1. Abstract

This document describes the master transmit/receive processes in I²C bus interface single master communication using the R8C/35C Group serial interface (UART2) special mode 1 (I²C mode).

For details on UART2 special mode 1, refer to the M16C Family, R8C Family I²C bus Interface Using UARTi Special Mode 1 application note.

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

The simplified I²C bus communication is enabled by controlling additional functions for I²C bus communication added to the UARTi clock synchronous circuit for I²C bus interface using UARTi special mode 1. The I²C bus interface using UARTi special mode 1 has more limitations for software processing time and timing than the I²C bus interface hardware module. Careful verification and evaluation of your system are recommended, including the interaction between the I²C bus communication program and programs other than the I²C bus communication program.
3. Application Example

3.1 Program Outline

Transmission is performed in 3-byte data both in master transmission and reception. Master transmission and reception are repeated alternately. This transmission procedure conforms to the I²C bus communication protocol when used under the following conditions:

- Slave address: 7 bits
- Transfer rate: Approximately 350 kbps (1)
- Transfer data length: 1 to 255 bytes (not including the slave address)
- Single master communication (multimaster is not supported)
- Restart condition generation is not supported.

Note:

1. The setting value is 384 kbps.

When the clock synchronous function is enabled, there is a sampling delay of the noise filter width plus 1 to 1.5 cycles of the U2BRG count source. As there is also a delay of the SCL clock when high is determined, the SCL clock high width is extended. Therefore, the actual SCL clock becomes slower than SCL clock transfer rate setting.

In this application example, the actual transfer rate becomes approximately 350 kbps since the clock synchronous function is enabled (reference value: pull up voltage 5 V, pull up resistance 1 kΩ). Standard-mode and Fast-mode are 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.
Figure 3.3 Outline Flowchart

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.

1. Initial setting
   Initialize the system clock, UART2 associated SFRs, and variables used.
2. Start master control
   Enable the start/stop condition generation interrupt and generate a start condition.
3. Start/stop condition generation interrupt
   An interrupt request is generated when start condition generation is completed and a stop condition is detected. When start condition generation is completed, the UART2 transmit interrupt is enabled and slave address is transmitted. When a stop condition is detected, SFR values which changed during communication are returned to their initial values.
4. UART2 transmit interrupt
   A UART2 transmit interrupt is generated at the falling edge of the ninth bit of the SCL clock. When transmitting, set the next byte transmit data. When receiving, set ACK/NACK for the next byte. When communication is completed, generate a stop condition.
Figure 3.4 Master Transmit Timing
I²C bus Interface Using UART2 Special Mode 1
(Master Transmit/Receive)

**Figure 3.5** Master Reception Timing (1)

- **Start condition**
  - SCL (master output)
  - SDA (master output)
  - SCL (slave output)
  - SDA (slave output)

- **STAREQ bit in the U2SMR4 register**
  - STPREQ bit in the U2SMR4 register
  - STSPSEL bit in the U2SMR4 register
  - CKPH bit in the U2SMR3 register

- **Program processing**
  - Initial setting
- **(2)** Start master control
- **(3)** Start condition generation interrupt
- **(4) UART2 transmit interrupt**

**Figure 3.6** Master Reception Timing (2)

- **Stop condition**
  - SCL (master output)
  - SDA (master output)
  - SCL (slave output)
  - SDA (slave output)

- **STAREQ bit in the U2SMR4 register**
  - STPREQ bit in the U2SMR4 register
  - STSPSEL bit in the U2SMR4 register
  - CKPH bit in the U2SMR3 register

- **Program processing**
  - UART2 transmit interrupt
- **(3)** Stop condition detection interrupt
- **(4) UART2 transmit interrupt**

**Timing from initial setting to ACK transmission**

- **SCL** (master output)
- **SDA** (master output)
- **SCL** (slave output)
- **SDA** (slave output)

**Timing from ACK transmission to stop condition generation**

- **SCL** (master output)
- **SDA** (master output)
- **SCL** (slave output)
- **SDA** (slave output)

- **STAREQ bit in the U2SMR4 register**
- **STPREQ bit in the U2SMR4 register**
- **STSPSEL bit in the U2SMR4 register**
- **CKPH bit in the U2SMR3 register**

- **Becomes 0 when a start condition is generated.**
- **Set to 0 or 1 by a program.**
- **Becomes 0 when a stop condition is generated.**
- **Set to 0 or 1 by a program.**
3.1.1 Peripheral Functions

Serial interface (UART2) special mode 1 (I2C mode) is used under the following setting conditions:

- I2C mode is used.
- Transfer clock is internal clock source.
- f1 is used as U2BRG count source.
- SDA2 and SCL2 pins are N-channel open-drain.
- Transfer format uses MSB first.
- Transmission completed (TXEPT is 1) is selected as the UART2 transmit interrupt source.
- Clock delay is used.
- Seven to eight cycles of U2BRG count source are selected as SDA2 digital delay value.
- Clock synchronization is enabled.
- SCL2 wait function is disabled.
- SDA2 output disable function is not used.
- Start/stop condition generation interrupt is used.
- UART2 transmit interrupt is used.
- UART2 receive interrupt is not used.
- Transfer rate is about 384 kbps.

Calculating the transfer rate

\[
\text{Transfer rate} = \frac{\text{U2BRG count source}}{2 \times (\text{U2BRG register setting value} + 1)}
\]

\[
= \frac{20 \text{ MHz (f1)}}{2 \times (25 + 1)}
\]

\[
\approx 384.615 \text{ kbps}
\]

Table 3.1 Pins Used and Their Function

<table>
<thead>
<tr>
<th>Pin</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3_4/SCL2</td>
<td>I/O</td>
<td>I2C mode clock I/O pin</td>
</tr>
<tr>
<td>P3_7/SDA2</td>
<td>I/O</td>
<td>I2C mode 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 XIN clock (20 MHz), change the setting value of the U2BRG count source and the U2BRG register according to the transfer rate calculation shown in 3.1.1 Peripheral Functions.

3.2 Memory

Table 3.2 Memory

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>538 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>RAM</td>
<td>6 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>Maximum user stack</td>
<td>16 bytes</td>
<td></td>
</tr>
<tr>
<td>Maximum interrupt stack</td>
<td>30 bytes</td>
<td></td>
</tr>
</tbody>
</table>

Usage memory size varies depending on C compiler version and compile options. The above applies under the following conditions:

C compiler: M16C Series, R8C Family 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 Usage Variables

Definition file name: main.c

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_txBufSize</td>
<td>255 bytes</td>
<td>Transmit buffer</td>
</tr>
<tr>
<td>unsigned char iic_rxBufSize</td>
<td>255 bytes</td>
<td>Receive buffer</td>
</tr>
</tbody>
</table>

Definition file name: iic.c

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte_dt iic_str1</td>
<td>-</td>
<td>Structure to store slave address</td>
</tr>
<tr>
<td>iic_slave_addr</td>
<td>1 byte</td>
<td>Slave address</td>
</tr>
<tr>
<td>iic_rw b0</td>
<td>R/W flag</td>
<td>Master transmit</td>
</tr>
<tr>
<td></td>
<td>0: Write (W) Master transmit</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Read (R) Master receive</td>
<td></td>
</tr>
<tr>
<td>iic_err_par b1</td>
<td>Parameter error flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Parameter error</td>
<td></td>
</tr>
<tr>
<td>iic_err_nack b2</td>
<td>NACK detection error flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: NACK detection error</td>
<td></td>
</tr>
<tr>
<td>iic_err_addr b3</td>
<td>No address match error flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: No address match error</td>
<td></td>
</tr>
<tr>
<td>iic_status 1 byte</td>
<td>All statuses</td>
<td></td>
</tr>
<tr>
<td>iic_start b0</td>
<td>Communication completed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: Communication completed</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Mid-communication</td>
<td></td>
</tr>
<tr>
<td>iic_err_par b1</td>
<td>Parameter error flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: Parameter error</td>
<td></td>
</tr>
<tr>
<td>iic_err_nack b2</td>
<td>NACK detection error flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: NACK detection error</td>
<td></td>
</tr>
<tr>
<td>iic_err_addr b3</td>
<td>No address match error flag</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0: No error</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1: No address match error</td>
<td></td>
</tr>
<tr>
<td>iic_length 1 byte</td>
<td>Transfer data length</td>
<td></td>
</tr>
<tr>
<td>iic_index 1 byte</td>
<td>Number of transmit/receive bytes</td>
<td></td>
</tr>
<tr>
<td>iic_pointer 2 bytes</td>
<td>Transmit/receive buffer pointer</td>
<td></td>
</tr>
</tbody>
</table>

- b7 to b4 Not used (undefined)
### 4.2 Function Tables

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void main (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Main processing</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>unsigned char iic_tx[BUFSIZE]</td>
</tr>
<tr>
<td></td>
<td>unsigned char iic_rx[BUFSIZE]</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>After initializing the system clock and UART2, master transmission and reception are repeated alternately. Call the iic_master_start function to start master control and call the iic_master_end function to wait for completion of master control.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void mcu_init (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>System clock setting</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Call this function from the main processing. Set the system clock (XIN clock).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void uart2_init (unsigned char ini)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>UART2 initial setting</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>unsigned char ini</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Call this function from the main processing. Initialize SFR to use UART2 in special mode 1 (I2C mode).</td>
</tr>
</tbody>
</table>
### Declaration

```c
unsigned char iic_master_start (
    unsigned char addr,
    unsigned char rw,
    unsigned char *buf,
    unsigned char len)
```

### Outline

Master control start processing

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char addr</td>
<td>0x00 to 0x7F: Specify slave address</td>
</tr>
<tr>
<td>unsigned char rw</td>
<td>0x00: Master transmit 0x01: Master receive</td>
</tr>
<tr>
<td>unsigned char *buf</td>
<td>Transmit or receive buffer pointer</td>
</tr>
<tr>
<td>unsigned char len</td>
<td>0x01 to 0xFF: Transfer data length</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>
<tr>
<td>(structure member) iic_start</td>
<td>Mid-communication flag</td>
</tr>
<tr>
<td>(structure member) iic_err_par</td>
<td>Parameter error flag</td>
</tr>
<tr>
<td>(structure member) iic_slave_addr</td>
<td>Slave address</td>
</tr>
<tr>
<td>unsigned char iic_length</td>
<td>Transfer data length</td>
</tr>
<tr>
<td>unsigned char *iic_pointer</td>
<td>Transmit/receive buffer pointer</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>0</td>
<td>Bus busy</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Bus free</td>
</tr>
<tr>
<td></td>
<td>0xFF</td>
<td>Parameter error</td>
</tr>
</tbody>
</table>

### Function

This function is called by the main function to perform master control start processing. Before executing this function, execute the uart2_init function to enable I2C mode. In the function header, all statuses are initialized and argument parameters are checked. If any parameter value is invalid, the parameter error flag is set to 1 and 0xFF is returned. Master control start processing is not performed when a parameter error is detected. Next, the bus status is checked.

- When the bus is busy, the returned value is 0 and the master control start processing is not performed.
- When the bus is free, the returned value is 1 and master control start processing is performed. Set the mid-communication flag to 1 and a start condition is generated.

### Declaration

```c
void _uart2_bcnic (void)
```

### Outline

Start/stop condition generation 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>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

An interrupt is generated when the start condition generation is completed and a stop condition is detected. The sta_int function is called when the start condition generation is completed. The stp_int function is called when a stop condition is detected.
### Declaration
```
static void sta_int (void)
```

### Outline
Start 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>iic_slave_addr</td>
<td>Slave address</td>
</tr>
<tr>
<td>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>None</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

#### Function
Called from the start/stop condition generation interrupt handling. UART2 transmit/receive interrupt is enabled. Transmit the slave 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>iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>iic_start</td>
<td>Mid-communication 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
Called from the start/stop condition generation interrupt handling. The SFRs changed during communication are reset, and the mid-communication flag is set to 0.

### Declaration
```
void _uart2_trance (void)
```

### Outline
UART2 transmit 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>iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>iic_err_addr</td>
<td>No address match error flag</td>
</tr>
<tr>
<td>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 falling edge of the ninth bit of the SCL clock. Read the U2RB register in the function header. When a NACK is detected during slave address transmission, set the no address match error flag to 1. At all other times, the master_trn_int function is called in master transmission and the master_rcv_int function is called in master reception. When communication is completed, generate a stop condition.
### Master Transmit Processing

#### Function

Called from the UART2 transmit interrupt handling.

**IIC_SP_OFF** is returned in the following case:
- ACK is detected and not the last byte (starts the next transmission).

**IIC_SP_ON** is returned in the following cases:
- NACK is detected (NACK detect error flag is set to 1).
- The last byte transmission is completed.

#### Declaration

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>IIC_SP_ON</td>
<td>0: Stop condition generated</td>
</tr>
<tr>
<td></td>
<td>IIC_SP_OFF</td>
<td>1: Stop condition not generated</td>
</tr>
</tbody>
</table>

#### Declaration

<table>
<thead>
<tr>
<th>static unsigned char master_trn_int (unsigned short rb_data)</th>
</tr>
</thead>
</table>

#### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short rb_data</td>
<td>Data read from the U2RB register</td>
</tr>
</tbody>
</table>

#### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(structure member) iic_err_nack</td>
<td>NACK detection error flag</td>
</tr>
<tr>
<td>unsigned char iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>unsigned char iic_length</td>
<td>Transfer data length</td>
</tr>
<tr>
<td>unsigned char *iic_pointer</td>
<td>Transmit/receive buffer pointer</td>
</tr>
</tbody>
</table>

#### Returned value

#### Master Receive Processing

#### Function

Called from UART2 transmit interrupt handling.

Argument value is stored in the receive buffer (except slave address data).

NACK is set to the transmit register when the following data is the last byte. ACK is set to the transmit register when the following data is a byte other than the last byte.

After setting ACK or NACK to the transmit register, the next transmit operation starts.

**IIC_SP_OFF** is returned in the following case:
- The following data is the last byte data.

**IIC_SP_ON** is returned in the following case:
- The following data is not the last byte data.

#### Declaration

<table>
<thead>
<tr>
<th>static unsigned char master_rcv_int (unsigned short rb_data)</th>
</tr>
</thead>
</table>

#### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short rb_data</td>
<td>Data read from the U2RB register</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_length</td>
<td>Transfer data length</td>
</tr>
<tr>
<td>unsigned char *iic_pointer</td>
<td>Transmit/receive buffer pointer</td>
</tr>
</tbody>
</table>

#### Returned value

#### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>IIC_SP_ON</td>
<td>0: Stop condition generated</td>
</tr>
<tr>
<td></td>
<td>IIC_SP_OFF</td>
<td>1: Stop condition not generated</td>
</tr>
</tbody>
</table>
### Declaration
```
unsigned char iic_master_end (void)
```

### Outline
Master control completed 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_status</td>
<td>All statuses</td>
</tr>
<tr>
<td>(structure member) iic_start</td>
<td>Mid-communication flag</td>
</tr>
<tr>
<td>(structure member) iic_err_par</td>
<td>Parameter error flag</td>
</tr>
<tr>
<td>(structure member) iic_err_nack</td>
<td>NACK detection error flag</td>
</tr>
<tr>
<td>(structure member) iic_err_addr</td>
<td>No address match error flag</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>0</td>
<td>Mid-communication</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Communication completed</td>
</tr>
</tbody>
</table>

### Function
Called from the main function. It informs the user of the master control state. During communication, this function returns 0. When communication is completed, this function returns 1. Additional processing after communication is completed can be added as needed.
4.3 Main Processing

```
main()
asm("fclr I")

System clock initial setting
mcu_init()

UART2 initial setting
uart2_init()
asm("set I")

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

mode ← WRITE
```

<table>
<thead>
<tr>
<th>R/W mode initial setting (master transmit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returned value is 0 ?</td>
</tr>
<tr>
<td>≠ WRITE (master receive)</td>
</tr>
<tr>
<td>= WRITE (master transmit)</td>
</tr>
<tr>
<td>Master control start processing</td>
</tr>
<tr>
<td>iic_master_start()</td>
</tr>
<tr>
<td>Returned value is 0 ?</td>
</tr>
<tr>
<td>≠ 0 (bus busy)</td>
</tr>
<tr>
<td>= 0 (bus free or parameter error)</td>
</tr>
<tr>
<td>mode ← READ</td>
</tr>
<tr>
<td>Master control completed processing</td>
</tr>
<tr>
<td>iic_master_end()</td>
</tr>
<tr>
<td>Returned value is 0 ?</td>
</tr>
<tr>
<td>= 0 (mid-communication)</td>
</tr>
<tr>
<td>≠ 0 (communication completed)</td>
</tr>
</tbody>
</table>

Disable interrupts.
System clock initial setting (XIN clock setting)
UART2 special mode 1 (I2C mode) enabled
Enable interrupts.
Set transmit data to transmit buffer. Clear receive buffer.
Change R/W mode

```
Returned value is 0 ?
≠ 0 (communication completed)
```
4.4 System Clock Setting

```
mcu_init()

prc0 ← 1  Disable protection.

cm14 ← 0  Start low-speed on-chip oscillator.

cm13 ← 1  Switch port/XIN-XOUT: XIN-XOUT pin

cm05 ← 0  Oscillate XIN clock.

loop
  i <= 2040
  i++
  loop

  cm07 ← 0  Select XIN clock.

  ocd2 ← 0  System clock: Select XIN clock.

  cm1 ← cm1 & 0x3F  Select CPU clock no division.

  cm06 ← 0  Enable bits CM16 and CM17.

  cm14 ← 0  Stop low-speed on-chip oscillator.

  prc0 ← 0  Enable protection.

  return
```

4.5 UART2 Initial Setting

```
uart2_init()

ini = 1 ?
≠ 1 (I²C mode disabled)
= 1 (I²C mode enabled)

  u2smr ← 0x01
  Select I²C mode.

  P_IIC ← P_IIC & P_IIC_INIT
  Set the initial value:
P3_7 (SDA2) = P3_4 (SCL2) = 1 (high)

  PD_IIC ← PD_IIC & PD_IIC_INIT
  Set the port direction:
PD3_7 (SDA2) = PD3_4 (SCL2) = 0 (input mode)

  u2sr0 ← 0x11
  Assign SDA2 pin to P3_7.

  u2sr1 ← 0x00
  Assign SCL2 pin to P3_4.

  u2bcnic ← 0x00
  Perform U2SR1 register initial setting.

  s2tic ← 0x00
  Disable start/stop condition generation interrupt.

  u2c1 ← 0x00
  Disable UART2 transmit interrupt.

  u2mr ← 0x0A
  Select I²C mode, select external clock.

  u2smr2 ← 0x03
  Select UART2 transmit interrupt, enable clock synchronization.

  u2smr3 ← 0xE0
  SDA2 digital delay: 7 or 8 cycles of U2BRG count source

  u2smr4 ← 0x70
  Enable NACK output (release SDA2 pin), enable SCL output stop.

  u2c0 ← 0xB0
  U2BRG count source: f1
  Data output: Set pins SDA2 and SCL2 as N-channel open-drain output.
  Transfer format: MSB first

  u2brg ← IIC_BRG
  Set 384 kbps transfer rate.

  u2mr ← 0x02
  Select internal clock.

  u2c1 ← 0x10
  Select transmission completed (TXEPT is 1)
as UART2 transmit interrupt source.

  u2bcnic ← 0x00
  Set IR bit to 0.

  s2tic ← 0x00
  Disable transmission/reception.

  u2c1 ← 0x10
  Disable transmission/reception.

  u2mr ← 0x00
  Disable serial interface.

PD_IIC ← PD_IIC & PD_IIC_INIT
Set the port direction:
PD3_7 (SDA2) = PD3_4 (SCL2) = 0 (input mode)

u2sr0 ← 0x00
Do not use pins SDA2 and SCL2.

u2sr1 ← 0x00
Perform U2SR1 register initial setting.

u2bcnic ← 0x00
Disable start/stop condition generation interrupt.

s2tic ← 0x00
Disable UART2 transmit interrupt.

u2c1 ← 0x10
Disable transmission/reception.

u2mr ← 0x00
Disable serial interface.
```

Perform U2SR1 register initial setting.

Set the initial value:
P3_7 (SDA2) = P3_4 (SCL2) = 1 (high)

Assign SDA2 pin to P3_7.
Assign SCL2 pin to P3_4.

Perform U2SR1 register initial setting.

Disable start/stop condition generation interrupt.

Disable UART2 transmit interrupt.

Select I²C mode, select external clock.

Select UART2 transmit interrupt, enable clock synchronization.

SDA2 digital delay: 7 or 8 cycles of U2BRG count source

Enable NACK output (release SDA2 pin), enable SCL output stop.

U2BRG count source: f1
Data output: Set pins SDA2 and SCL2 as N-channel open-drain output.
Transfer format: MSB first

Set 384 kbps transfer rate.

Select internal clock.

Select transmission completed (TXEPT is 1) as UART2 transmit interrupt source.

Set IR bit to 0.
4.6 Master Control Start Processing

```
iiic_master_start()

iiic_status ← 0x00
Clear all statuses.

(len = 0) || (rw > 1) || (addr > 0x7F) ?
Yes (parameter error)

iiic_slave_addr ← addr << 1
Set slave address.

iic_slave_addr ← iic_slave_addr + rw
Set R/W.

iic_length ← len
Set transfer data length.

iic_pointer ← buf
Set address of buffer.

iic_start ← 1
Set mid-communication flag to 1.

u2bcnic ← 0x01 Enable start/stop condition generation interrupt.

u2smr4 ← 0x09
Set STSPSEL bit to 0.

u2mr ← 0x02 Select I2C mode, select internal clock.

u2brg ← 0
Set U2BRG to 0 to set the shortest wait time.

u2smr2 ← 0x03
More than half an SCL clock cycle (50 ns) is needed to execute this instruction.

u2brg ← IIC_BRG
Return U2BRG setting to target transfer rate.

Generate start condition.

return(1)
```

Yes (parameter error)

```
iiic_err_par ← 1
Set parameter error flag to 1.

return(0xFF)
```

No (no parameter error)

```
bbs = 1 ?
= 1 (bus busy)

return(0)
```

1 (bus free)

```
iic_slave_addr ← addr << 1
Set slave address.

iic_slave_addr ← iic_slave_addr + rw
Set R/W.

iic_length ← len
Set transfer data length.

iic_pointer ← buf
Set address of buffer.

iic_start ← 1
Set mid-communication flag to 1.

u2bcnic ← 0x01 Enable start/stop condition generation interrupt.

u2smr4 ← 0x09
Set STSPSEL bit to 0.

u2mr ← 0x02 Select I2C mode, select internal clock.

u2brg ← 0
Set U2BRG to 0 to set the shortest wait time.

u2smr2 ← 0x03
More than half an SCL clock cycle (50 ns) is needed to execute this instruction.

u2brg ← IIC_BRG
Return U2BRG setting to target transfer rate.

Generate start condition.

return(1)
```
4.7 Start/Stop Condition Generation Interrupt Handling

```plaintext
_uart2_bcnic()

bbs = 1 ?

≠ 1 (stop condition detection)

= 1 (start condition detection)

Start condition detection processing
sta_int()

Stop condition detection processing
stp_int()

return
```

4.8 Start Condition Detection Processing

```plaintext
sta_int()

u2smr3 ← 0xE2

u2c1 ← 0x15

u2smr4 ← 0x00

u2rb ← 0x0000

temp.byte.byte0 ← iic_slave_addr

temp.byte.byte1 ← 0x01

u2tb ← temp.all

u2bcnic ← 0x01

s2tic ← 0x01

iic_index ← 0x00

return
```

Clock phase: With clock delay

Enable transmission/reception.

Do not generate start/stop conditions.

Initialize receive buffer register.

Set slave address.

Set data to release SDA2 pin at the ninth bit.

Start first byte (slave address) transmission.

Set IR bit to 0 after changing CKPH bit.

Enable UART2 transmit interrupt.

Initialize number of transmit/receive bytes.
4.9 Stop Condition Detection Processing

stp_int()

u2c1 ← 0x10

Disable transmission/reception.

P_IIC ← P_IIC | P_IIC_INIT

Set the initial value: P3_7 (SDA2) = P3_4 (SCL2) = 1 (high)

u2mr ← 0x00

Disable serial interface.

u2smr3 ← 0xE0

Set clock phase: No clock delay

u2smr4 ← 0x70

Enable NACK output (release SDA2 pin), enable SCL output stop.

u2mr ← 0x02

Select I²C mode, select internal clock.

s2tic ← 0x00

Disable UART2 transmit interrupt.

u2bcnic ← 0x00

Disable start/stop condition generation interrupt.

iic_index ← 0x00

Initialize number of transmit/receive bytes.

iic_start ← 0

Set mid-communication flag to 0.

return
4.10 UART2 Transmit Interrupt Handling

```
_uart2_trance()

temp.all ← u2rb

iic_index = 0x00 &&
temp.bit.b8 = 1 ?

Yes (slave address not match)

No (slave address match)

iic_rw = 0 ?

= 0 (master transmit)

≠ 0 (master receive)

Master receive processing

master_rcv_int()

stop_req ← Returned value

Master transmit processing

master_tm_int()

stop_req ← Returned value

stop_req = IIC_SP_ON ?

= IIC_SP_ON (stop condition generated)

≠ IIC_SP_ON (stop condition not generated)

u2smr4 ← 0x04

u2smr4 ← 0x3C

Generate stop condition.

return

iic_err_addr ← 1

stop_req ← IIC_SP_ON

Set address not matched error flag to 1.

Generate stop condition.

iic_index = 0x00

Read from receive buffer register.

Yes (slave address not match)

stop_req ← Returned value

= 0 (master transmit)

≠ 0 (master receive)
```

Stop condition:
- Stop condition generated: u2smr4 ← 0x04
- Stop condition not generated: u2smr4 ← 0x3C
4.11 Master Transmit Processing

master_trn_int()

temp.all ← rb_data  
Set data read from the receive buffer register

temp.bit.b8 ≠ 0 ?  
≠ 0 (NACK detection)

= 0 (ACK detection)

iic_index < iic_length ?  
>= iic_length (last byte)

< iic_length (not last byte)

iic_err_nack ← 1  
Set NACK detection error flag to 1.

temp.byte.byte0 ← *iic_pointer  
Set next byte transmit data.

temp.byte.byte1 ← 0x01  
Set data to release the SDA2 pin at the ninth bit.

u2tb ← temp.all  
Set transmit data to U2TB register (start transmission).

iic_pointer++  
Pointer to the transmit buffer + 1

iic_index++  
Number of transmit/receive bytes + 1

return(IIC_SP_OFF)  
Return(IIC_SP_OFF)
4.12 Master Receive Processing

- master_rcv_int()

(first byte (slave address))

= 0x00

iic_index = 0x00 ?

≠ 0x00 (from the second byte on)

*iic_pointer ← (unsigned char)rb_data

Store data read from the receive buffer register to the receive buffer

iic_pointer++

Pointer to the receive buffer + 1

iic_index >= iic_length ?

>= iic_length (last byte)

Number of transmit/receive bytes + 1

iic_index++

< iic_length (not last byte)

iic_index >= iic_length ?

>= iic_length (next received data is the last byte)

< iic_length (next received data is not the last byte)

u2tb ← 0x01FF

Set NACK for next byte (prepare for next receive)

u2tb ← 0x00FF

Set ACK for next byte (prepare for next receive)

return(IIC_SP_ON)

iic_index >= iic_length ?

>= iic_length (last byte)

iic_index++

< iic_length (not last byte)

return(IIC_SP_OFF)

Notes:
1. Additional processing of communication completed normally can be added as needed.
2. Additional processing of communication completed with error can be added as needed.

4.13 Master Control Completed Processing

- iic_master_end()

iic_start = 1 ?

≠ 1 (mid-communication)

≠ 1 (communication completed)

(iic_status & 0x0E) ≠ 0x00 (error)

= 0x00 (normal end)

See Note 1.

See Note 2.

return(1)
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

Application Note
M16C Family, R8C Family I²C Bus Interface Using UARTi Special Mode 1 (REJ05B1349)
The latest version can be downloaded from the Renesas Electronics website.

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.

C Compiler Manual
M16C Series, R8C Family C Compiler Package V.5.45
C Compiler User’s Manual Rev.1.00
The latest version 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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Sep. 01, 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 damages 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 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 liable in any way 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 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. You use Renesas Electronics products in compliance with all applicable laws and regulations that require 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.

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

(Notice 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.