1. Abstract

This document describes the slave transmit/receive processes in the I²C bus single master control program using the R8C/35C Group I²C 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.
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.

**Figure 3.3 Outline Flowchart**
A description of the process outline is as follows:

1. **Initial setting**
   Initialize the system clock, I^2^C bus interface associated SFRs, and variables used.

2. **I^2^C 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/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. **I^2^C 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/W) is 1.
   Disable the transmit data empty interrupt request and receive data full interrupt request. Enable the transmit end interrupt request.

4. **I^2^C 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. **I^2^C 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**

- **SCL** (master output)
- **SDA** (master output)
- **SCL** (slave output)
- **SDA** (slave output)
- **RDRF** bit in the ICSR register
- **STOP** bit in the ICSR register

Start condition

Stop condition

1. **(1) initial setting**
2. **(2) I^2^C bus interface interrupt (receive data full interrupt request)**
3. **(3) I^2^C bus interface interrupt (transmit data empty interrupt request)**
4. **(4) I^2^C bus interface interrupt (transmit end interrupt request)**
5. **(5) I^2^C bus interface interrupt (stop condition detection interrupt request)**
### Figure 3.5 Slave Transmit Timing (1)

Timing from initial setting to ACK reception

- **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. **Initial setting**
2. **Slave receive mode**
3. **Slave transmit mode**

### Figure 3.6 Slave Transmit Timing (2)

Timing from ACK reception to stop condition generation

- **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. **Initial setting**
2. **Slave receive mode**
3. **Slave transmit mode**
4. **Stop condition**

**Notes:**
- **D7, D6, D1, D0:** Data bits
- **R/W:** Read/Write indicator
- **ACK:** Acknowledgment
- **NACK:** No Acknowledgment
- **TRS:** Transmit mode
- **TRS:** Receive mode
- **Set to 0 by a program after reading the ICDRR register.
- **Set to 0 by a program.
- **Set the TRS bit to 0.**
3.1.1 Peripheral Functions

The I2C bus interface mode of the I2C bus interface is used under the following setting conditions:

- I2C 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>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3_5/SCL</td>
<td>I2C bus clock I/O pin</td>
</tr>
<tr>
<td>P3_7/SDA</td>
<td>I2C bus data I/O pin</td>
</tr>
</tbody>
</table>

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 I2C 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

<table>
<thead>
<tr>
<th>Memory</th>
<th>Size</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>622 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>RAM</td>
<td>4 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>Maximum user stack</td>
<td>21 bytes</td>
<td></td>
</tr>
<tr>
<td>Maximum interrupt stack</td>
<td>24 bytes</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_tx[BUFSIZE]</td>
<td>255 bytes</td>
<td>Transmit buffer</td>
</tr>
<tr>
<td>unsigned char iic_rx[BUFSIZE]</td>
<td>255 bytes</td>
<td>Receive buffer</td>
</tr>
<tr>
<td>unsigned char rcv_data[BUFSIZE]</td>
<td>255 bytes</td>
<td>Store received data</td>
</tr>
</tbody>
</table>

Table 4.2 Definition File Name: iic.c

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size /Bit-number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte_dt iic_str</td>
<td>—</td>
<td>Structure to store status</td>
</tr>
<tr>
<td>iic_status</td>
<td>1 byte</td>
<td>All statuses</td>
</tr>
<tr>
<td>iic_rw</td>
<td>b0</td>
<td>R/W flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Write (W)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Read (R)</td>
</tr>
<tr>
<td>iic_buf_full</td>
<td>b1</td>
<td>Buffer full flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Less than buffer size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Buffer full</td>
</tr>
<tr>
<td>iic_end</td>
<td>b2</td>
<td>Communication end flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Busy (mid-communication)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Ready (not mid-communication)</td>
</tr>
<tr>
<td>iic_nack_det</td>
<td>b3</td>
<td>NACK detection flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: No NACK detection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: NACK detection</td>
</tr>
<tr>
<td></td>
<td>—</td>
<td>b7 to b4 Not used (undefined)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_tx[BUFSIZE]</td>
<td>Transmit buffer</td>
</tr>
<tr>
<td>unsigned char iic_rx[BUFSIZE]</td>
<td>Receive buffer</td>
</tr>
<tr>
<td>unsigned char rcv_data[BUFSIZE]</td>
<td>Store received data</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Function
After setting the system clock, enable the I^2^C 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 I^2^C module.

### Declaration
```
void mcu_init(void)
```

### Outline
System clock setting

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### 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 I^2^C bus interface

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char ini</td>
<td>0: I^2^C module disabled</td>
</tr>
<tr>
<td></td>
<td>1: I^2^C module enabled</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Structure member) iic_status</td>
<td>All statuses</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Function
This function is called from the main processing. Initialize SFRs to use the I^2^C bus interface. When the I^2^C 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

```c
void _ssuic(void)
```

### Outline

**I2C bus interface interrupt handling**

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>unsigned char *iic_pointer</td>
<td>Transmit/receive buffer pointer</td>
</tr>
<tr>
<td>(Structure member) iic_status</td>
<td>All statuses</td>
</tr>
<tr>
<td>(Structure member) iic_rw</td>
<td>R/W flag</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### 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/W flag.
* Call the slave_trn_int function at slave transmit and the slave_rcv_int function at slave receive.

---

### Declaration

```c
unsigned char* iic_get_address(unsigned char rw)
```

### Outline

**Obtain buffer address processing**

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char rw</td>
<td>R/W flag</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char*</td>
<td>iic_rx</td>
<td>Receive buffer address</td>
</tr>
<tr>
<td></td>
<td>iic_tx</td>
<td>Transmit buffer address</td>
</tr>
</tbody>
</table>

### Function

This function is called from the I2C bus interface interrupt handling. Determine the R/W flag and return the buffer address.
### Declaration

| static void stp_int(void) |

### Outline

Stop condition detection processing

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Structure member) iic_end</td>
<td>Communication end flag</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Function

This function is called from the I²C bus interface interrupt handling. Reset the I²C 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

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>unsigned char *iic_pointer</td>
<td>Transmit/receive buffer pointer</td>
</tr>
<tr>
<td>(Structure member) iic_buf_full</td>
<td>Buffer full flag</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

### Function

This function is called from the I²C 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.
# I2C bus Single Master Control Program
## (Slave Transmit/Receive)

<table>
<thead>
<tr>
<th>Declaration</th>
<th>static void slave_trn_int(void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Slave transmit processing</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name Meaning</td>
</tr>
<tr>
<td></td>
<td>None —</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name Contents</td>
</tr>
<tr>
<td></td>
<td>unsigned char iic_index Number of transmit/receive bytes</td>
</tr>
<tr>
<td></td>
<td>unsigned char far *iic_pointer Transmit/receive buffer pointer</td>
</tr>
<tr>
<td></td>
<td>(structure member) iic_buf_full Buffer full flag</td>
</tr>
<tr>
<td></td>
<td>(structure member) iic_nack_det NACK detection flag</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type Value Meaning</td>
</tr>
<tr>
<td></td>
<td>None —</td>
</tr>
<tr>
<td>Function</td>
<td>This function is called from the I²C bus interface interrupt handling.</td>
</tr>
<tr>
<td></td>
<td>• When the number of transmit bytes has not reached the buffer size and ACK is detected, set the transmit data for the next byte.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• 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.</td>
</tr>
<tr>
<td></td>
<td>• When the NACK detection error flag is 1 or the buffer full flag is 1, discard the received data.</td>
</tr>
</tbody>
</table>
### Declaration

```
unsigned short iic_slave_end(void)
```

### Outline

Slave control complete processing

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>—</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Structure member) iic_end</td>
<td>Communication end flag</td>
</tr>
<tr>
<td>(Structure member) iic_buf_full</td>
<td>Buffer full flag</td>
</tr>
<tr>
<td>(Structure member) iic_rw</td>
<td>R/W flag</td>
</tr>
<tr>
<td>unsigned char iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low-order byte</td>
<td>IIC_BUSY</td>
<td>Mid-communication</td>
</tr>
<tr>
<td></td>
<td>IIC_REND</td>
<td>Reception completed</td>
</tr>
<tr>
<td></td>
<td>IIC_TEND</td>
<td>Transmission completed</td>
</tr>
<tr>
<td></td>
<td>IIC_ERR</td>
<td>Overrun error detected</td>
</tr>
<tr>
<td>High-order byte</td>
<td>1 to 255</td>
<td>Number of transmit/receive bytes</td>
</tr>
</tbody>
</table>

### 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 I2C module. Otherwise return IIC_BUSY (mid-communication). When the communication end flag is 0 after disabling the I2C 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")  

System clock setting
mcu_init()

asm("FSET I")

loop
   i = 0; i< BUFSIZE; i++
   iic_tx[i] ← i+1
   iic_rx[i] ← 0x00
loop

Initial setting of I2C bus interface
iic_init()

Slave control complete
processing
iic_slave_end()

temp.all ← Returned value

temp.byte.byte0

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
Wait until oscillation stabilizes.

i++

loop

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 I2C bus Interface

```plaintext
ini = 1 ?
≠ 1 (I2C module disabled)

mstiic ← 0
iicsel ← 1
icier ← 0x00
stop_icsr ← 0
ice_iccr1 ← 1
iccr1 ← 0x88
iccr2 ← 0xf0
icmr ← 0x00
pinsr ← pinsr & 0x09
sar ← DEVICE_ADDRESS<<1
icsr ← icsr & 0x15
rie_icier ← 1
tie_icier ← 1
icic ← 0x01
ic_status ← 0x00
asm("FSET I")
return
```

Set PD3_7 (SDA) and PD3_5 (SCL) to input mode.
Disable I2C bus interface interrupt.
Set I2C bus to active.
Select I2C bus interface function.
Disable stop condition detection interrupt request.
Disable receive data full interrupt request.
Disable transmit end interrupt request.
Disable transmit data empty interrupt request.
Clear stop condition detection flag.
This module is enabled for transfer operations.
Set to slave receive mode.
Set 20 Tcyc for setup time.
Continue next receive operation.
Initialize I2C bus control register 2.
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.
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.
Enable receive data full interrupt request.
Enable transmit data empty interrupt request.
Enable I2C bus interface interrupt.
Clear all status flags.
Enable interrupts.
4.6 I2C bus Interface Interrupt Handling

ssuic()

Yes
(stop condition detected)

No
(stop condition not detected)

aas_icsr = 1 ?

≠ 1 (no slave address match or starting from second byte)

= 1 (slave address match)

Clear slave address recognition flag.
Initialize number of transmit/receive bytes.
Clear all status flags.
Disable transmit data empty interrupt request.
Clear stop condition detection flag.

asm("nop")

Enable stop condition detection interrupt request.
Obtain buffer address (read ICDRR register).

Buffer address obtain processing
iic_get_address()

iic_pointer ← Returned value

iic rw ← trs iccr1

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

iic_get_address()

(rw & 0x01) = 0x00 ?

≠ 0x00 (slave transmit)

= 0x00 (slave receive)

return(iic_rx)

return(iic_tx)
### 4.8 Stop Condition Detection Processing

1. `stp_int()`: Clear stop condition detection flag.
2. `stop_icr ← 0`: Disable stop condition detection interrupt request.
3. `stie_icier ← 0`: Disable transmit end interrupt request.
4. `iccr1 ← iccr1 & 0xef`: Set to slave receive mode.
5. `ackbt_icier ← 0`: In receive mode, transmit 0 as the acknowledge bit.
6. `icr ← icr & 0x15`: Clear slave address recognition flag.
7. Clear receive data register full flag.
8. Clear transmit end flag.
10. `rie_icier ← 1`: Enable receive data full interrupt request.
11. `tie_icier ← 1`: Enable transmit data empty interrupt request.
12. `iic_end ← 1`: Set communication end flag.
13. `return`: End function.
4.9 Slave Receive Processing

slave_rcv_int()

\[\text{iic\_index} = 0 ?\]

- \(\neq 0\) (first byte (slave address))
  \[\text{iic\_index} \leftarrow 1\]
  Set number of transmit/receive bytes to 1.

- \(< \text{BUFSIZE}\) (less than buffer size)
  \[\text{iic\_index} < \text{BUFSIZE} ?\]
  \[\geq \text{BUFSIZE}\) (buffer size or greater)
  \[\text{iic\_buffer} \leftarrow 0 ?\]
  \[\neq 0\) (buffer full)
    \[\text{dummy\_data} \leftarrow \text{icdrr}\]
    Dummy read
  \[= 0\) (less than buffer size)
    \[\text{dummy\_data} \leftarrow \text{icdrr}\]
    Store read data from ICDRR register to receive buffer.

- \(\neq 0\) (starting from second byte)
  \[\text{iic\_index} < \text{BUFSIZE} ?\]
  \[\geq \text{BUFSIZE}\) (buffer size or greater)
  \[\text{iic\_index} \leftarrow 1\]
  \[\text{iic\_buffer} \leftarrow 1\]
  \[\text{iic\_pointer}++\]
  \[\text{iic\_index}++\]
  Number of transmit/receive bytes + 1

\[\text{return}\]
### 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 ?`
      - `ic_index < BUFSIZE (buffer size or greater)`
        - `iic_index < BUFSIZE (less than buffer size)`
          - `ackbr_icier = 0 ?`
            - `= 0 (NACK detected)`
              - `tend_icsr ← 0`
                - Clear transmit end flag.
              - `scl0_iccr2 = 1 ?`
                - `= 1 (SCL pin is held high.)`
                  - `ichr1 ← icr1 & 0xef`
                    - Set to receive mode.
                - `= 1 (SCL pin is held low.)`
                  - `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.
                - `tiee_icier ← 0`
                  - Disable transmit end interrupt request.
                - `rie_icier ← 1`
                  - Enable receive data full interrupt request.
                - `iic_buf_full ← 1`
                  - Set buffer full flag.
            - `≠ 0 (ACK detected)`
              - `tend_icsr ← 0`
                - Clear transmit end flag.
              - `scl0_iccr2 = 1 ?`
                - `= 1 (SCL pin is held high.)`
                  - `ichr1 ← icr1 & 0xef`
                    - Set to receive mode.
                - `= 1 (SCL pin is held low.)`
                  - `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.
                - `tiee_icier ← 0`
                  - Disable transmit end interrupt request.
                - `rie_icier ← 1`
                  - Enable receive data full interrupt request.
                - `iic_buf_full ← 1`
                  - Set buffer full flag.
          - `icr1 ← icr1 & 0xef`
            - In receive mode, transmit 1 as the acknowledge bit.
          - `dummy_data ← icdrr`
            - Dummy read
          - `tdre_icsr ← 0`
            - Clear transmit data empty flag.
          - `tiee_icier ← 0`
            - Disable transmit end interrupt request.
          - `rie_icier ← 1`
            - Enable receive data full interrupt request.
          - `iic_buf_full ← 1`
            - Set buffer full flag.
    - `ic_index = 0 ?`
      - `= 0 (first byte (slave address))`
        - `rie_icier ← 0`
          - Disable receive data full interrupt request.
        - `tiee_icier ← 1`
          - Enable transmit end interrupt request.
      - `≠ 0 (starting from second byte)`
        - `icdrt ← *iic_pointer`
          - Set transmit data for next byte.
        - `iic_pointer++`  
          - Pointer to receive buffer + 1
        - `iic_index++`  
          - Number of transmit/receive bytes + 1
      - `- 1 (SCL pin is held high.)`
        - `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.
        - `tiee_icier ← 0`
          - Disable transmit end interrupt request.
        - `rie_icier ← 1`
          - Enable receive data full interrupt request.
        - `iic_buf_full ← 1`
          - Set NACK detection flag.
    - `- 1 (SCL pin is held low.)`
      - `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.
      - `tiee_icier ← 0`
        - Disable transmit end interrupt request.
      - `rie_icier ← 1`
        - Enable receive data full interrupt request.
      - `iic_buf_full ← 1`
        - Set NACK detection flag.
  - `iic_index = 0 ?`
    - `= 0 (first byte (slave address))`
      - `rie_icier ← 0`
        - Disable receive data full interrupt request.
      - `tiee_icier ← 1`
        - Enable transmit end interrupt request.
    - `≠ 0 (starting from second byte)`
      - `icdrt ← *iic_pointer`
        - Set transmit data for next byte.
      - `iic_pointer++`
        - Pointer to receive buffer + 1
    - `iic_index++`  
      - Number of transmit/receive bytes + 1
  - `return`
4.11 Slave Control Complete Processing

```
return(temp.all)
```

```
(iic_end = 1) && (iic_index > 1) ?

Yes
(transmit/receive data but not mid-communication)
```

```
iic_slave_end()
```

```
Initial setting of I2C bus interface

iic_init()
```

```
iic_end = 0 ?

= 0 (mid-communication)
```

```
(iic_buf_full = 1 ?

= 1 (buffer full)
```

```
temp.byte.byte1 ← iic_index
```

```
temp.byte.byte1 ← iic_index - 1
```

```
Set number of transmit/receive bytes.
```

```
iic_rw = 0 ?

= 0 (slave receive)
```

```
temp.byte.byte0 ← IIC_REND
```

```
Set status (reception completed).
```

```
iic_rw ≠ 0 (slave transmit)

= 0 (mid-communication or no transmit/receive data)
```

```
Yes
(transmit/receive data but not mid-communication)
```

```
iic_slave_end()
```

```
(iic_end = 1) && (iic_index > 1) ?

No
```

```
(iic_end = 1) && (iic_index > 1) ?

= 0 (mid-communication or no transmit/receive data)
```

```
= 0 (not mid-communication)
```

```
iic_rw = 0 ?

≠ 0 (slave transmit)
```

```
= 0 (slave receive)
```

```
temp.byte.byte0 ← IIC_BUSY
```

```
Set status (mid-communication).
```

```
temp.byte.byte0 ← IIC_ERR
```

```
Set status (overrun error).
```

```
temp.byte.byte0 ← IIC_REND
```

```
Set status (transmission completed).
```

```
temp.byte.byte0 ← IIC_TEND
```

```
Set status (transmission completed).
```

```
Initial setting of I2C bus interface

iic_init()
```

```
I2C module disabled
```

```
temp.byte.byte0 ← IIC_REND
```

```
Set status (reception completed).
```

```
temp.byte.byte0 ← IIC_TEND
```

```
Set status (transmission completed).
```

```
return(temp.all)
```
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
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Aug. 31, 2010</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>

All trademarks and registered trademarks are the property of their respective owners.
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 example. 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 product 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 manufacturers, 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 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.

8. You should use the Renesas Electronics products described in this document in 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 malfunction under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard 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, the 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 damage 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.

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

(Notice 2) "Renesas Electronics products" means any product developed or manufactured by or for Renesas Electronics.