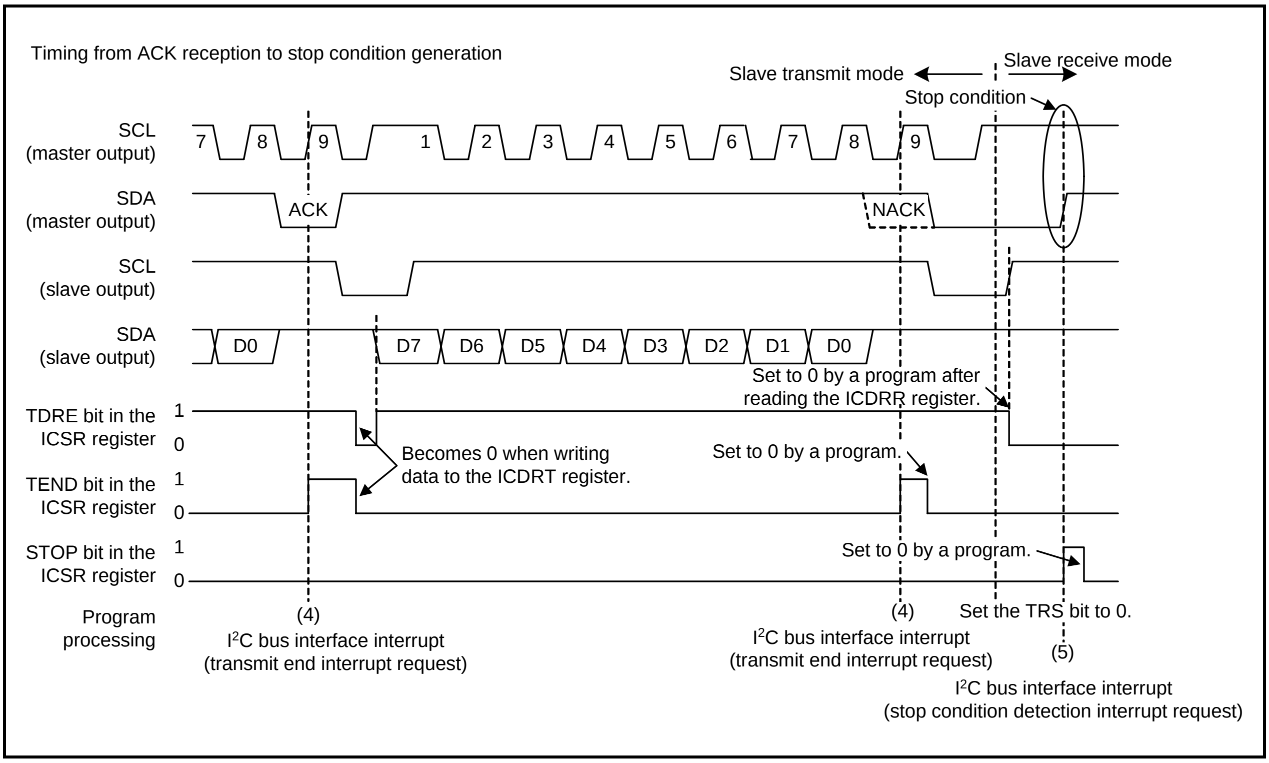

Timing from ACK reception to stop condition generation
Slave transmit mode
Slave receive mode
Stop condition
SCL (master output)
SDA (master output)
SCL (slave output)
SDA (slave output)
TDRE bit in the ICSR register
TEND bit in the ICSR register
STOP bit in the ICSR register
Program processing
7
8
9
1
2
3
4
5
6
ACK
NACK
D0
D7
D6
D5
D4
D3
D2
D1
1
0
Set to 0 by a program after reading the ICDRR register.
Becomes 0 when writing data to the ICDRT register.
Set to 0 by a program.
Set to 0 by a program.
Set the TRS bit to 0.
(4) I2C bus interface interrupt (transmit end interrupt request)
(4) I2C bus interface interrupt (transmit end interrupt request)
(5) I2C bus interface interrupt (stop condition detection interrupt request)

Figure 3.6 Slave Transmit Timing (2)

3.1.1 Peripheral Functions

The I^2C bus interface mode of the I^2C bus interface is used under the following setting conditions:

- I^2C bus format is used.
- MSB first is used for the transfer format.
- No wait states are set (data and the acknowledge bit are transferred consecutively).
- 20 Tcyc is selected for the setup time in transmit mode.
- 3 × f1 cycles are used for the SDA digital delay value.
- The ACKBR bit in the ICIER register is used to determine an acknowledge signal.
- The AAS bit in the ICSR register is used to detect the slave address.
- The receive data full interrupt request is used.
- The transmit end interrupt request is used.
- The transmit data empty interrupt request is used.
- The stop condition detection interrupt request is used.
- The NACK receive interrupt request and arbitration lost/overrun error interrupt request are not used.

Calculating the setup time in transmit mode
Setup time = CKS3 bit in the ICCR1 register setting
= 20 ÷ 20 MHz (f1)
= 1 μs

Table 3.1 Pins Used and Their Functions

Pin	I/O	Function
P3_5/SCL	I/O	I^2C bus clock I/O pin
P3_7/SDA	I/O	I^2C bus data I/O pin

3.1.2 Notes on Using the Attached Sample Program

Note the following when using the program included with this application note:

- Do not use multiple interrupts.
- When setting the system clock to anything other than the 20 MHz XIN clock, change the setting value of the CKS3 bit according to the setup time calculation shown in **3.1.1 Peripheral Functions**.
- The transmit/receive buffer sizes are set to 255 bytes. Use BUFSIZE in the iic.h file to set the buffer size (1 to 255 bytes).
- After a master generates a stop condition and the slave processing time [1] passes, start the next transmission and reception (generate a start condition).

Note

1. The slave processing time shows the time between the stop condition detection and time to enable the I^2C module in the main processing, and depends on a user program. The processing time in the attached sample program is approximately 500 μs.

3.2 Memory

Table 3.2 Memory

Memory	Size	Remarks
ROM	622 bytes	In the iic.c module
RAM	4 bytes	In the iic.c module
Maximum user stack	21 bytes	
Maximum interrupt stack	24 bytes	

Memory size varies depending on the C compiler version and compile options.
The above applies to the following conditions:
C compiler: M16C Series, R8C Family C Compiler V.5.45 Release 01
Compile options: -c -finfo -dir “$(CONFIGDIR)” -R8C

4. Software

This section shows the program example to set the example described in section **3. Application Example**. Refer to the latest **R8C/35C Group** hardware user's manual for details on individual registers.

4.1 Variables

Table 4.1 Definition File Name: r01an0075_src.c

Variable Name	Size	Description
unsigned char iic_tx[BUFSIZE]	255 bytes	Transmit buffer
unsigned char iic_rx[BUFSIZE]	255 bytes	Receive buffer
unsigned char rcv_data[BUFSIZE]	255 bytes	Store received data

Table 4.2 Definition File Name: iic.c

Variable Name		Size /Bit-number	Description
static byte_dt iic_str		—	Structure to store status
Structure member	iic_status	1 byte	All statuses
	iic_rw	b0	R/$\overline{W}$ flag 0: Write ($\overline{W}$) 1: Read (R)
	iic_buf_full	b1	Buffer full flag 0: Less than buffer size 1: Buffer full
	iic_end	b2	Communication end flag 0: Busy (mid-communication) 1: Ready (not mid-communication)
	iic_nack_det	b3	NACK detection flag 0: No NACK detection 1: NACK detection
	—	b7 to b4	Not used (undefined)
unsigned char far *iic_pointer		2 bytes	Transmit/receive buffer pointer
unsigned char iic_index		1 byte	Number of transmit/receive bytes

4.2 Function Tables

Declaration	void main(void)		
Outline	Main processing		
Argument	Argument name		Meaning
	None		—
Variable (global)	Variable name		Contents
	unsigned char iic_tx[BUFSIZE]		Transmit buffer
	unsigned char iic_rx[BUFSIZE]		Receive buffer
	unsigned char rcv_data[BUFSIZE]		Store received data
Returned value	Type	Value	Meaning
	None	—	—
Function	After setting the system clock, enable the I2C module. Use the returned value of the iic_slave_end function to determine the communication state. When communication is completed, perform processing on each status, call the iic_init function, and enable the I2C module.		

Declaration	void mcu_init(void)		
Outline	System clock setting		
Argument	Argument name		Meaning
	None		—
Variable (global)	Variable name		Contents
	None		—
Returned value	Type	Value	Meaning
	None	—	—
Function	This function is called from the main processing. Set the system clock (XIN clock).		

Declaration	void iic_init(unsigned char ini)		
Outline	Initial setting of I2C bus interface		
Argument	Argument name	Meaning	
	unsigned char ini	0: I2C module disabled 1: I2C module enabled	
Variable (global)	Variable name		Contents
	(Structure member) iic_status		All statuses
Returned value	Type	Value	Meaning
	None	—	—
Function	This function is called from the main processing. Initialize SFRs to use the I2C bus interface. When the I2C module is enabled, set iic_status to 00h (all statuses are cleared). Interrupts are disabled by the I flag while this function is being executed.		

Declaration	void _ssuic(void)		
Outline	I^{2}C bus interface interrupt handling		
Argument	Argument name		Meaning
	None		—
Variable (global)	Variable name		Contents
	unsigned char iic_index		Number of transmit/receive bytes
	unsigned char *iic_pointer		Transmit/receive buffer pointer
	(Structure member) iic_status		All statuses
	(Structure member) iic_rw		$R/\overline{W}$ flag
Returned value	Type	Value	Meaning
	None	—	—
Function	An interrupt is generated at the rising edge of the ninth bit of the SCL clock or when a stop condition is detected. When a stop condition is detected: •Call the stp_int function. When a stop condition is not detected: •When the slave address is detected, clear the AAS bit, number of transmit/receive bytes, and all statuses. Disable the transmit data empty interrupt request and enable the stop condition interrupt request. Obtain the buffer address and set the $R/\overline{W}$ flag. •Call the slave_trn_int function at slave transmit and the slave_rcv_int function at slave receive.		

Declaration	unsigned char* iic_get_address(unsigned char rw)		
Outline	Obtain buffer address processing		
Argument	Argument name		Meaning
	unsigned char rw		$R/\overline{W}$ flag
Variable (global)	Variable name		Contents
	None		—
Returned value	Type	Value	Meaning
	unsigned char*	iic_rx	Receive buffer address
		iic_tx	Transmit buffer address
Function	This function is called from the I^{2}C bus interface interrupt handling. Determine the $R/\overline{W}$ flag and return the buffer address.		

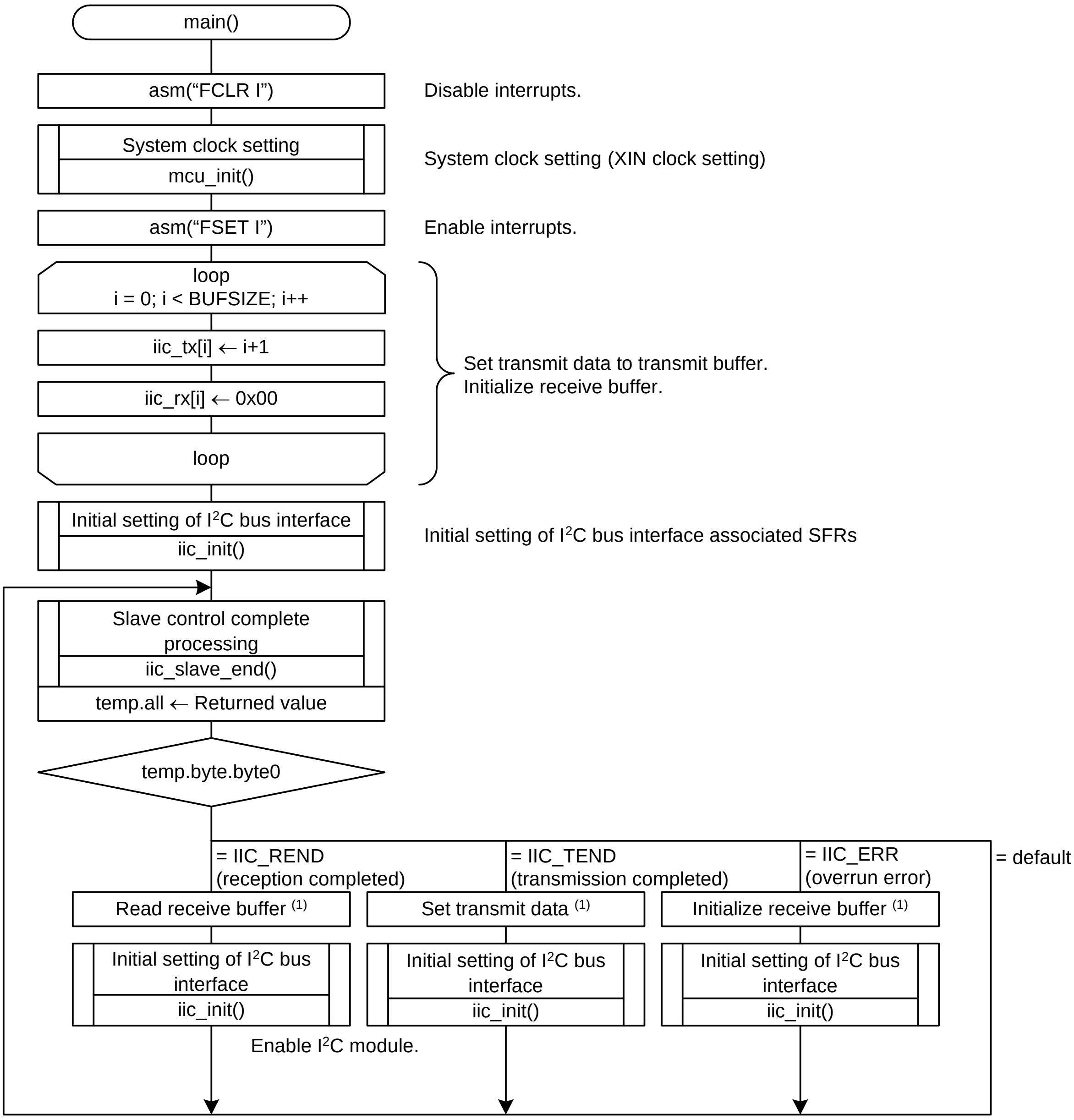
Declaration	static void stp_int(void)		
Outline	Stop condition detection processing		
Argument	Argument name		Meaning
	None		—
Variable (global)	Variable name		Contents
	(Structure member) iic_end		Communication end flag
Returned value	Type	Value	Meaning
	None	—	—
Function	This function is called from the I^2C bus interface interrupt handling. Reset the I^2C bus interface associated SFRs changed during communication and set the communication end flag to 1.		

Declaration	static void slave_rcv_int(void)		
Outline	Slave receive processing		
Argument	Argument name		Meaning
	None		—
Variable (global)	Variable name		Contents
	unsigned char iic_index		Number of transmit/receive bytes
	unsigned char far *iic_pointer		Transmit/receive buffer pointer
	(Structure member) iic_buf_full		Buffer full flag
Returned value	Type	Value	Meaning
	None	—	—
Function	This function is called from the I^2C bus interface interrupt handling. • When the number of receive bytes has not reached the buffer size, store the received data to the receive buffer (not the slave address). • When the number of receive bytes has reached the buffer size, store the received data to the receive buffer and set the buffer full flag to 1. • When the number of receive bytes is greater than the buffer size, discard the received data.		

Declaration	static void slave_trn_int(void)		
Outline	Slave transmit processing		
Argument	Argument name		Meaning
	None		—
Variable (global)	Variable name		Contents
	unsigned char iic_index		Number of transmit/receive bytes
	unsigned char far *iic_pointer		Transmit/receive buffer pointer
	(structure member) iic_buf_full		Buffer full flag
	(structure member) iic_nack_det		NACK detection flag
Returned value	Type	Value	Meaning
	None	—	—
Function	This function is called from the I^{2}C bus interface interrupt handling. • When the number of transmit bytes has not reached the buffer size and ACK is detected, set the transmit data for the next byte. • When the number of transmit bytes has not reached the buffer size and NACK is detected, set to slave receive mode. Disable the transmit end interrupt request and enable the receive data full interrupt request. Set the NACK detection flag to 1. • When the number of transmit bytes has reached the buffer size, set to slave receive mode. Disable the transmit end interrupt request and enable the receive data full interrupt request. Set the buffer full flag to 1. • When the NACK detection error flag is 1 or the buffer full flag is 1, discard the received data.		

Declaration	unsigned short iic_slave_end(void)			
Outline	Slave control complete processing			
Argument	Argument name			Meaning
	None			—
Variable (global)	Variable name			Contents
	(Structure member) iic_end			Communication end flag
	(Structure member) iic_buf_full			Buffer full flag
	(Structure member) iic_rw			$R/\overline{W}$ flag
	unsigned char iic_index			Number of transmit/receive bytes
Returned value	Type		Value	Meaning
	unsigned short	Low-order byte	IIC_BUSY	Mid-communication
			IIC_REND	Reception completed
			IIC_TEND	Transmission completed
			IIC_ERR	Overrun error detected
		High-order byte	1 to 255	Number of transmit/receive bytes
Function	This function is called from the main processing and informs the user of the slave control complete state. When the communication end flag is 1 and data excluding the slave address is transmitted or received, disable the I^{2}C module. Otherwise return IIC_BUSY (mid-communication). When the communication end flag is 0 after disabling the I^{2}C module, determine when the next communication starts and return IIC_ERR (overrun error detected). When the communication end flag is 1, return IIC_REND (reception completed) or IIC_TEND (transmission completed).			

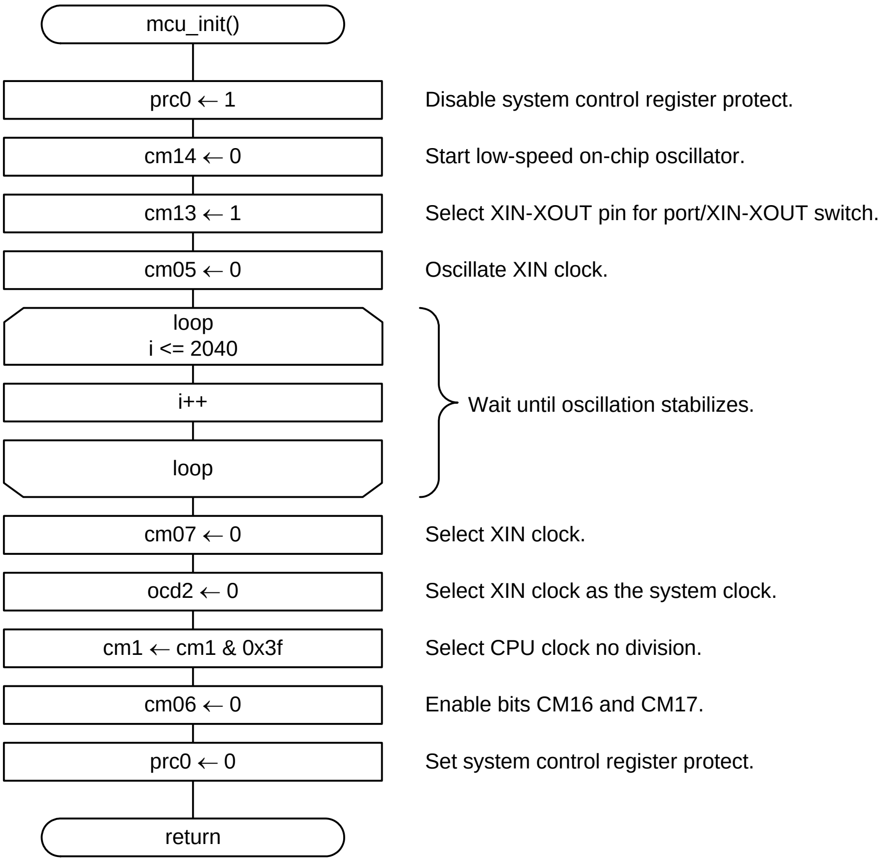
4.3 Main Processing


main()
asm(“FCLR I”)
Disable interrupts.
System clock setting
mcu_init()
System clock setting (XIN clock setting)
asm(“FSET I”)
Enable interrupts.
loop
i = 0; i < BUFSIZE; i++
iic_tx[i] ← i+1
iic_rx[i] ← 0x00
loop
Set transmit data to transmit buffer.
Initialize receive buffer.
Initial setting of I2C bus interface
iic_init()
Initial setting of I2C bus interface associated SFRs
Slave control complete processing
iic_slave_end()
temp.all ← Returned value
temp.byte.byte0
= IIC_REND (reception completed)
= IIC_TEND (transmission completed)
= IIC_ERR (overrun error)
= default
Read receive buffer (1)
Set transmit data (1)
Initialize receive buffer (1)
Initial setting of I2C bus interface
iic_init()
Enable I2C module.

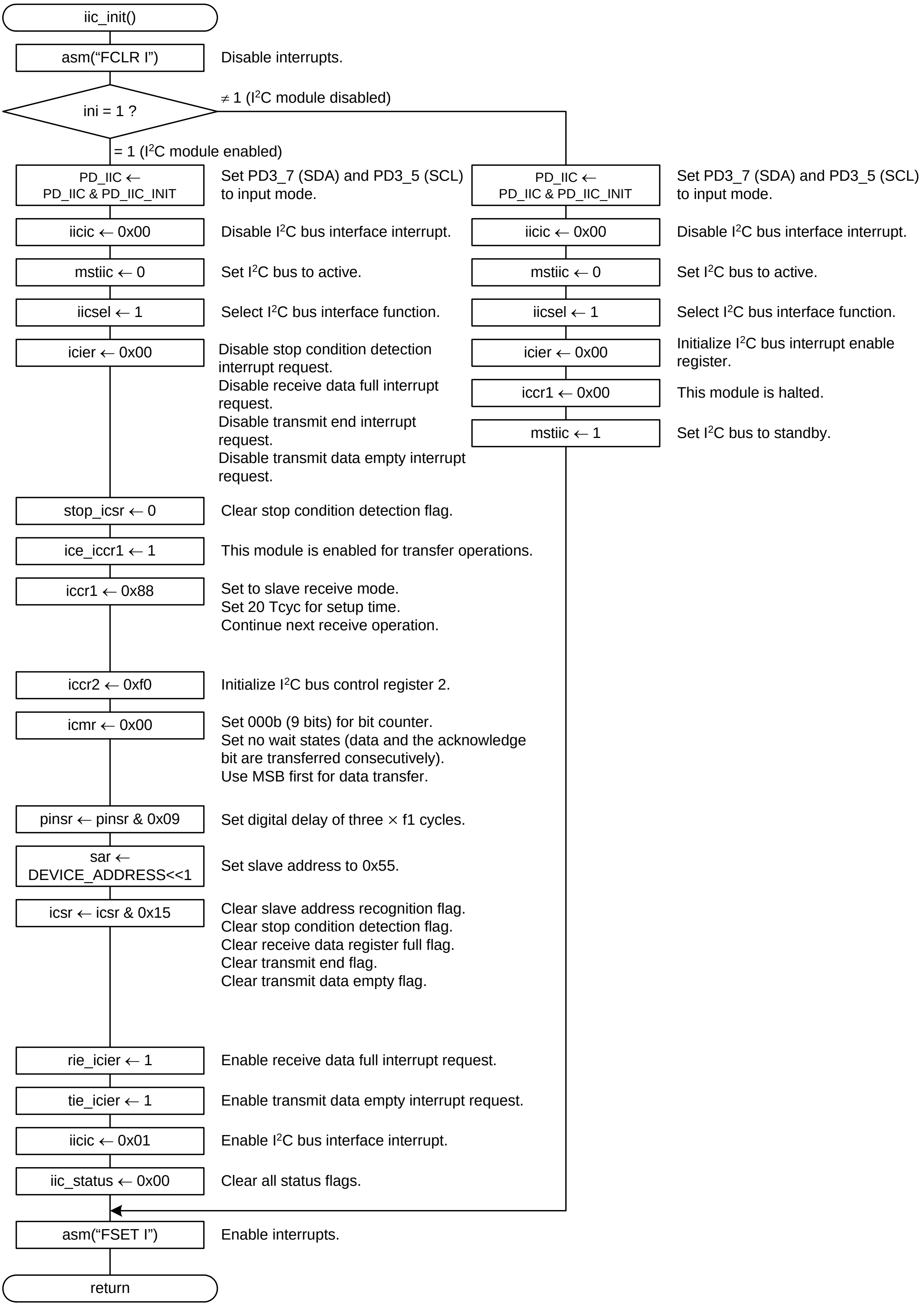
Note:

1. Additional processing can be added as needed.

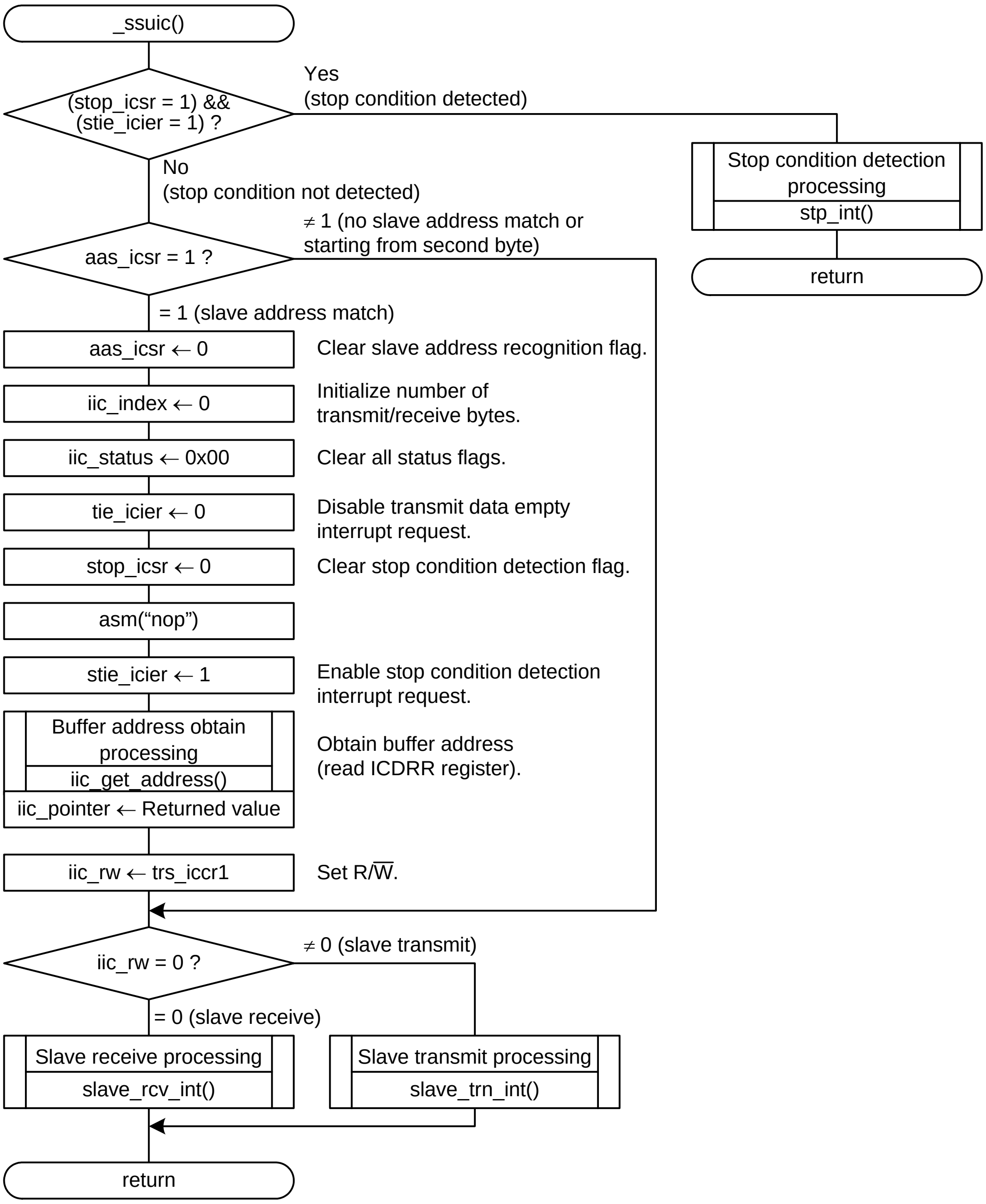
4.4 System Clock Setting


mcu_init()
prc0 ← 1
Disable system control register protect.
cm14 ← 0
Start low-speed on-chip oscillator.
cm13 ← 1
Select XIN-XOUT pin for port/XIN-XOUT switch.
cm05 ← 0
Oscillate XIN clock.
loop
i <= 2040
i++
loop
Wait until oscillation stabilizes.
cm07 ← 0
Select XIN clock.
ocd2 ← 0
Select XIN clock as the system clock.
cm1 ← cm1 & 0x3f
Select CPU clock no division.
cm06 ← 0
Enable bits CM16 and CM17.
prc0 ← 0
Set system control register protect.
return

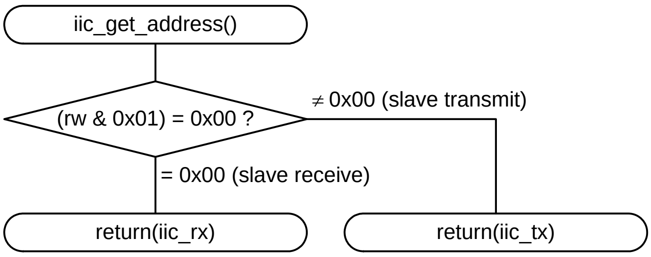
4.5 Initial Setting of I^{2}C bus Interface


iic_init()
asm("FCLR I")
Disable interrupts.
ini = 1 ?
≠ 1 (I2C module disabled)
= 1 (I2C module enabled)
PD_IIC ← PD_IIC & PD_IIC_INIT
Set PD3_7 (SDA) and PD3_5 (SCL) to input mode.
iicic ← 0x00
Disable I2C bus interface interrupt.
mstiic ← 0
Set I2C bus to active.
iicsel ← 1
Select I2C bus interface function.
icier ← 0x00
Disable stop condition detection interrupt request.
Disable receive data full interrupt request.
Disable transmit end interrupt request.
Disable transmit data empty interrupt request.
stop_icsr ← 0
Clear stop condition detection flag.
ice_iccr1 ← 1
This module is enabled for transfer operations.
iccr1 ← 0x88
Set to slave receive mode.
Set 20 Tcyc for setup time.
Continue next receive operation.
iccr2 ← 0xf0
Initialize I2C bus control register 2.
icmr ← 0x00
Set 000b (9 bits) for bit counter.
Set no wait states (data and the acknowledge bit are transferred consecutively).
Use MSB first for data transfer.
pinsr ← pinsr & 0x09
Set digital delay of three × f1 cycles.
sar ← DEVICE_ADDRESS<<1
Set slave address to 0x55.
icsr ← icsr & 0x15
Clear slave address recognition flag.
Clear stop condition detection flag.
Clear receive data register full flag.
Clear transmit end flag.
Clear transmit data empty flag.
rie_icier ← 1
Enable receive data full interrupt request.
tie_icier ← 1
Enable transmit data empty interrupt request.
iicic ← 0x01
Enable I2C bus interface interrupt.
iic_status ← 0x00
Clear all status flags.
PD_IIC ← PD_IIC & PD_IIC_INIT
Set PD3_7 (SDA) and PD3_5 (SCL) to input mode.
iicic ← 0x00
Disable I2C bus interface interrupt.
mstiic ← 0
Set I2C bus to active.
iicsel ← 1
Select I2C bus interface function.
icier ← 0x00
Initialize I2C bus interrupt enable register.
iccr1 ← 0x00
This module is halted.
mstiic ← 1
Set I2C bus to standby.
asm("FSET I")
Enable interrupts.
return

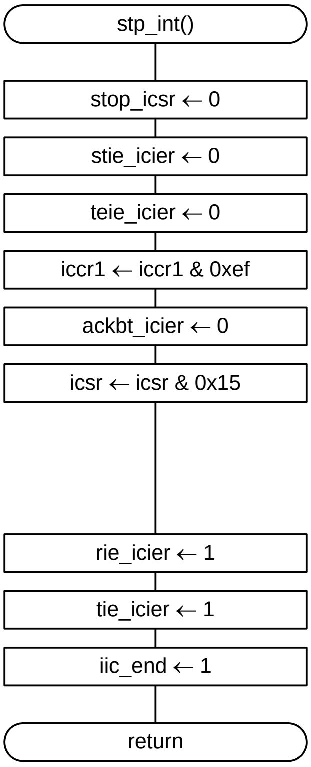
4.6 I^{2}C bus Interface Interrupt Handling


_ssuic()
(stop_icsr = 1) &&
(stie_icier = 1) ?
Yes
(stop condition detected)
Stop condition detection processing
stp_int()
return
No
(stop condition not detected)
aas_icsr = 1 ?
≠ 1 (no slave address match or starting from second byte)
= 1 (slave address match)
aas_icsr ← 0
Clear slave address recognition flag.
iic_index ← 0
Initialize number of transmit/receive bytes.
iic_status ← 0x00
Clear all status flags.
tie_icier ← 0
Disable transmit data empty interrupt request.
stop_icsr ← 0
Clear stop condition detection flag.
asm("nop")
stie_icier ← 1
Enable stop condition detection interrupt request.
Buffer address obtain processing
iic_get_address()
iic_pointer ← Returned value
Obtain buffer address (read ICDRR register).
iic_rw ← trs_iccr1
Set R/W̄.
iic_rw = 0 ?
≠ 0 (slave transmit)
= 0 (slave receive)
Slave receive processing
slave_rcv_int()
Slave transmit processing
slave_trn_int()
return

4.7 Obtain Buffer Address Processing



4.8 Stop Condition Detection Processing



Clear stop condition detection flag.

Disable stop condition detection interrupt request.

Disable transmit end interrupt request.

Set to slave receive mode.

In receive mode, transmit 0 as the acknowledge bit.

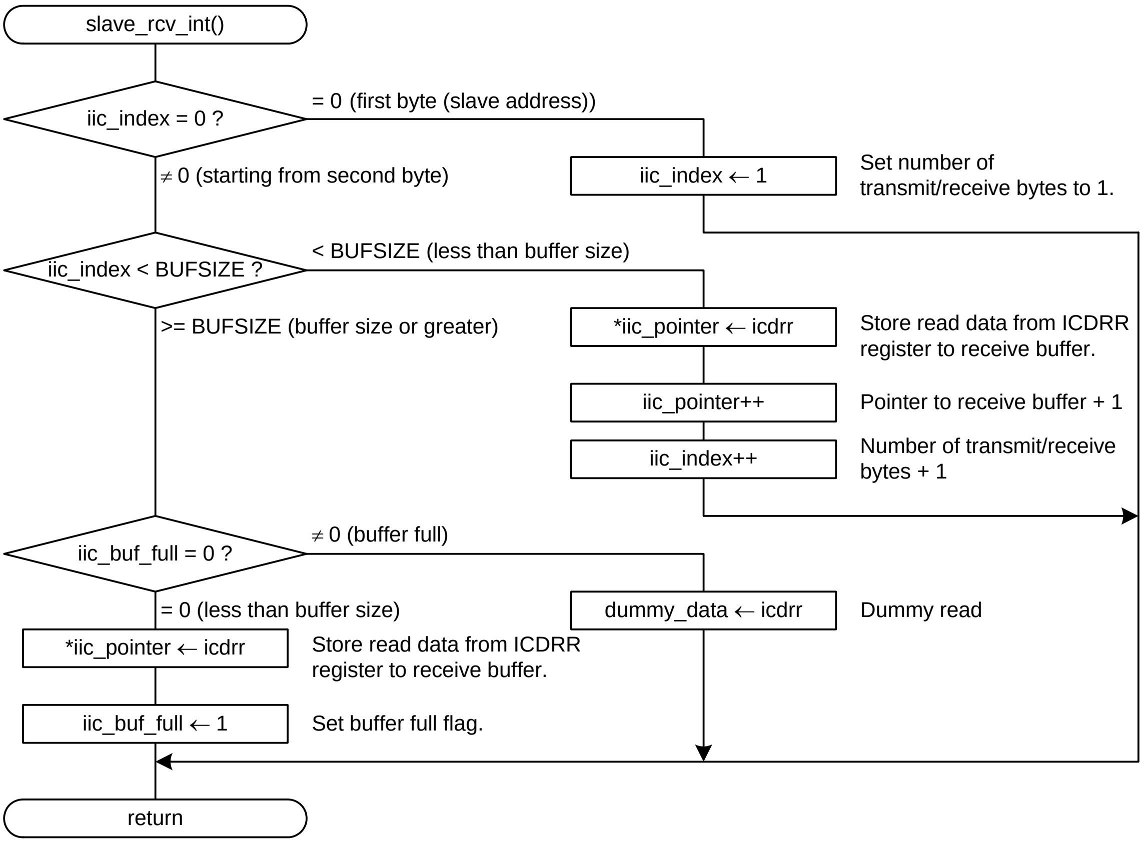
Clear slave address recognition flag.
Clear receive data register full flag.
Clear transmit end flag.
Clear transmit data empty flag.

Enable receive data full interrupt request.

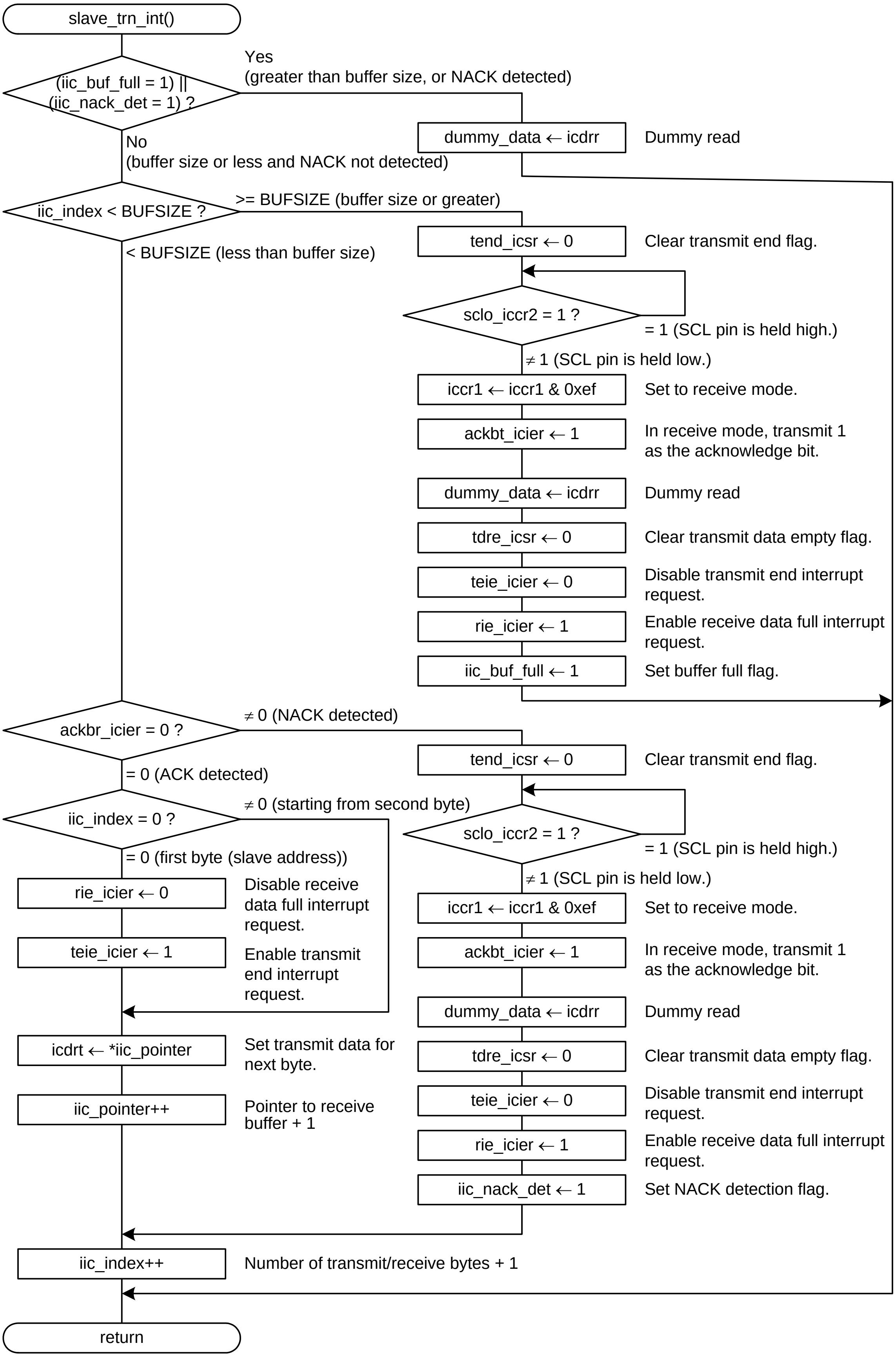
Enable transmit data empty interrupt request.

Set communication end flag.

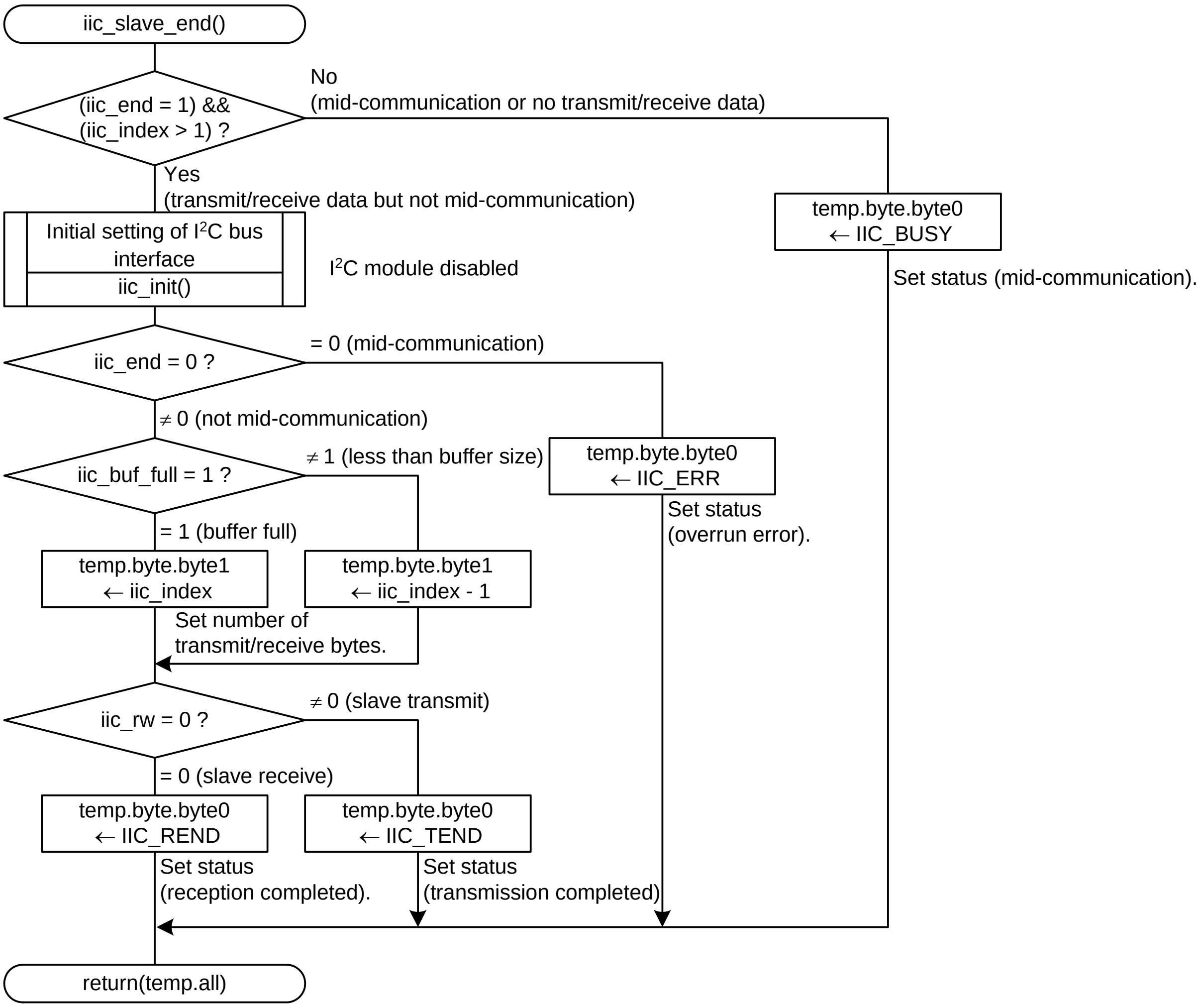
4.9 Slave Receive Processing


slave_rcv_int()
iic_index = 0 ?
= 0 (first byte (slave address))
≠ 0 (starting from second byte)
iic_index ← 1
Set number of transmit/receive bytes to 1.
iic_index < BUFSIZE ?
< BUFSIZE (less than buffer size)
>= BUFSIZE (buffer size or greater)
*iic_pointer ← icdrr
Store read data from ICDRR register to receive buffer.
iic_pointer++
Pointer to receive buffer + 1
iic_index++
Number of transmit/receive bytes + 1
iic_buf_full = 0 ?
≠ 0 (buffer full)
= 0 (less than buffer size)
dummy_data ← icdrr
Dummy read
*iic_pointer ← icdrr
Store read data from ICDRR register to receive buffer.
iic_buf_full ← 1
Set buffer full flag.
return

4.10 Slave Transmit Processing


slave_trn_int()
(iic_buf_full = 1) || (iic_nack_det = 1) ?
Yes (greater than buffer size, or NACK detected)
dummy_data ← icdrr
Dummy read
No (buffer size or less and NACK not detected)
iic_index < BUFSIZE ?
>= BUFSIZE (buffer size or greater)
tend_icsr ← 0
Clear transmit end flag.
< BUFSIZE (less than buffer size)
sclo_iccr2 = 1 ?
= 1 (SCL pin is held high.)
≠ 1 (SCL pin is held low.)
iccr1 ← iccr1 & 0xef
Set to receive mode.
ackbt_icier ← 1
In receive mode, transmit 1 as the acknowledge bit.
dummy_data ← icdrr
Dummy read
tdre_icsr ← 0
Clear transmit data empty flag.
teie_icier ← 0
Disable transmit end interrupt request.
rie_icier ← 1
Enable receive data full interrupt request.
iic_buf_full ← 1
Set buffer full flag.
ackbr_icier = 0 ?
≠ 0 (NACK detected)
tend_icsr ← 0
Clear transmit end flag.
= 0 (ACK detected)
iic_index = 0 ?
≠ 0 (starting from second byte)
sclo_iccr2 = 1 ?
= 1 (SCL pin is held high.)
≠ 1 (SCL pin is held low.)
= 0 (first byte (slave address))
rie_icier ← 0
Disable receive data full interrupt request.
iccr1 ← iccr1 & 0xef
Set to receive mode.
teie_icier ← 1
Enable transmit end interrupt request.
ackbt_icier ← 1
In receive mode, transmit 1 as the acknowledge bit.
dummy_data ← icdrr
Dummy read
icdrt ← *iic_pointer
Set transmit data for next byte.
tdre_icsr ← 0
Clear transmit data empty flag.
iic_pointer++
Pointer to receive buffer + 1
teie_icier ← 0
Disable transmit end interrupt request.
rie_icier ← 1
Enable receive data full interrupt request.
iic_nack_det ← 1
Set NACK detection flag.
iic_index++
Number of transmit/receive bytes + 1
return

4.11 Slave Control Complete Processing



5. Sample Program

A sample program can be downloaded from the Renesas Electronics website.
To download, click “Application Notes” in the left-hand side menu of the R8C Family page.

6. Reference Documents

R8C/35C Group User’s Manual: Hardware Rev.1.00
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

Website and Support

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

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

Revision History	R8C/35C Group I^{2}C bus Single Master Control Program (Slave Transmit/Receive)

Rev.	Date	Description	
		Page	Summary
1.00	Aug. 31, 2010	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

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

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.

   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.

   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.

   — The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.

Notice

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

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

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

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

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

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

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

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

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

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

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

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

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

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

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

Colophon 1.0