1. Abstract

The multi-master I²C-bus interface is a serial communication circuit based on the I²C-bus data transmit/receive format, and is equipped with arbitration lost detection that makes multi-master communication possible.

This document describes how to use the I²C-bus interface function.

Note: I²C-bus is a trademark of Philips Electronics N.V.

2. Introduction

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

- MCUs: M16C/63 Group
  - M16C/64C Group
  - M16C/65C Group
  - M16C/6C Group
  - M16C/5L Group
  - M16C/56 Group
  - M16C/5M Group
  - M16C/57 Group

The sample program in this application note can be used with other M16C Family MCUs which have the same special function registers (SFRs) as the above groups. Check the manual 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.\(^{(1)}\)

*Note:*
1. The master transmits general call address 00h to all slaves.

3.2 **Addressing Format**

The 7-bit addressing format is supported.

Only the 7 high-order bits of the I²C address register (slave address) are compared with the address data.

3.3 **I²C-bus Interface Related Pins**

- SCLMM pins: Clock I/O pins of the I²C-bus interface
- SDAMM pins: Data I/O pins of the I²C-bus interface
3.4 Selectable Functions

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

1) Communication mode

There are four communication modes available when performing data communication:

- **Master transmission**: Start and stop conditions are generated (master mode). Address and control data are output to the SDA in synchronization with the SCLMM clock generated by the master device.
- **Master reception**: Data from the transmitting device is received in synchronization with the SCLMM clock generated by the master device.
- **Slave transmission**: Start and stop conditions generated by the master device are received (slave mode).
- **Slave reception**: 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 the following two modes:

- **Standard clock mode**: The bit rate can be selected in the range 16.1 to 100 kHz.
- **High-speed clock mode**: The bit rate can be selected in the range 32.3 to 400 kHz.

3) ACK clock

ACK clock can be selected from the following two modes:

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

4) Data format

Data format can be selected from the following two modes:

- **Addressing format**: The received slave address and the bits SAD6 to SAD0 in the S0Di register (i=0 to 2) 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 example are described in this section. The conditions for the example are as follows.

- Slave address: 7 bits
- Data: 8 bits
- ACK clock available
- Standard clock mode, bit rate: 100 kbps (fIIC: 20 MHz, fVIIC: 4 MHz)
  
  20 MHz (fIIC) divided-by-5 = 4 MHz (fVIIC),
  
  4 MHz (fIIC) divided-by-8 and further divided-by-5 = 100 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, I²C-bus interrupt at the 8th clock (before the ACK clock): Disabled
- Stop condition detection interrupt: Enabled
- Timeout detection interrupt: Disabled
- Set own slave address to the S0D0 register (do not use register S0D1 or S0D2).

If an I²C-bus interrupt at 8th clock (just before ACK clock) is enabled in data receive, a receiver generates ACK or NACK after each byte of data has been received.

4.1 **Initial Settings**

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

1. Write an own slave address to bits SAD6 to SAD0 in the S0D0 register.
2. Write 85h to the S20 register. (CCR value: 5, standard clock mode, ACK clock available)
3. Write 18h to the S4D0 register. (fVIIC: fIIC divided-by-5, timeout interrupt disabled)
4. Write 01h to the S3D0 register. (I²C-bus interrupt at 8th clock (before the ACK clock) is disabled when receiving data and stop condition detect interrupt enabled)
5. Write 0Fh to the S10 register. (slave receive mode)
6. Write 98h to the S2D0 register. (SSC value: 18h, start/stop condition generation timing: long mode)
7. Write 08h to the S1D0 register. (bit counter: 8, I²C-bus interface enabled, addressing format, input level: I²C-bus input)

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. The initial settings are described in 4.1 Initial Settings. Initial settings are assumed to be completed. Programs (A) to (C) below refer to (A) to (C) in the following figure.

![Diagram of Master Transmission](image)

**Figure 4.1 Example of Master Transmission**

(A) Slave address transmission

1. The BB bit in the S10 register must be 0 (bus free).
2. Write E0h to the S10 register. (start condition standby)
3. Write a slave address to the seven most significant bits (MSB) and a 0 to the least significant bit (LSB). (start condition generated, then slave address transmitted)

Note that after a stop condition is generated and the BB bit becomes 0, the S10 register is write disabled for 1.5 cycle of fVIIC. Therefore, when writing E0h to the S10 register and a slave address to the S00 register during the 1.5 fVIIC cycles, a start condition is not generated. When generating a start condition immediately after the BB bit changes from 1 to 0, confirm that both the TRX and MST bits are 1 (transmit 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 S00 register. (data transmission)

(C) Completion of master transmission

(in the I²C-bus interrupt routine)

1. Write C0h to the S10 register. (stop condition standby)
2. Write dummy data to the S00 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. The initial settings are described in 4.1 Initial Settings. Initial settings are assumed to be completed. Programs (A) to (D) below refer to (A) to (D) in the following figure.

(A) Slave address transmission
(1) The BB bit in the S10 register must be 0 (bus free).
(2) Write E0h to the S10 register. (start condition standby)
(3) Write a slave address to the seven most significant bits (MSB) and a 1 to the least significant bit (LSB). (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 S10 register (master receive mode).
(2) Set the ACKBIT bit in the S20 register to 0 (ACK is available) because the data is not the last one.
(3) Write dummy data to the S00 register.

(C) Data reception 2 (data reception)
(in the I²C-bus interrupt routine)
(1) Read the received data from the S00 register.
(2) Set the ACKBIT bit in the S20 register to 1 (no ACK) because the data is the last one.
(3) Write dummy data to the S00 register.

(D) End of master reception
(in the I²C-bus interrupt routine)
(1) Read the received data from the S00 register.
(2) Write C0h to the S10 register. (stop condition standby state)
(3) Write dummy data to the S00 register. (stop condition generated)
4.4 Slave Reception

Slave reception is described in this section. The initial settings are described in 4.1 Initial Settings. Initial settings are assumed to be completed. Programs (A) to (C) below refer to (A) to (C) in the following figure.

- **Program (A):** Start of slave reception
  1. Check the content of S10 register. When the TRX bit is 0 (receive mode), the I2C-bus interface is in slave receive mode.
  2. Write dummy data to the S00 register.

- **Program (B):** Data reception 1
  1. Read the received data from the S00 register.
  2. Set the ACKBIT bit in the S20 register to 0 (ACK is available) because the data is not the last one.
  3. Write dummy data to the S00 register.

- **Program (C):** Data reception 2
  1. Read the received data from the S00 register.
  2. Set the ACKBIT bit in the S20 register to 1 (no ACK) because the data is the last one.
  3. Write dummy data to the S00 register.

![Figure 4.3 Example of Slave Reception](image-url)

- **S:** Start condition
- **A:** ACK
- **W:** Write
- **P:** Stop condition
- **A:** NACK
- **W:** Write

- **SCLMM:** This signal is driven low by accepting an interrupt request or by setting the bit to 0.
- **SDAMM:**
- **IR bit in the IICIC register:**

(A) Start of slave reception

(B) Data reception 1

(C) Data reception 2

End of slave reception
4.5 Slave Transmission

Slave transmission is described in this section. The initial settings are described in 4.1 Initial Settings. Initial settings are assumed to be completed. Program (A) and (B) below refer to (A) and (B) in the following figure. When arbitration lost is detected, the TRX bit becomes 0 (receive mode) even when the bit after the slave address is 1 (read). Therefore, after arbitration lost is detected, read the S00 register. When bit 0 in the S00 register is 1, write 4Fh (slave transmit mode) to the S10 register and execute slave transmission.

![Figure 4.4 Example of Slave Transmission](image)

(A) Start of slave transmission
(in the \(i^2C\)-bus interrupt routine)

1. Check the content of the S10 register. When the TRX bit is 1 (transmit mode), the \(i^2C\)-bus interface is in slave transmit mode.
2. Write transmit data to the S00 register.

(B) Data transmission
(in the \(i^2C\)-bus interrupt routine)

1. Write transmit data to the S00 register.

Write dummy data to the S00 register even if an interrupt occurs at an ACK clock of the last transmitted data. When the S00 register is written, the SCLMM pin becomes high-impedance.
5. Arbitration Lost

The following describes the operation of the I²C-bus interface when arbitration lost occurs. Figure 5.1 shows the Operation Timing of the Arbitration Lost Detect Flag.

![Arbitration Lost Timing Diagram]

**Figure 5.1 Operation Timing of the Arbitration Lost Detect Flag**

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 which enables to receive the slave address. If the selected data format is the addressing format, the slave address can be determined by reading the AAS bit in the S10 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, which enables to receive the data.
6. Interrupt

The I2C-bus interface has the following four interrupt sources:

1) Interrupt when 9-bit transmission/reception is completed (including ACK/NACK)
   The interrupt source can be determined by reading the WIT bit in the S3D0 register. When the WIT 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 WIT bit to 1 enable this interrupt source.
   The interrupt source can be determined by reading the WIT bit. When the WIT 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 SIM bit in the S3D0 register to 1 enables this interrupt source.
   The interrupt source can be determined by reading the SCPIN bit in the S4D0 register. When a stop condition is detected, the SCPIN bit becomes 1.

4) Interrupt when the SCL clock remains high for more than a predetermined time during communication
   Setting the TOE bit in the S4D0 register to 1 enables this interrupt source.
   The interrupt source can be determined by reading the TOF bit in the S4D0 register.
   When the SCL 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.
Multi-Master I²C-bus Interface

Figure 6.1   I²C-bus Interface Interrupt Request Generation Timing

1. **I²C-bus interface interrupt source attributed to setting bits WIT and SIM in the S3D0 register.**

2. **I²C-bus interface interrupt attributed to timeout detection.**

The above figure assumes the following:
1. The TRX bit in the S10 register is 0 (receive mode).
2. The ACKCLK bit in the S20 register is 1 (ACK is available).
7. Notes on I²C-bus Interface

7.1 Generating Start Condition

After a stop condition is generated and BB bit becomes 0 (bus free), the S10 register is write disabled for 1.5 cycles of fVIIC. When writing a slave address to the S00 register afterwards, a start condition is not generated. When generating a start condition immediately after the BB bit changes from 1 to 0, confirm that both the TRX and MST bits are 1, and then write a slave address to the S00 register.
8. Sample Program

This sample program is provided for reference purpose only, and is not guaranteed to operate properly in all system. When incorporating it into a system, careful examination is recommended before using this sample program. Furthermore, since its functionality as integral part of a system cannot be evaluated with this program alone, evaluation with the final system is indispensable.

8.1 Connection Example

Figure 8.1 shows the Connection Example.

![Connection Example Diagram]

Note
1. Pins SCLMM and SDAMM are N-channel open drain output. When connecting, pull-up each pin.

8.2 Operation Conditions

Table 8.1 lists the Sample Program Operation Conditions.

Table 8.1 Sample Program Operation Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral function clock (fIIC)</td>
<td>20 MHz (Xin: 20 MHz, no division mode)</td>
</tr>
<tr>
<td>I²C-bus system clock (fVIIC)</td>
<td>4 MHz (fIIC divided-by-5)</td>
</tr>
<tr>
<td>Bit rate</td>
<td>100 kbps (fVIIC divided-by-8 and further divided-by-5)</td>
</tr>
<tr>
<td>SCL mode</td>
<td>Standard clock mode</td>
</tr>
<tr>
<td>Data format</td>
<td>Addressing mode</td>
</tr>
<tr>
<td>Slave address compare</td>
<td>S0D0 register only</td>
</tr>
<tr>
<td>Stop condition detect interrupt</td>
<td>Enabled</td>
</tr>
<tr>
<td>Data receive interrupt</td>
<td>Enabled</td>
</tr>
<tr>
<td>Timeout detection function</td>
<td>Enabled</td>
</tr>
</tbody>
</table>
8.3  Sample Program Setting

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

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

Figure 8.2 shows the Setting Example of Master Transmission. Figure 8.3 shows the 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*/
  /*======================================================================*/

Figure 8.2  Communication Mode Setting Example

/***********************************************************************************/
/*      DEFINE
/***********************************************************************************/
/*======================================================================*/
/* Modify start*/
/*======================================================================*/
#define M16C5L    0                         /* (1:M16C/5L 0:Other) */
#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 the Master Transmission and Slave Reception Operation Example.

(1) Master: A start condition is generated after writing E0h to the S10 register and transmit data to the S00 register.

(2) Master: The slave address set to bits b7 to b1 in the S00 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 S0D1 register.

(3) Master: After ACK reception, the IR bit in the IICIC 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 ACKBIT 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 ACKBIT 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.

Figure 8.4 Master Transmission and Slave Reception Operation Example
8.4.2 Master Reception and Slave Transmission

Figure 8.5 shows the Master Reception and Slave Transmission Operation.

(1) Master: A start condition is generated after writing E0h to the S10 register and transmit data to the S00 register.
(2) Master: The slave address set to bits b7 to b1 in the S00 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 value in the S0Di register.
   The TRX bit in the S10 register becomes 1 (transmit mode) (only when the ALS bit in the S1D0 register is 1
   (addressing mode)).
(3) Master: After ACK reception, the IR bit in the IICIC 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 ACKBIT 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 ACKBIT bit to 1 and NACK is output.
(7) Slave: After receiving NACK, the TRX 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.

Figure 8.5 Master Reception and Slave Transmission Operation
## 8.5 Function Tables

### Declaration

```
void iic_ini(unsigned char ini, unsigned char sub_address)
```

**Outline**

I²C-bus initialization function

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ini</td>
<td>I²C-bus function enabled/disabled</td>
</tr>
<tr>
<td></td>
<td>ENABLED: I²C-bus function enabled</td>
</tr>
<tr>
<td></td>
<td>DISABLED: I²C-bus function disabled</td>
</tr>
<tr>
<td>sub_address</td>
<td>Slave address setting</td>
</tr>
</tbody>
</table>

### Variable(global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>iic_mode</td>
<td>For selecting communication mode</td>
</tr>
<tr>
<td>iic_index</td>
<td>For the number of transfers</td>
</tr>
</tbody>
</table>

### Returned value

None

### Function

When Argument *ini* = **ENABLED** (I²C-bus function enabled), initializes the I²C-bus before enabling interrupts.

When Argument *ini* = **DISABLED** (I²C-bus function disabled), disables the I²C-bus interface and the interrupt.

---

### Declaration

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

**Outline**

Function for setting respective communication modes

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms</td>
<td>Select master or slave.</td>
</tr>
<tr>
<td></td>
<td>MASTER: Master</td>
</tr>
<tr>
<td></td>
<td>SLAVE: Slave</td>
</tr>
<tr>
<td>sr</td>
<td>Select transmission or reception</td>
</tr>
<tr>
<td></td>
<td>SND: Transmit mode</td>
</tr>
<tr>
<td></td>
<td>REV: Receive mode</td>
</tr>
</tbody>
</table>

### Variable(global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>iic_ram[]</td>
<td>Data storage alignment for master transmit</td>
</tr>
<tr>
<td>iic_length</td>
<td>For transmit and receive size</td>
</tr>
</tbody>
</table>

### Returned value

None

### Function

Sets the respective communication modes.
### Declaration

```c
unsigned char iic_master_start(unsigned char slave, unsigned char sr,
                                  unsigned char *buf, unsigned char len)
```

### Outline

Master start function

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<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></td>
<td>SND: Transmit mode</td>
</tr>
<tr>
<td></td>
<td>REV: Receive mode</td>
</tr>
<tr>
<td>*buf</td>
<td>Pointer to transmit buffer</td>
</tr>
<tr>
<td>len</td>
<td>Transmit/receive data size (0x00 to 0xff)</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<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 to transmit buffer</td>
</tr>
<tr>
<td>iic_mode</td>
<td>For selecting communication mode</td>
</tr>
<tr>
<td>iic_rw</td>
<td>READ/WRITE</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>Master start failure/start successful</td>
</tr>
<tr>
<td></td>
<td>FALSE: Master start failure</td>
</tr>
<tr>
<td></td>
<td>TRUE: Master start successful</td>
</tr>
</tbody>
</table>

### Function

Transmits the start condition and slave address after master setting.
### Multi-Master I²C-bus Interface

#### Declaration

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void master_transfer(void)</th>
</tr>
</thead>
</table>

#### Outline

Master transmit function

#### Argument

None

#### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>iic_mode</td>
<td>For selecting communication mode</td>
</tr>
<tr>
<td>iic_length</td>
<td>For transmit and receive size</td>
</tr>
<tr>
<td>iic_pointer</td>
<td>Transmit buffer pointer</td>
</tr>
</tbody>
</table>

#### Returned value

None

#### Function

Detects arbitration lost, confirms ACK/NACK reception, and transmits data.

---

#### Declaration

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void master_receive(void)</th>
</tr>
</thead>
</table>

#### Outline

Master receive function

#### Argument

None

#### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>iic_length</td>
<td>For selecting communication mode</td>
</tr>
<tr>
<td>iic_index</td>
<td>For transmit and receive size</td>
</tr>
<tr>
<td>iic_pointer</td>
<td>Receive buffer pointer</td>
</tr>
</tbody>
</table>

#### Returned value

None

#### Function

Detects arbitration lost, transmits ACK/NACK, and receives data.

---

#### Declaration

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void slave_receive(void)</th>
</tr>
</thead>
</table>

#### Outline

Slave receive function

#### Argument

None

#### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>iic_length</td>
<td>For transmit and receive size</td>
</tr>
<tr>
<td>iic_index</td>
<td>Number of transfers</td>
</tr>
<tr>
<td>iic_pointer</td>
<td>Receive buffer pointer</td>
</tr>
</tbody>
</table>

#### Returned value

None

#### Function

Receives data and transmits ACK/NACK.

---

#### Declaration

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void slave_transfer(void)</th>
</tr>
</thead>
</table>

#### Outline

Slave transmit function

#### Argument

None

#### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>iic_length</td>
<td>For transmit and receive size</td>
</tr>
<tr>
<td>iic_index</td>
<td>Number of transfers</td>
</tr>
<tr>
<td>iic_pointer</td>
<td>Transmit buffer pointer</td>
</tr>
</tbody>
</table>

#### Returned value

None

#### Function

Receives data and transmits ACK/NACK.
### idle_mode

**Declaration**
```c
void idle_mode(void)
```

**Outline**
Transmit and receive mode select function

**Argument**
None

**Variable (global)**
- **Variable name**: `iic_mode`
- **Content**: For selecting communication mode

**Returned value**
None

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

### select_buffer

**Declaration**
```c
unsigned char* select_buffer(unsigned char RW)
```

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

**Argument**
- **Argument 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**
Obtains transmit and receive buffer addresses.

### receive_stop_condition

**Declaration**
```c
void receive_stop_condition(void)
```

**Outline**
Stop condition reception state processing function

**Argument**
None

**Variable (global)**
- **Variable name**: `iic_mode`
- **Content**: For selecting communication mode
- **Variable name**: `iic_index`
- **Content**: Number of transfers

**Returned value**
None

**Function**
Clears the stop condition detection interrupt request bit and initializes the communication mode.
## Multi-Master I^2^C-bus Interface

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

### Outline
Master control completion function

### Argument
<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Status after master control</td>
</tr>
</tbody>
</table>

### Argument Values
- 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

### Variable(global)
None

### Returned value
None

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

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

### Outline
Slave control completion function

### Argument
<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>Status after slave control completed</td>
</tr>
</tbody>
</table>

### Argument Values
- 0x10: Master transmission completed

### Variable(global)
None

### Returned value
None

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

### Declaration
```c
void stop_condition(void)
```

### Outline
Stop condition generation function

### Argument
None

### Variable(global)
None

### Returned value
None

### Function
Generates a stop condition.
### 8.6 Flowcharts

#### 8.6.1 I\(^2\)C-bus Initialization Function

```
asm("pushc FLG")
```

**I2C-bus mode mode enabled?**

- **Yes (ini is 1)**
  - **S1D0 register ← 0x00**
  - Initialize S1D0 register
  - **I flag ← 0**
  - Disable interrupts
  - **S0D0 register ← sub_address << 1**
  - Set the slave address
  - **S20 register ← 0x85**
  - I\(^2\)C divided-by-5, Standard-clock mode (divide-by 8), ACK is returned, ACK clock is available
  - **S4D0 register ← 0x19**
  - Time out detection function: Enabled
  - I\(^2\)C-bus system clock: f\(\text{IIC}\) divided-by 5

- **No (ini is 0)**
  - **S1D0 register ← 0x03**
  - Select communication mode: Slave receive mode
  - **S10 register ← 0x0F**
  - Recommended value of 4 MHz (11000b)
  - **S2D0 register ← 0x98**
  - Interrupt pin: SDAMM enabled
  - **I flag ← 0**
  - Disable interrupts
  - **IFSR2A register ← 0x04**
  - Interrupt source: I\(^2\)C-bus interface
  - **IICIC register ← 0x01**
  - Set interrupt priority level to 1
  - **iic_mode ← MODE_IDLE**
  - Set to idle mode
  - **iic_index ← 0**
  - Initialize the number of transfers
  - **ES0 bit in the S1D0 register ← 1**
  - Enable I\(^2\)C-bus interface

**.asm("popc FLG")**

**End**
### 8.6.2 Function for Setting Respective Communication Modes

#### Argument
- **ms**: Select communication mode (MASTER: Master mode, SLAVE: Slave mode)
- **sr**: Transmit/receive flag (SND: Transmit, REV: Receive)

#### Code Snippet
```c
mode_ini (unsigned char ms, unsigned char sr)
```

#### Diagram

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

Transmit mode ?
Receive mode ?

Transmit mode

Receive mode

Set master transmit data

Master mode ?
Slave mode ?

Master mode
Slave mode

iic_master_start (1)
Start master transmission

Note 1. SLAVE_ADD,
sr, iic_ram, SEND_TIMES

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

iic_length ← SEND_TIMES

Set transmit/receive data size.

End
```
### 8.6.3 Master Start Function

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

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

```c
iic_master_start (1)
```

1. **Bus busy?**
   - Yes
     - `result ← FALSE`
     - `asm ("pushc FLG")`
     - `i_flag ← 0`
     - `iic_slave ← slave << 1`
     - `iic_length ← len`
     - `iic_pointer ← buf`
   - No (bus free)
     - Data transmitted?
       - Yes (sr is SND)
         - `iic_mode ← MODE_M_T`
         - `iic_rw: iic_slave b0`
         - `S10 register ← 0xE0`
       - No (sr is not SND)
         - Set master transmit mode
           - `iic_mode ← MODE_M_T`
           - `iic_rw: iic_slave b0`
         - Start condition
           - Are both bits TRX and MST in the S10 register 1?
             - Yes (both bits are 1)
               - `S00 register ← iic_slave`
               - `asm ("popc FLG")`
               - `result ← TRUE`
             - No
               - Disable interrupts
                 - `asm ("pushc FLG")`
                 - `i_flag ← 0`
                 - `iic_slave ← slave << 1`
                 - `iic_length ← len`
                 - `iic_pointer ← buf`
                 - Data transmitted?
                   - Yes (sr is SND)
                     - `iic_mode ← MODE_M_T`
                     - `iic_rw: iic_slave b0`
                     - `S10 register ← 0xE0`
                   - No (sr is not SND)
                     - Set master receive mode
                       - `iic_mode ← MODE_M_R`
                       - `iic_rw: iic_slave b0`
```
8.6.4 Master Transmit Function

```c
master_transfer (void)

Arbitration lost detected ?
   Yes (al is 1)
      iic_mode ← MODE_IDLE
      iic_master_end (0x11)  
      Complete master control
   No (al is 0)

NACK received ?
   Yes (lrb is 1)
      stop_condition ()  
      Generate stop condition
   No (lrb is 0)

Address matched ?
   Yes (AAS is 1)
      S10 register ← 0x4F
      Slave transmit mode
   No (AAS is 0)

Write receive ?
   Yes
      iic_pointer++
      Increment transmit buffer pointer
   No
      idle_mode ()  
      Select transmit or receive mode

Data transmit completed
   Yes (iic_length is 0)
      iic_length--
      Write transmit data
   No (iic_length is not 0)
      iic_pointer++
      Increment transmit buffer pointer

stop_condition ()  
Generate stop condition

iic_master_end (0x10)  
Complete master control

iic_master_end (0x12)  
Complete master control

iic_master_end (0x11)  
Complete master control

stop_condition ()  
Generate stop condition

S00 register ← "iic_pointer"

iic_pointer++

End
```
8.6.5 Master Receive Function

master_receive (void)

Arbitration lost detected ?

Yes (al is 1)

ic_mode ← MODE_IDLE

ic_master_end (0x21)

Complete master control

No (al is 0)

Transmit mode

Yes (trx is 1)

NACK received ?

Yes (lrb is 1)

S10 register ← 0xAF

S00 register ← 0xFF

Master transmission

No (lrb is 0)

S10 register ← 0x0F

S00 register ← 0xFF

Write dummy data

Address matched ?

Yes (AAS is 1)

iic_master_end (0x22)

Complete master control

No (AAS is 0)

No (Read received)

b0 of the receive data is 1 ?

Yes (Write received)

S10 register ← 0x4F

Idle mode ()

Select transmit or receive mode

No

WIT bit is 1 ?

Yes

iic_length--

Data received ?

Yes (iic_length is 0)

ACKBIT bit in the S20 register ← 1

ACK is available ACK is not returned

iic_pointer ← S00 register

iic_pointer++

Data received ?

Yes (iic_length is 0)

ACKBIT bit in the S20 register ← 0

ACK is available

S00 register ← 0xFFFF

Write dummy data

No (iic_length is not 0)

stop_condition ()

Generate stop condition

iic_pointer++

ACKBIT bit in the S20 register ← 0

ACK is available

S00 register ← 0xFF

Write dummy data

stop_condition ()

Generate stop condition

iic_master_end (0x20)

Complete master control

End
8.6.6 Slave Receive Function

slave_receive (void)

WIT bit is 1 ?
Yes (I2C-bus interrupt is enabled at 8th clock)

iic_length--

First transmission ?
Yes (index is 0)
No (index is not 0)

Data received ?
Yes

S20 register
ACKBIT ← 1
NACK
S20 register
ACKBIT ← 0

ACK
iic_pointer ← select_buffer(0)
Obtain the transmit and receive buffer addresses

*iic_pointer ← S00 register
++iic_pointer

No (obtained the receive buffer successfully)

iic_pointer is 0 ?
Yes (failed to obtain the receive buffer)

return

No

Data received ?
Yes

++iic_index

S00 register ← 0xFF
iic_pointer ← select_buffer(0)
Obtain the transmit and receive buffer addresses

*iic_pointer ← S00 register
++iic_pointer

No (failed to obtain the receive buffer)

return

End
8.6.7 Slave Transmit Function

slave_transfer (void)

NACK received ?

Yes (lrb is 1)

Complete slave control

iic_slave_end (0x10)

No (lrb is 0)

First transmission ?

Yes (iic_index is 0)

iic_pointer ← select_buffer(1)

Obtain the transmit buffer address

Yes (iic_index is not 0)

No (iic_index is 0)

iic_pointer is 0 ?

Yes (failed)

S00 register ← 0x00

return

No (succeeded)

++iic_index

Initialize number of transfers

iic_index ← 0

S00 register ← 0xFF

S00 register ← *iic_pointer

**iic_pointer

**iic_index

End

Yes (iic_index is 0)

S00 register ← 0x00

return

Yes (failed)

S00 register ← 0x00

return

Yes (iic_index is not 0)

No (iic_index is 0)

++iic_index

Initialize number of transfers

iic_index ← 0

S00 register ← 0xFF

S00 register ← *iic_pointer

**iic_pointer

**iic_index

End
### 8.6.8 Transmit and Receive Mode Select Function

```
idle_mode(void)

Transmit mode ?

- No: Receive mode
- Yes: Transmit mode

  iic_mode ← MODE_S_T

  slave_transfer()
  Transmit slave data

  End

  iic_mode ← MODE_S_R

  slave_receive()
  Receive slave data
```

### 8.6.9 Function for Obtaining Transmit and Receive Buffer Addresses

```
select_buffer (unsigned char RW)

- RW is 1 ? No
- Yes

  return(&sw_buf[0])

  Transmit buffer address

  return(&sr_buf[0])

  Receive buffer address
```

### 8.6.10 Stop Condition Reception State Processing Function

```
receive_stop_condition (void)

SCPIN bit in the S4D0 register ← 0

Slave receive mode ?

- No
- Yes

  iic_slave_end (0x20)

  Complete slave control

  Slave receive completed

  iic_mode ← MODE_IDLE

  Set to idle mode.

  iic_index ← 0

  Initialize number of transfers

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

```plaintext
\[\text{i2c\_bus\_interface (void)\]}

Stop condition detect interrupt?

Yes

- receive\_stop\_condition()
  - Stop condition reception state processing
  - return

No

- General call detected?
  - Yes
    - return
  - No
    - Time out detected?
      - Yes
        - return
      - No
        - Confirm mode
          - iic\_mode is MODE\_M\_T
          - master\_transfer()
            - Transmit master data
          - iic\_mode is MODE\_M\_R
          - master\_receive()
            - Receive master data
          - iic\_mode is MODE\_S\_T
          - slave\_transfer()
            - Transmit slave data
          - iic\_mode is MODE\_S\_R
          - slave\_receive()
            - Receive slave data
          - iic\_mode is default
          - idle\_mode()
            - Select transmit or receive mode

- End
```
9. Sample Code
Sample code can be downloaded from the Renesas Electronics website.

10. Reference Documents
M16C/63 Group User’s Manual: Hardware Rev.1 .00
M16C/64C Group User’s Manual: Hardware Rev.0 .10
M16C/65C Group User’s Manual: Hardware Rev.0 .10
M16C/6C Group User’s Manual: Hardware Rev.1 .00
M16C/5L Group, M16C/56 Group User’s Manual: Hardware Rev.1 .00
M16C/5M Group, M16C/57 Group User’s Manual: Hardware Rev.1 .01
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.

11. 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>Dec. 02, 2009</td>
<td>First edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>Feb. 28, 2011</td>
<td>Add: M16C/64C, M16C/65C, M16C/56, M16C/5M and M16C/57</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 product(s) 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 licenses, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.

4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.

5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose 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 in or omissions from the information included herein.

7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics.

8. 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; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tool; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (autos, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for its 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 implantation, or healthcare intervention (e.g. excision, etc.); and any other applications or purposes that pose a direct threat to human life.

9. 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.

10. 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 reconfigurable computer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.

11. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

12. This document is not to be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.

13. Please contact a Renesas Electronics sales office or Renesas Electronics if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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

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