1. Abstract

This document describes how to use the multi-master \( \text{I}^2\text{C}\)-bus interface (\( \text{I}^2\text{C}\)-bus interface) function.

2. Introduction

The application example described in this document applies to the following microcomputers (MCUs):

MCUs: R32C/116 Group, R32C/117 Group, R32C/118 Group, R32C/116A Group, R32C/117A Group, and R32C/118A Group

This program can be used with other R32C/100 Series MCUs which have the same special function registers (SFRs) as the above groups. Check the manuals for any modifications to functions. Careful evaluation is recommended before using the program described in this application note.
3. Overview

The I²C-bus interface is a serial communication circuit based on the I²C-bus data transmit/receive format, and is equipped with arbitration lost detection and clock synchronous functions.

3.1 General Call

A general call can be detected when the address data is all 0's. A general call means that the master transmits general call address 00h to all slaves.

3.2 Addressing Format

7-bit addressing format is supported. Only the 7 high-order bits of the I²C-bus slave address register (slave address) are compared with the address data.

3.3 I²C-bus Interface Related Pins

MSCL pin: Clock I/O pin of the I²C-bus interface.
MSDA pin: Data I/O pin of the I²C-bus interface.
3.4 Selectable Functions

The functions below can be selected when using the I²C-bus interface.

(1) Transmit/Receive modes

There are four transmit/receive modes available when performing data communication:

- Master-transmission mode: Start and stop conditions are generated (master mode). Address and control data are output to the MSDA in synchronization with the MSCL clock generated by the master device.
- Master-reception mode: Data from the transmitting device is received in synchronization with the MSCL clock generated by the master device.
- Slave-transmission mode: Start and stop conditions generated by the master device are received (slave mode). Control data is output in synchronization with the clock generated by the master device.
- Slave-reception mode: Data from the transmitting device is received in synchronization with the clock generated by the master device.

(2) SCL mode

SCL mode can be selected from Standard-mode (100 kHz or less) and Fast-mode (400 kHz or less).

(3) ACK clock

ACK clock can be selected from the following:

- ACK clock not available: No ACK clocks are generated after a data transfer.
- ACK clock available: The master generates an ACK clock each time 1 byte of data is transferred.

(4) Data format

Data format can be selected from the following:

- Addressing format: The received slave address and bits SAD6 to SAD0 in the I2CSAR register are compared. When an address match is found or when a general call is received, an interrupt request is generated and additional data is transmitted and received.
- Free data format: An interrupt request is generated and additional data is transmitted and received regardless of the received slave address.
4. Data Transmit/Receive Example

The data transmit/receive examples are described in this section. The conditions for the examples are below.

- Slave address: 7 bits
- Data: 8 bits
- ACK clock available
- Standard-mode, bit rate: 100 kbps (fIIC: 16 MHz; φIIC: 4 MHz)
  
  \[
  \frac{16 \text{ MHz (fIIC)}}{4 \text{ MHz (φIIC)}} = 4 \text{ MHz (φIIC)}
  \]

\[
\frac{4 \text{ MHz (φIIC)}}{8} \text{ and further divided-by-5} = 100 \text{ kbps (bit rate)}.
\]

- In receive mode, ACK is returned for data other than the last data. NACK is returned after the last data is received.
- When receiving data, I\(^2\)C-bus interrupt at the eighth clock (before the ACK clock): disabled
- Stop condition detection interrupt: enabled
- Timeout detection interrupt: disabled
- Set own slave address to the I2CSAR register.

While receiving data, if an interrupt is enabled at the eighth clock, ACK or NACK can be set after each byte of received data is checked.

4.1 Initial Settings

Follow the initial setting procedure below for 4.2 Master Transmission to 4.5 Slave Transmission.

1. Write own slave address to bits SAD6 to SAD0 in the I2CSAR register.
2. Write 85h to the I2CCR register (CCR value: 5, standard-mode selected, ACK clock generated).
3. Write 01h to the I2CCR2 register (φIIC: set bits ICK1 and ICK0 in the I2CCR1 register, timeout detector is disabled by the interrupt event).
4. Write 03h to the I2CCR1 register (φIIC: fIIC divided-by-2, successful receive interrupt is enabled and a stop condition detection interrupt is enabled).
5. Write 0Fh to the I2CSR register (transmit/receive mode: slave-receive mode).
6. Write 98h to the I2CSSCR register (SSC value: 18h; start condition/stop condition generating mode selection: long mode).
7. Write 08h to the I2CCR0 register (number of transmit/receive bits: 8, I\(^2\)C-bus interface enabled, addressing format).

If the MCU uses a single-master system and the MCU itself is the master, start the initial setting procedures from step (2).
4.2 Master Transmission

Master transmission is described in this section. Programs (A) to (C) below refer to (A) to (C) in the following figure.

(A) Slave address transmission
   1. The BBSY bit in the I2CSR register must be 0 (bus free).
   2. Write E0h to the I2CSR register (start condition standby).
   3. Write a slave address to the upper 7 bits and set the least significant bit (LSB) to 0 (start condition generated, then slave address transmitted).

After a stop condition is generated and the BBSY bit becomes 0, the I2CSR register is write disabled for 1.5 cycles of φIIC. Therefore, when writing E0h to the I2CSR register and a slave address to the I2CTRSR register, a start condition is not generated.

When generating a start condition immediately after the BBSY bit in the I2CSR register changes from 1 to 0, confirm that both the TRX and MST bits are 1 (transmission mode and master mode) after step (1), and then execute step (2).

(B) Data transmission (in the I2C-bus interrupt routine)
   1. Write transmit data to the I2CTRSR register (data transmission).

(C) Completion of master transmission (in the I2C-bus interrupt routine)
   1. Write C0h to the I2CSR register (stop condition standby).
   2. Write dummy data to the I2CTRSR register (stop condition generated).

When the transmission is completed or ACK is not returned from the slave device (NACK returned), master transmission should be completed as shown in the example above.
4.3 Master Reception

Master reception is described in this section. Programs (A) to (D) below refer to (A) to (D) in the following figure.

(A) Slave address transmission
   (1) The BBSY bit in the I2CSR register must be 0 (bus free).
   (2) Write E0h to the I2CSR register (start condition standby).
   (3) Write a slave address to the upper 7 bits and set the least significant bit (LSB) to 1 (start condition generated, then slave address transmitted).

(B) Data reception 1 (after slave address transmission) (in the I2C-bus interrupt routine)
   (1) Write AFh to the I2CSR register (master receive mode).
   (2) Set the ACKD bit in the I2CCCR register to 0 (ACK sent) because the data is not the last one.
   (3) Write dummy data to the I2CTRSR register.

(C) Data reception 2 (data reception) (in the I2C-bus interrupt routine)
   (1) Read the received data from the I2CTRSR register.
   (2) Set the ACKD bit in the I2CCCR register to 1 (NACK sent) because the data is the last one.
   (3) Write dummy data to the I2CTRSR register.

(D) End of master reception (in the I2C-bus interrupt routine).
   (1) Read the received data from the I2CTRSR register.
   (2) Write C0h to the I2CSR register (stop condition standby state).
   (3) Write dummy data to the I2CTRSR register (stop condition generated).
4.4 Slave Reception

Slave reception is described in this section. Programs (A) to (C) below refer to (A) to (C) in the following diagram.

(A) Start of slave reception (in the I2C-bus interrupt routine)
   1) Check the content of I2CSR register. When the TRS bit is 0, the I2C-bus interface is in slave receive mode.
   2) Write dummy data to the I2CTRSR register.

(B) Data reception 1 (in the I2C-bus interrupt routine)
   1) Read the received data from the I2CTRSR register.
   2) Set the ACKD bit in the I2CCCR register to 0 (ACK sent) because the data is not the last one.
   3) Write dummy data to the I2CTRSR register.

(C) Data reception 2 (in the I2C-bus interrupt routine)
   1) Read the received data from the I2CTRSR register.
   2) Set the ACKD bit in the I2CCCR register to 1 (NACK sent) because the data is the last one.
   3) Write dummy data to the I2CTRSR register.
4.5 Slave Transmission

Slave transmission is described in this section. Programs (A) and (B) below refer to (A) and (B) in the following diagram.

When arbitration lost is detected, the TRS bit becomes 0 (receive mode) even when the bit after the slave address is 1 (read). Therefore, after arbitration lost is detected, read the I2CTRSR register. When bit 0 in the I2CTRSR register is 1, write 4Fh (slave transmit mode) to the I2CSR register and execute slave transmission.

![Diagram of Slave Transmission](image)

**Figure 4.4 Example of Slave Transmission**

**(A) Start of slave transmission (in the I2C-bus interrupt routine)**

1. Read the I2CSR register. When the TRS bit is 1, the I2C-bus interface is in slave transmit mode.
2. Write transmit data to the I2CTRSR register.

**(B) Data transmission (in the I2C-bus interrupt routine)**

1. Write transmit data to the I2CTRSR register.

Write dummy data to the I2CTRSR register even if an interrupt occurs at an ACK clock of the last transmit data. When the I2CTRSR register is written, the MSCL pin becomes high-impedance.
5. **Arbitration Lost**

The following describes the operation of the I\(^2\)C-bus interface when arbitration lost occurs. Figure 5.1 shows the Operation Timing of the Arbitration Lost Detect Flag.

![Figure 5.1 Operation Timing of the Arbitration Lost Detect Flag](image)

When arbitration lost occurs, the arbitration lost detect flag becomes 1.

1. Arbitration lost occurs while transmitting a slave address.
   When arbitration lost is detected, the communication mode automatically changes to slave reception, enabling the slave address to be received.
   If the selected data format is the addressing format, the slave address can be resolved by reading the AAS bit in the I2CSR register.

2. Arbitration lost occurs while transmitting data following the slave address.
   When arbitration lost is detected, the communication mode automatically changes to slave reception, enabling the data to be received.
6. Interrupts

The I2C-bus interface has the interrupt source listed below.

1) Interrupt when 9-bit transmission/reception is completed (including ACK/NACK)
   The interrupt source can be determined by reading the RIE bit in the I2CCR1 register. When the RIE bit is 0, it is
determined that the generated interrupt is attributable to this interrupt source.

2) Interrupt when 8 bits are received
   Setting the RIE bit to 1 enables this interrupt source.
   The interrupt source can be determined by reading the RIE bit. When the RIE bit is 1, it is determined that the
   generated interrupt is attributable to this interrupt source.
   If no determination is made of ACK/NACK transmissions, there is no need to use this interrupt.

3) Interrupt when a stop condition is detected
   Setting the STIE bit in the I2CCR1 register to 1 enables this interrupt source.
   The interrupt source can be determined by reading the STOP bit in the I2CCR2 register. When a stop condition is
detected, the STOP bit becomes 1.

4) Interrupt when the SCL clock is held high for a specified period of time during communication.
   Setting the TOE bit in the I2CCR2 register to 1 enables this interrupt source.
   The interrupt source can be determined by reading the TOF bit in the I2CCR2 register. When the MSCL clock
   remains high for more than a predetermined time during communication, the TOF bit becomes 1.

Figure 6.1 shows the I2C-bus Interface Interrupt Request Generation Timing.
Figure 6.1  \( \text{i}^2\text{C}-\text{bus Interface Interrupt Request Generation Timing} \)

1. \( \text{i}^2\text{C}-\text{bus Interface interrupt source attributed to setting bits RIE and STIE in the I2CCR1 register.} \)

2. \( \text{i}^2\text{C}-\text{bus Interface interrupt attributed to timeout detection.} \)

The above figure assumes the following:
1. The TRS bit in the I2CSR register is 0 (receive mode).
2. The ACKCLK bit in the I2CCCR register is 1 (ACK clock generated).
7. Notes on I²C-bus Interface

7.1 Generating a Start Condition

After a stop condition is generated and the BBSY bit becomes 0 (bus free), the I2CSR register is write disabled for 1.5 cycles of φIIC. Therefore, when writing E0h to the I2CSR register and a slave address to the I2CTRSR register, a start condition is not generated.

When generating a start condition immediately after the BBSY bit changes from 1 to 0, confirm that bits TRS and MST are both 1 after writing E0h in the I2CSR register, and then write a slave address to the I2CTRSR register.

8. Sample Program

8.1 Connection Example

Figure 8.1 shows a Connection Example.

![Connection Example Diagram]

(See Note 1)

Figure 8.1 Connection Example

8.2 Operation Conditions

Table 8.1 lists the Sample Program Operation Conditions.

<table>
<thead>
<tr>
<th>Item</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>I²C-bus interface clock (fIIC)</td>
<td>16 MHz (Xin: 16 MHz)</td>
</tr>
<tr>
<td>I²C-bus system clock (φIIC)</td>
<td>4 MHz (fIIC divided-by-5)</td>
</tr>
<tr>
<td>Bit rate</td>
<td>100 kbps (φIIC divided-by-8 and further divided-by-5)</td>
</tr>
<tr>
<td>SCL mode</td>
<td>Standard-mode</td>
</tr>
<tr>
<td>Data format</td>
<td>Addressing format</td>
</tr>
<tr>
<td>Slave address compare</td>
<td>Only I2CSAR register enabled</td>
</tr>
<tr>
<td>Data receive interrupt</td>
<td>Enabled</td>
</tr>
<tr>
<td>Stop condition detect interrupt</td>
<td>Enabled</td>
</tr>
<tr>
<td>Timeout detector</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
8.3 Setting Example

Four transmit/receive modes can be used in the sample program: master transmission, master reception, slave reception, and slave transmission. When calling the mode_ini function, the transmit/receive modes can be selected by setting arguments.

Set the other slave address and own slave address in the define declaration area in the sample program.

Figure 8.2 shows a Setting Example of Master Transmission. Figure 8.3 shows a Setting Example of Slave Address (0x09) and Own Slave Address (0x10).

---

```c
/***func comment**************************************************************/
/* Main Program
/**func comment end********************************************************/
void main(void){
  // Omitted
  /*==================================================*/
  /*=   Modify start
  /*==================================================*/
  mode_ini(MASTER, SND); /* First argument */
  /* MASTER : master */
  /* SLAVE : slave */
  /* Second argument */
  /* SND : transfer */
  /* REV : receive */
  /*==================================================*/
  /*=   Modify end
  /*==================================================*/
  /* Set the master (MASTER)/slave (SLAVE) as the first argument and send (SND)/receive (REV) as the second argument. */

Figure 8.2 Transmit/Receive Mode Setting Example
```

---

```c
/***func comment**************************************************************/
/* DEFINE
/**func comment end********************************************************/
/*==================================================*/
/*=   Modify start
/*==================================================*/
#define SLAVE_ADD 0x09 /* Other slave address(7bit) */
#define SELF_ADD 0x10 /* My slave address(7bit) */
/*==================================================*/
/*=   Modify end
/*==================================================*/

Figure 8.3 Slave Address Setting Example
```
8.4 Operation Example

8.4.1 Master Transmission and Slave Reception

Figure 8.4 shows a Master Transmission and Slave Reception Operation Example.

Figure 8.4 Master Transmission and Slave Reception Operation Example

1. Master: A start condition is generated after writing E0h to the I2CSR register and transmit data to the I2CTRSR register.
2. Master: The slave address set to bits b7 to b1 in the I2CTRSR register and the Write("0") set to b0 are output.
   Slave: ACK is output when a match is found between the received slave address and the value in the I2CSAR register.
3. Master: After ACK reception, the IR bit in the I2CIC register becomes 1.
   Slave: After ACK transmission, the IR bit becomes 1.
4. Slave: After data reception, the IR bit becomes 1. During interrupt handling, set the ACKD bit to 0 and ACK is output.
5. Master: After ACK reception, the IR bit becomes 1.
   Slave: After ACK transmission, the IR bit becomes 1.
6. Slave: After receiving 5 bytes, set the ACKD bit to 1 and NACK is output.
7. Master: A stop condition is generated.
   Slave: When detecting the stop condition, the IR bit becomes 1.
8.4.2 Master Reception and Slave Transmission

Figure 8.5 shows a Master Reception and Slave Transmission Operation Example.

(1) Master: A start condition is generated after writing E0h to the I2CSR register and transmit data to the I2CTSR register.

(2) Master: The slave address set to bits b7 to b1 in the I2CTSR register and Read("1") set to b0 are output.
   Slave: ACK is output when a match is found between the received slave address and the I2CSAR register value.
   The TRS bit in the I2CSR register becomes 1 (transmit mode) (only when the ALS bit in the I2CCR0 register is 1 (addressing mode)).

(3) Master: After ACK reception, the IR bit in the I2CIC register becomes 1.
    Slave: After ACK transmission, the IR bit becomes 1.

(4) Slave: After data reception, the IR bit becomes 1. During interrupt handling, set the ACKD bit to 0 and ACK is output.

(5) Master: After ACK transmission, the IR bit becomes 1.
    Slave: After ACK reception, the IR bit becomes 1.

(6) Slave: After receiving 5 bytes, set the ACKD bit to 1 and NACK is output.

(7) Slave: After receiving NACK, the TRS bit becomes 0 (receive mode) only when the AAS bit is 1.

(8) Master: A stop condition is generated.
    Slave: When detecting the stop condition, the IR bit becomes 1.
### 8.5 Function Tables

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void iic_ini(unsigned char ini, unsigned char sub_address)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>I²C-bus initialization function</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>ini</td>
</tr>
<tr>
<td></td>
<td>sub_address</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>iic_mode</td>
</tr>
<tr>
<td></td>
<td>iic_index</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Argument ini = When ENABLED (I²C-bus function enabled), initialize the I²C-bus before enabling interrupts. Argument ini = When DISABLED (I²C-bus function disabled), disable the I²C-bus interface and the I²C-bus interrupt.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void mode_ini(unsigned char ms, unsigned char sr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Function for setting respective transmit/receive modes</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>ms</td>
</tr>
<tr>
<td></td>
<td>sr</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>iic_ram[]</td>
</tr>
<tr>
<td></td>
<td>iic_length</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Set the respective transmit/receive modes.</td>
</tr>
</tbody>
</table>
### Multi-Master I2C-bus Interface

<table>
<thead>
<tr>
<th>Declaration</th>
<th>unsigned char iic_master_start(unsigned char slave, unsigned char sr, unsigned char *buf, unsigned char len)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Outline</strong></td>
<td>Master start function</td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td></td>
</tr>
<tr>
<td>Argument name</td>
<td>Meaning</td>
</tr>
<tr>
<td>slave</td>
<td>Specified slave address (0x00 to 0x7f)</td>
</tr>
<tr>
<td>sr</td>
<td>Select transmission or reception</td>
</tr>
<tr>
<td>_snd</td>
<td>SAND: Transmission mode</td>
</tr>
<tr>
<td>_rev</td>
<td>REV: Reception mode</td>
</tr>
<tr>
<td>*buf</td>
<td>Pointer for transmit buffer</td>
</tr>
<tr>
<td>len</td>
<td>Transmit/receive data size (0x00 to 0xff)</td>
</tr>
<tr>
<td><strong>Variable (global)</strong></td>
<td></td>
</tr>
<tr>
<td>Variable name</td>
<td>Content</td>
</tr>
<tr>
<td>iic_slave</td>
<td>Variable for storing slave address</td>
</tr>
<tr>
<td>iic_length</td>
<td>For transmit and receive size</td>
</tr>
<tr>
<td>iic_pointer</td>
<td>Pointer for transmission buffer</td>
</tr>
<tr>
<td>iic_mode</td>
<td>For selecting transmit/receive modes</td>
</tr>
<tr>
<td>iic_rw</td>
<td>READ/WRITE</td>
</tr>
<tr>
<td><strong>Returned value</strong></td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>Meaning</td>
</tr>
<tr>
<td>unsigned char</td>
<td>Master start failure/start successful</td>
</tr>
<tr>
<td>Function</td>
<td>Transmit the start condition and slave address after master setting.</td>
</tr>
<tr>
<td>Returned value</td>
<td></td>
</tr>
<tr>
<td>Declaration</td>
<td>void master_transfer(void)</td>
</tr>
<tr>
<td>Outline</td>
<td>Master transmit function</td>
</tr>
<tr>
<td>Argument</td>
<td>None</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>iic_mode</td>
</tr>
<tr>
<td></td>
<td>iic_length</td>
</tr>
<tr>
<td></td>
<td>iic_pointer</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Detect arbitration lost, confirming ACK/NACK reception, and transmitting data.</td>
</tr>
</tbody>
</table>

| Declaration | void master_receive(void) |
| Outline     | Master receive function |
| Argument    | None |
| Variable (global) | Variable name | Content |
|             | iic_mode | For selecting transmit/receive modes |
|             | iic_length | For transmit and receive size |
|             | iic_pointer | Receive buffer pointer |
| Returned value | None |
| Function    | Detecting arbitration lost, transmitting ACK/NACK, and receiving data. |

| Declaration | void slave_receive(void) |
| Outline     | Slave receive function |
| Argument    | None |
| Variable (global) | Variable name | Content |
|             | iic_index | For the number of transfers |
|             | iic_length | For transmit and receive size |
|             | iic_pointer | Transmit buffer pointer |
| Returned value | None |
| Function    | Receive data and transmit ACK/NACK. |

| Declaration | void slave_transfer(void) |
| Outline     | Slave transmit function |
| Argument    | None |
| Variable (global) | Variable name | Content |
|             | iic_index | For the number of transfers |
|             | iic_pointer | Transmit buffer pointer |
| Returned value | None |
| Function    | Transmit data and receive ACK/NACK. |
### void idle_mode(void)

**Outline**
Transmit and receive mode select function

**Argument**
None

**Variable (global)**
- **Variable name**: iic_mode
  - **Content**: For selecting transmit/receive modes

**Returned value**
None

**Function**
Select transmit mode or receive mode when receiving data.

### unsigned char* select_buffer(unsigned char RW)

**Outline**
Function for obtaining transmit and receive buffer addresses

**Argument**
- **Variable name**: RW
  - **Meaning**: Select transmit and receive buffer
    - 0: Slave receive buffer
    - 1: Slave transmit buffer

**Variable (global)**
None

**Returned value**
- **Type**: unsigned char*
  - **Meaning**: Transmit and receive buffer address

**Function**
Obtain transmit and receive buffer addresses.

### void receive_stop_condition(void)

**Outline**
Stop condition reception state processing function

**Argument**
None

**Variable (global)**
- **Variable name**: iic_mode
  - **Content**: For selecting transmit/receive modes
- **Variable name**: iic_length
  - **Content**: Number of transfers
- **Variable name**: iic_index
  - **Content**: For the number of transfers

**Returned value**
None

**Function**
Clear the stop condition detection interrupt request bit and initialize the transmit/receive mode.
### Multi-Master I²C-bus Interface

#### Declaration
void iic_master_end(unsigned char status)

#### Outline
Master control completion function

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Status after master control</td>
<td>0x10: Master transmission completed 0x11: Arbitration lost is detected during master transmission 0x12: NACK is received during master transmission 0x20: Master reception completed 0x21: Arbitration lost is detected during master reception 0x22: NACK is received during master reception</td>
</tr>
</tbody>
</table>

| Variable (global) | None |
| Returned value    | None |

**Function**
Carry out the processing after master control is completed. This application note does not include any processing. Add if the need arises.

#### Declaration
void iic_slave_end(unsigned char status)

#### Outline
Slave control completion function

<table>
<thead>
<tr>
<th>Argument</th>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Status after slave control completed</td>
<td>0x10: Slave transmission completed</td>
</tr>
</tbody>
</table>

| Variable (global) | None |
| Returned value    | None |

**Function**
Carry out the processing after slave control is completed. This application note does not include any processing. Add if the need arises.

#### Declaration
void stop_condition(void)

#### Outline
Stop condition generation function

| Argument | None |
| Variable (global) | None |
| Returned value    | None |

**Function**
A stop condition is generated.
### 8.6 Flowcharts

#### 8.6.1 I²C-bus Initialization Function

![Flowchart](image-url)
8.6.2 Function for Setting Respective Transmit/Receive Modes

```
mode_ini (unsigned char ms, unsigned char sr)

Transmit mode ?
  Yes
  Set master transmit data
  No

Master mode ?
  Yes
  iic_master_start (*1)
    Note 1. SLAVE_ADD, sr, iic_ram, SEND_TIMES
    Start master transmission
  No

Master start successful ?
  TRUE
    (OK)
  FALSE
    (NG)

iic_length ← SEND_TIMES
Set transmit/receive data size.
```

Argument
ms: Select mode (MASTER: Master mode; SLAVE: Slave mode)
sr: Transmit/receive flag (SND: Transmit; REV: Receive)

Note 1. SLAVE_ADD, sr, iic_ram, SEND_TIMES

iic_master_start
Start master transmission
### 8.6.3 Master Start Function

```c
iic_master_start(1)  

Bus busy ?
   No
   result ← FALSE
   asm("pushc FLG")
   I flag ← 0
   Iic_slave ← slave << 1
   Iic_length ← len
   Iic_pointer ← buf
   Data transmitted ?
      Yes
      Iic_mode ← MODE_M_T
      Iic_rw ← 0
      Iic_slave b0
      I2CSR register ← 0xE0 Start condition
      Are both bits TRS and MST in the I2CSR register 1 ?
         Yes (TRS is 1, MST is 1)
         I2CTRSR register ← iic_slave
         asm("popc FLG")
         result ← TRUE
      No
      Iic_mode ← MODE_M_R
      Iic_rw ← 1
      Iic_slave b0
   No
   Iflag ← 0
   Iic_slave ← slave << 1
   Iic_length ← len
   Iic_pointer ← buf
   return (result)
```

Note 1. unsigned char slave, unsigned char sr, unsigned char *buf, unsigned char len

Argument
slave: Transmit slave address
sr: Transmit/receive flag (SND: Transmit, REV: Receive)
*buf: Transmit/receive buffer pointer
len: Transmit/receive size

Set transmit/receive slave address (set to b7 to b1).
Set transmit/receive size.
Set transmit/receive buffer pointer.
Set transmit/receive slave address (b7 to b1).
Set transmit/receive receive mode.
Set transmit/receive mode.

Argument
slave: Transmit slave address
sr: Transmit/receive flag (SND: Transmit, REV: Receive)
*buf: Transmit/receive buffer pointer
len: Transmit/receive size
8.6.4 Master Transmit Function

```
master_transfer (void)

Arbitration lost detected ?
  Yes
    iic_mode ← MODE_IDLE
    iic_master_end (0x11)  // Complete master control
  No

Address matched ?
  Yes
    80 of the receive data is 1 ?
      (read received)
        slave transmit mode
        i2CSR register ← 0x4F
        idle_mode ()  // Select transmit or receive mode
      (write receive)
        No
          Yes (write receive)
            iic_length--
            stop_condition ()  // Generate stop condition
            iic_master_end (0x12)  // Complete master control
          No
            Yes
              iic_length--
              stop_condition ()  // Generate stop condition
              iic_master_end (0x10)  // Complete master control
              iic_pointer++
              write transmit data.
              increment transmit buffer pointer.

  No
    NACK received ?
      Yes
        Set to idle mode.
      No
        Yes (write receive)
          iic_length--
          stop_condition ()  // Generate stop condition
          iic_master_end (0x10)  // Complete master control
          iic_pointer++
          write transmit data.
          increment transmit buffer pointer.

End
```
8.6.5 Master Receive Function

```c
master_receive (void)

Arbitration lost detected ?
Yes

iic_mode ← MODE_IDLE

Transmit mode ?
Yes

Set to idle mode.

NACK received ?
Yes

stop_condition ()

Generate stop condition

iic_master_end (0x21)

Complete master control

No

Address matched ?
(AAS is 1 ?)
Yes

b0 of the receive data is 1 ?

Yes (write received)

i2CSR register ← 0x4F

Slave transmission

idle_mode ()

Select transmit or receive mode

No (read received)

No

Data received ?

Yes

ACKD bit in the I2CCR register ← 1

NACK is sent

ACK is sent

ACKD bit in the I2CCR register ← 0

iic_pointer ← I2CRSR register

iic_pointer++

Data received ?

No

ACK is sent

ACKD bit in the I2CCR register ← 0

stop_condition ()

Generate stop condition

iic_master_end (0x20)

Complete master control

End
```
### 8.6.6 Slave Receive Function

```plaintext
slave_receive (void)

Last bit received ?

Yes
No

iic_length--

First transmission ?

Yes
No

Data received ?

Yes
No

I2CCCR register
ACKD ← 1

NACK

I2CCCR register
ACKD ← 0

ACK

iic_pointer is 0 ?

Yes
No

return

++iic_index

I2CTRSR register ← 0xFF

Obtain the transmit and receive buffer addresses

select_buffer(0)

iic_pointer ← I2CTRSR register

++iic_pointer

I2CCCR register
ACKD ← 0

iic_length--

End
```

---

**Flowchart Diagram Description:**

1. **slave_receive (void)**
2. Check if the last bit was received?
   - Yes: Subtract 1 from `iic_length`.
   - No: Proceed to the next step.
3. Check if it is the first transmission?
   - Yes: Obtain the transmit and receive buffer addresses using `select_buffer(0)`.
   - No: Go to the internal state.
4. Check if the data was received?
   - Yes: Subtract 1 from `iic_length`.
   - No: Return.
5. Check if `iic_pointer` is 0?
   - Yes: Return.
   - No: Increment `iic_index`.
6. Set `I2CTRSR` register to 0xFF.
7. Repeat the process.
8.6.7  Slave Transmit Function

slave_transfer (void)

NACK received ?

Yes

iic_slave_end (0x10)
Complete slave control

iic_index ← 0
Initialize number of transfers.

I2CTRSR register ← 0xFF

No

First transmission ?

Yes

iic_pointer ← select_buffer(1)
Obtain the transmit buffer address

Fail to obtain the address ?

Yes

I2CTRSR register ← 0x00

return

I2CTRSR register ← *iic_pointer

++iic_pointer

++iic_index

End

8.6.8  Transmit and Receive Mode Select Function

idle_mode(void)

Transmit mode ?

Yes

iic_mode ← MODE_S_T
Set to slave transmit mode.

slave_transfer()
Transmit slave data

iic_mode ← MODE_S_R
Set to slave receive mode.

slave_receive()
Receive slave data

No

End
8.6.9  Function for Obtaining Transmit and Receive Buffer Addresses

```
select_buffer (unsigned char RW)

RW is 1 ?
   No               Receive buffer address
   Yes             Transmit buffer address

return(&sw_buf[0])
return(&sr_buf[0])
```

Argument
RW: Select the transmit or receive buffer (0: Slave receive buffer; 1: Slave transmit buffer)

- RW is 1?
  - No: Receive buffer address
  - Yes: Transmit buffer address

8.6.10  Stop Condition Reception State Processing Function

```
receive_stop_condition (void)

STOP bit in the I2CCR2 register ← 0

Slave receive mode ?
   No
   Yes
      iic_slave_end (0x20)
      Complete slave control

   iic_mode ← MODE_IDLE
   Set to idle mode.

   iic_index ← 0
   Initialize number of transfers.

   I2CCCR2 register ← 0x85
   IIC divided-by-5.
   Standard-clock mode (divide-by 8),
   ACK is sent, ACK clock is generated.

   iic_length ← SEND_TIMES
   Set transmit/receive size.

End
```
### I2C-bus Interface Interrupt Handling

![Flowchart of I2C-bus Interface Interrupt Handling]

- **i2c_bus_interface (void)**
  - Stop condition detect interrupt?
    - Yes: receive_stop_condition()
      - Stop condition reception state processing
      - return
    - No: return
  - General call detected?
    - Yes: return
    - No: return
  - Timeout detected?
    - Yes: return
    - No: Confirm mode
  - Confirm mode
    - iic_mode = MODE_M_T: master_transfer()
      - Transmit master data
      - End
    - iic_mode = MODE_M_R: master_receive()
      - Receive master data
      - End
    - iic_mode = MODE_S_T: slave_transfer()
      - Transmit slave data
      - End
    - iic_mode = MODE_S_R: slave_receive()
      - Receive slave data
      - End
    - iic_mode = MODE_IDLE: idle_mode()
      - Select transmit or receive mode
      - End
9. **Sample Program**

A sample program can be downloaded from the Renesas Electronics website.

10. **Reference Documents**

User’s Manuals
- R32C/118 Group User’s Manual: Hardware Rev.1.00
- R32C/117 Group User’s Manual: Hardware Rev.1.00
- R32C/116 Group User’s Manual: Hardware Rev.1.00
- R32C/118A Group User’s Manual: Hardware Rev.1.00
- R32C/117A Group User’s Manual: Hardware Rev.1.00
- R32C/116A Group User’s Manual: Hardware Rev.1.00
The latest versions 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
- R32C/100 Series C Compiler Package V.1.02 C Compiler User’s Manual Rev.2.00
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
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Aug 2, 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 later described, 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 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 party products or technology 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 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 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 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. 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” – Computer; office equipment; communication equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal 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; submarine repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems); surgical implants; or healthcare intervention (e.g. surgery, etc.), and any other application or purpose that poses 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 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 recomputer 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.

(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.