1. Abstract

This document describes the slave transmit/receive processes in I2C-bus interface slave communication using the M16C/63, 64, 64A, 64C, 65, 65C, 6C, 5LD, 56D, 5L, 56, 5M and 57 Groups serial interface (UART2) special mode 1 (I2C mode).

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

If this application note is applied for using channels except UART2, see the user's manual, and modify registers related to UARTi.

2. Introduction

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

- MCUs: M16C/65 Group and M16C/65C Group
- XIN Clock: 20 MHz

This application note can be used with other M16C Family MCUs which have the same special function registers (SFRs) as the above group. Check the user’s manual for any modifications to functions. Careful evaluation is recommended before using the program described in this application note.
3. Application Example

3.1 Program Outline

I2C-bus interface slave communication (slave transmission/reception) using UART2 special mode 1 is processed in the application example. A maximum of 255 bytes of data can be transmitted/received.

The transmission and reception procedures conform to the I2C-bus communication protocol when used under the following conditions:

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

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

- **Start**
- **(1) Initial setting**
- **(2) UART2 receive interrupt**
- **(3) UART2 transmit interrupt**
- **(4) Stop condition detection interrupt**
The numbers in Figure 3.3 correspond to the numbers indicated in the program processing in the operating timing charts in Figure 3.4 to Figure 3.6.

(1) Initial setting
   Initialize the system clock, UART2 associated SFRs, and variables used.

(2) UART2 receive interrupt
   When a slave address is received, a UART2 receive interrupt is generated at the falling edge of the eighth bit of the SCL clock. The slave address is determined after reading the U2RB register.
   When the slave address is matched:
   • Generate an ACK and set the SCL2 pin to low hold at the ninth bit.
   • Enable the stop condition detection interrupt and UART2 transmit interrupt. Disable the UART2 receive interrupt.
   • Set transmit/receive data to the U2TB register. (1)
   When the slave address is not matched:
   • Generate a NACK.
   After the above processing, release the SCL2 pin low hold at the eighth bit.

(3) UART2 transmit interrupt
   A UART2 transmit interrupt is generated at the falling edge of the ninth bit of the SCL clock. When the first byte (slave address) is received, ACK output set in the UART2 receive interrupt handling is released. When transmitting, determine the ACK/NACK and set the next byte transmit data. When receiving, store the receive data and set ACK for the next byte.

(4) Stop condition detection interrupt
   When a stop condition is detected, an interrupt is generated. SFR values which changed in mid-communication are returned to their initial settings. Disable the stop condition detection interrupt and UART2 transmit interrupt. Enable the UART2 receive interrupt.

Note:
1. When the TXEPT bit in the UiC0 register is 0 (data in the transmit register) in slave mode, write data to the UiTB register.
   If no data exists in the transmit buffer register, the TXEPT bit becomes 1 at the rising edge of the ninth bit of the SCL1 clock.
   The following procedure should be met the condition above.
   • When receiving the first byte (slave address):
     (1) Set the second byte data to the UiTB register in the receive interrupt.
     (2) Set the third byte data to the UiTB register in the transmit interrupt.
   • After the first byte, set fourth or later byte data every transmit interrupt is requested.
I²C-bus Interface Using UART2 Special Mode 1
(Slave Transmit/Receive)

Figure 3.4 Slave Receive Timing
I\(^2\)C-bus Interface Using UART2 Special Mode 1
(Slave Transmit/Receive)

Figure 3.5  Slave Transmit Timing (1)

Figure 3.6  Slave Transmit Timing (2)
3.1.1 Peripheral Functions

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

- I2C mode is used.
- Transfer clock is external clock.
- f1 is used as U2BRG count source.
- SDA2 and SCL2 pins are N-channel open drain output.
- Transfer format is MSB first.
- Transmission completed (TXEPT is 1) is selected as the UART2 transmit interrupt source.
- Clock delay is used.
- Seven to eight cycles of the U2BRG count source is selected as SDA2 digital delay value.
- UART2 initial setting is used.
- SCL2 wait output is enabled.
- SCL2 wait output 2 is disabled.
- SCL2 wait output 3 is enabled.
- SDA2 output disable function is used.
- Start condition detection interrupt is not used.
- Stop condition detection interrupt is used.
- UART2 transmit interrupt is used.
- UART2 receive interrupt is used.

3.1.2 Notes on Using the Attached Sample Program

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

1. Do not use multiple interrupts.
2. The size of the receive buffer and the transmit buffer are set to 255 bytes. The buffer size is defined by the BUFSIZE macro (1 to 255 bytes). When the number of transmit/receive bytes exceeds the size of the buffer, the slave disregards the communication. Disable the UART2 transmit interrupt, and release pins SCL2 and SDA2.
3. After the master generates a stop condition, when the slave processing time (1) has passed, start the next transmit/receive operation (start condition is generated/generate a start condition).
4. If pins SDAi and SCLi are not N-channel open drain, set the NCH bit in the UiC0 register to 1 (pins TXDi/SDAi and SCLi are N-channel open drain output).
5. UART0 and UART1 transmit interrupt source select bits are existed in the UCON register in M16C/60 Series. Set the U0IRS bit or the U1IRS bit in the UCON register to 1 (transmission completed (TXEPT = 1)).

Note:
1. The slave processing time indicates the time between detecting a stop condition and enabling I2C mode in the main processing, and is dependent on the processing of the user program. The maximum processing time for this sample program is approximately 500 μs.
3.2 Memory

Table 3.2 Memory

<table>
<thead>
<tr>
<th>Memory</th>
<th>Size</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>718 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>RAM</td>
<td>6 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>Maximum user stack</td>
<td>21 bytes</td>
<td></td>
</tr>
<tr>
<td>Maximum interrupt stack</td>
<td>27 bytes</td>
<td></td>
</tr>
</tbody>
</table>

Usage memory size varies depending on C compiler version and compile options. The above applies under the following conditions:
- C compiler: M16C/60, 30, 20, 10, Tiny, R8C/Tiny Series Compiler V.5.45 Release 01
- C compile option: -c -finfo -dir "$(CONFIGDIR)"
4. Software

This chapter shows the program example to set the example described in chapter 3. Application Example. Refer to the latest hardware user’s manual for details on individual registers.

4.1 Variables

Definition file name: rej05b1423_src.c

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

Definition file name: iic.c

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size /Bit Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte_dt iic_str</td>
<td>-</td>
<td>Structure to store statuses</td>
</tr>
<tr>
<td>iic_status</td>
<td>1 byte</td>
<td>All statuses</td>
</tr>
<tr>
<td>iic_rw</td>
<td>b0</td>
<td>R/W flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Write (W) slave receive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Read (R) slave transmit</td>
</tr>
<tr>
<td>iic_buf_full</td>
<td>b1</td>
<td>Buffer full flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Within buffer size (transmit/receive bytes &lt; buffer size)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Buffer full (transmit/receive bytes ≥ buffer size)</td>
</tr>
<tr>
<td>iic_end</td>
<td>b2</td>
<td>Communication completed flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Busy (mid-communication)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Ready (except for mid-communication)</td>
</tr>
<tr>
<td></td>
<td>b7 to b3</td>
<td>Not used (undefined)</td>
</tr>
<tr>
<td>static unsigned char far* iic_pointer</td>
<td>4 bytes</td>
<td>Transmit/receive buffer pointer</td>
</tr>
<tr>
<td>static unsigned char iic_index</td>
<td>1 byte</td>
<td>Number of transmit/receive bytes</td>
</tr>
</tbody>
</table>
## 4.2 Function Tables

### Declaration
```c
void main(void)
```

### Outline
Main processing

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

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

### Returned value
<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Function
After setting the system clock, I²C mode is enabled. Communication status is determined by the returned value of the iic_slave_end function. Each status is processed after communication is completed, and the uart2_init function is called to enable I²C mode.

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

### Outline
System clock setting

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

### Variable (global)
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
</tr>
</tbody>
</table>

### Returned value
<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Function
Call this function from the main processing. Set the system clock (XIN clock).

### Declaration
```c
void uart2_init(unsigned char ini)
```

### Outline
UART2 initial setting

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

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

### Returned value
<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Function
Call this function from the main processing. Initialize SFRs to use UART2 special mode 1 (I²C mode). When I²C mode is enabled, set iic_status to 0x00 (clear all statuses). When executing this function, interrupts are disabled by the I flag.
### I2C-bus Interface Using UART2 Special Mode 1

#### (Slave Transmit/Receive)

**Declaration**

`void _stop_condition_detection(void)`

**Outline**

Stop condition detection interrupt handling

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

<table>
<thead>
<tr>
<th>Variable (global)</th>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned value</th>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Function**

An interrupt is generated when a stop condition is detected, and the **stp_int** function is called.

**Declaration**

`static void stp_init(void)`

**Outline**

Stop condition detection processing

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

<table>
<thead>
<tr>
<th>Variable (global)</th>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned value</th>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Function**

Called from the stop condition detection interrupt handling. UART2 associated SFR values changed mid-communication are returned to their initial values, and the communication completed flag is set to 1.

**Declaration**

`void uart2_receive(void)`

**Outline**

UART2 receive interrupt handling

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

<table>
<thead>
<tr>
<th>Variable (global)</th>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Returned value</th>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Functions**

An interrupt is generated at the falling edge of the eighth bit of the SCL clock. This function calls the iic_id_check function after reading the U2RB register in the function header.

- When the slave address is matched, generate an ACK, and set the SCL2 pin to low hold at the ninth bit. The receive interrupt is disabled, and the transmit interrupt and stop condition detection interrupt are enabled. The number of transmit/receive bytes and all statuses are cleared. When the slave is receiving, set the ACK for the next byte. When the slave is transmitting, set transmit data for the next byte.
- When the slave address is not matched, generate a NACK. After the above processing, release the SCL2 pin low hold.
<table>
<thead>
<tr>
<th>Declaration</th>
<th>unsigned char* iic_id_check(unsigned char id, unsigned char rw)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Slave address determine processing</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>unsigned char id</td>
</tr>
<tr>
<td></td>
<td>unsigned char rw</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>unsigned char*</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Called from the UART2 receive interrupt handling. The received slave address is determined. When the slave address is matched, the returned value is the buffer address. When the slave address is not matched, the returned value is NULL.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void _uart2_transmit(void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>UART2 transmit interrupt handling</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>unsigned char iic_index</td>
</tr>
<tr>
<td></td>
<td>(structure member) iic_rw</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>An interrupt is generated at the falling edge of the ninth bit of the SCL clock. The U2RB register is read in the function header. When the first byte (slave address) is received, disable ACK output set by the receive interrupt handler. After the first byte is received, the slave_rcv_int function is called when the slave is receiving and the slave_trn_int function is called when the slave is transmitting.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>static void slave_rcv_int(unsigned char rb_data)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Slave receive processing</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>unsigned char rb_data</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>unsigned char iic_index</td>
</tr>
<tr>
<td></td>
<td>unsigned char far* iic_pointer</td>
</tr>
<tr>
<td></td>
<td>(structure member) iic_buf_full</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Called from the UART2 transmit interrupt handling. The argument value is stored in the receive buffer (except the slave address). When the number of received bytes is less than the buffer size, set an ACK for the next byte. Release the SCL2 pin low hold, then enable the SCL2 pin to low hold for the next byte. When the number of received bytes is the same as or greater than the buffer size, the buffer full flag is set to 1. Release pins SCL2 and SDA2, and disable the UART2 transmit interrupt.</td>
</tr>
</tbody>
</table>
### I2C-bus Interface Using UART2 Special Mode 1

#### Slave Transmit/Receive

**Declaration**

```
static void slave_trn_int(unsigned char rb_data)
```

**Outline**

Slave transmit processing

**Argument**

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char rb_data</td>
<td>ACK/NACK read from the U2RB register</td>
</tr>
</tbody>
</table>

**Variable (global)**

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

**Returned value**

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

**Function**

Called from the UART2 transmit interrupt handling.

- When an ACK is detected and the number of transmit bytes is less than the buffer size, set transmit data for the next byte. Release the SCL2 pin low hold, then enable the SCL2 pin to low hold for the next byte.
- When the number of transmit bytes is the same as or greater than the buffer size, set the buffer full flag to 1. Release pins SCL2 and SDA2, and disable the UART2 transmit interrupt.
- When a NACK is detected, release pins SCL2 and SDA2, and disable the UART2 transmit interrupt.

---

**Declaration**

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

**Outline**

Slave control completed processing

**Argument**

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

**Variable (global)**

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

**Returned value**

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned short</td>
<td>Lower byte</td>
<td>IIC_BUSY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IIC_REND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IIC_TEND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IIC_ERR</td>
</tr>
<tr>
<td>Upper byte</td>
<td>1 to 255</td>
<td>Number of transmit/receive bytes</td>
</tr>
</tbody>
</table>

**Function**

Called from the main processing. It informs the user of the state of slave control completion.

- When the communication completed flag is 1 and there is transmit/receive data except for the slave address, disable I2C mode. Otherwise, return IIC_BUSY (mid-communication).
- After disabling I2C mode, when the communication completed flag is 0, the next communication is determined to be started and the IIC_ERR (overrun error detection) function is returned. When the communication completed flag is 1, return IIC_REND (reception completed) or IIC_TEND (transmission completed).
4.3 Main Processing

main()

asm("FCLR I")

System clock setting
mcu_init()

asm("FSET I")

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

UART2 initial setting
uart2_init()

Slave control completed processing
iic_slave_end()

temp.all ← Returned value

temp.byte.byte0

= IIC_REND (reception completed)
= IIC_TEND (transmission completed)
= IIC_ERR (overrun error)

Read receive buffer (1)
Set transmit data (1)
Initialize receive buffer (1)

UART2 initial setting
uart2_init()

UART2 initial setting
uart2_init()

UART2 initial setting
uart2_init()

IIC special mode 1 (I²C mode) enabled

Note:
1. Additional processing can be added as needed.
4.4 System Clock Setting Processing

```
mcu_init()

PRCR ← 0x03
PM0 ← 0x00
PM1 ← 0x08
CM2 ← 0x00
CM1 ← 0xA0
CM0 ← 0x08
CM1 ← 0x60
CM1 ← 0x20
UCLKSEL0 (1) ← 0x00
PCLKR ← 0x03
PRCR ← 0x00

Note:
1. The UCLKSEL0 register may not be existed depending on MCU products.
```
4.5 UART2 Initial Setting

uart2_init()

asm("FCLR I")

ini = 1?

= 1 (I²C mode enabled)

BCNIC ← 0x00

S2TIC ← 0x00

U2C1 ← 0x00

U2MR ← 0x0a

U2SMR2 ← 0x11

U2SMR3 ← 0xe2

U2SMR4 ← 0x30

U2C0 ← 0x90 (1)

U2C1 ← 0x10

BCNIC ← 0x00

S2TIC ← 0x00

S2RIC ← 0x01

iic_status ← 0x00

U2C1 ← 0x15

asm("FSET I")

return

[Argument] unsigned char ini
0: I²C mode disabled
1: I²C mode enabled

Disable interrupts.

U2SMR ← 0x01

Select I²C mode.

(PD_IIC ←
P_IIC | P_IIC_INIT)

PD_IIC ←
PD_IIC & PD_IIC_INIT

BCNIC ← 0x00

S2TIC ← 0x00

U2C1 ← 0x00

U2SMR2 ← 0x01

Disable UART2 transmit interrupt.

Disable transmission/reception.

U2MR ← 0x00

Select I²C mode, select external clock.

Select UART2 transmit/UART2 receive interrupt, enable UART2 initialization.

Clock phase: With clock delay
SDA2 digital delay:
Seven to eight cycles of U2BRG count source

U2SMR4 ← 0x30

Enable NACK output.

U2C0 ← 0x90 (1)

Transfer format: MSB first

Disabling transmission/reception.

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

Set IR bit to 0.

Enable UART2 receive interrupt.

Clear all status flags.

Enable transmission/reception.

Enable interrupts.

Note:

1. Ports P7_0 and P7_1 in the M16C/50 Series are not N-channel open drain outputs.
   When using M16C/50 Series, set the NCH bit in the U2C0 register to 1 (pins TXDi/SDAi and SCl i are N-channel
   open drain output) before set initial value of SDAi output (P7_0).
4.6 Stop Condition Detection Interrupt Handling

```plaintext
_stop_condition_detection()

BBS = 0? = 0 (start condition detection (busy))
= 0 (stop condition detection)

Stop condition detection processing
stp_int()
```

4.7 Stop Condition Detection Processing

```plaintext
stp_int()

U2C1 ← 0x10 Disable transmission/reception.
_P_IIC ← P_IIC | P_IIC_INIT Set the initial value: P7_0 (SDA2) = 1 (high)
U2MR ← 0x00 Disable serial interface.
U2SMR4 ← 0x30 Output NACK (release SDA2 pin).
U2MR ← 0x0A Select I²C mode, select external clock.
U2SMR2 ← 0x11 Select UART2 transmit/UART2 receive interrupt.
BCNIC ← 0x00 Disable stop condition detection interrupt.
S2TIC ← 0x00 Disable UART2 transmit interrupt.
S2RIC ← 0x01 Enable UART2 receive interrupt.
iic_end ← 1 Set communication completed flag to 1.
U2C1 ← 0x15 Enable transmission/reception.
return
```
4.8 UART2 Receive Interrupt Handling

```c
_uart2_receive()

  temp.all ← U2RB

  Slave address determine processing
    iic_id_check()
  iic_pointer ← Returned value

  (slave address not match) = NULL
    iic_pointer = NULL?

      ≠ NULL (slave address match)
        U2SMR4 ← 0xA0
        U2BRG count source × Waiting time of SDA2 digital delay or more
        S2TC ← (S2TIC | 0x01) & 0x0F
        S2RIC ← 0x00
        BCNIC ← 0x01
        iic_index ← 0
        iic_status ← 0x00
        iic_rw ← temp.bit.b8

      = 0 (slave receive)
        iic_rw = 0?

          ≠ 0 (slave transmit)
            temp.byte.byte0 ← *iic_pointer
            temp.byte.byte1 ← 0x01
            iic_pointer++
            U2TB ← temp.all

          = 0 (slave receive)
            U2SMR2 ← 0x11

  Read the U2RB register.

  Generated an ACK, set the SCL2 pin to low hold at the ninth bit.

  Wait releasing SDA2 pin

  Enable UART2 transmit interrupt.

  Disable UART2 receive interrupt.

  Enable stop condition detection interrupt.

  Clear number of transmit/receive bytes.

  Clear all status flags.

  Set R/W (1: Read; 0: Write).

  = NULL (slave address match)

  Set the next byte transmit data.

  Set data to release SDA2 pin on ninth bit.

  Pointer transmit buffer + 1

  Set the U2TB register to transmit data (transmit start).

  Release SCL2 pin to low hold.
```

4.9 Slave Address Determine Processing

```c
iic_id_check()

(id & 0x7F) ≠ DEVICE_ADDRESS (slave address not match)

(id & 0x7F) = DEVICE_ADDRESS (slave address match)

(rw & 0x01) = 0x00 (slave transmit)

(rw & 0x01) ≠ 0x00 (slave receive)

return(NULL)

(id & 0x7F) ≠ DEVICE_ADDRESS (slave address not match)

(id & 0x7F) = DEVICE_ADDRESS (slave address match)

(rw & 0x01) = 0x00 (slave address match)

(rw & 0x01) ≠ 0x00 (slave transmit)

return(iic_rx)

return(iic_tx)

return(NULL)
```

4.10 UART2 Transmit Interrupt Handling

```c
_uart2_transmit()

temp.all ← U2RB

iic_index = 0?

= 0 (from second byte on)

= 0 (first byte (slave address))

ACKC ← 0

Disable ACK output.

Wait releasing SDA2 pin(1)

iic_rw = 0?

= 0 (slave transmit)

= 0 (slave receive)

Slave receive processing

slave_rcv_int()

Slave transmit processing

slave_trn_int()

return
```

Note:

1. U2BRG count source × Waiting time of SDA2 digital delay or more
4.11 Slave Receive Processing

slave_rcv_int()

iic_index < BUFSIZE?

≥ BUFSIZE (buffer size or greater)

< BUFSIZE (less than buffer size)

iic_index = 0?

≠ 0 (from the second byte on)

*iic_pointer ← rb_data

Receive data stored to the receive buffer.

Pointer to receive buffer + 1

iic_pointer++

iic_index++

Number of transmit/receive bytes + 1

U2TB ← 0x00FF

Release SDA2 pin.

SWC9 ← 0

Release SCL2 pin.

SWC9 ← 1

After ninth bit received, set the SCL2 pin to low fixed.

return

*iic_pointer ← rb_data

Receive data stored to the receive buffer.

iic_buf_full ← 1

Set buffer full flag to 1.

U2SMR4 ← 0xB0

Release SDA2 pin (SWC9 is 0).

U2SMR4 ← 0x30

Release SCL2 pin.

S2TIC ← 0x00

Disable UART2 transmit interrupt.

Note:
1. U2BRG count source × Waiting time of SDA2 digital delay or more
### 4.12 Slave Transmit Processing

#### [Argument]

```c
unsigned char rb_data: ACK/NACK data read from the U2RB register
```

#### slave_tm_int()

1. **iic_index ≥ BUFSIZE (buffer size or greater)**
   - `iic_buf_full ← 1`
   - `U2SMR4 ← 0xB0`
   - Release SDA2 pin.

2. **iic_index < BUFSIZE (less than buffer size)**
   - `U2SMR4 ← 0x30`
   - Release SCL2 pin (SWC9 is 0).
   - `S2TIC ← 0x00`
   - Disable UART2 transmit interrupt.

3. **(rb_data & 0x01) = 0x00 (NACK detection)**
   - `iic_index++`
   - Number of transmit/receive bytes + 1
   - `U2SMR4 ← 0xB0`
   - Release SDA2 pin.

4. **iic_index ≥ (BUFSIZE - 1)**
   - `iic_index++`
   - Number of transmit/receive bytes + 1
   - `U2SMR4 ← 0xB0`
   - Release SDA2 pin (SWC9 is 0).
   - `S2TIC ← 0x00`
   - Disable UART2 transmit interrupt.

5. **iic_index < (BUFSIZE - 1)**
   - `temp.byte.byte0 ← *iic_pointer`
   - Set the next byte transmit data.
   - `temp.byte.byte1 ← 0x01`
   - Set data to release SDA2 pin at the ninth bit.
   - `iic_pointer++`
   - Pointer transmit buffer + 1
   - `U2TB ← temp.all`
   - Set transmit data to the U2TB register (transmission started).
   - `Wait releasing SDA2 pin (1)`

6. **iic_index++**
   - Number of transmit/receive bytes + 1
   - `SWC9 ← 0`
   - Release SCL2 pin low fixed setting.
   - `SWC9 ← 1`
   - After ninth bit received, set the SCL2 pin to low fixed.

7. **return**

---

**Note:**

1. U2BRG count source × Waiting time of SDA2 digital delay or more

---
4.13 Slave Control Completed Processing

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

**Diagram:**
- `iic_slave_end()`
- `(iic_end = 0) or (iic_index ≤ 1)` (mid-communication or no transmit/receive data)
- `(iic_end = 1) and (iic_index > 1)` (other than mid-communication and with transmit/receive data)
- `UART2 initial setting`
  - `uart2_init()` Disable UART2 special mode 1 (I2C mode).
  - `iic_end = 0?` ≠ 0 (mid-communication)
  - `iic_end = 1?` ≠ 0 (other than mid-communication)
  - `iic_buf_full = 1?` ≠ 1 (within buffer size)
  - `temp.byte.byte0 ← IIC_BUSY`
  - `temp.byte.byte1 ← iic_index`
  - `iic_rw = 0?` ≠ 0 (slave transmit)
  - `temp.byte.byte0 ← IIC_REND`
  - `temp.byte.byte0 ← IIC_TEND`
  - `temp.byte.byte1 ← iic_index - 1`
  - Set number of transmit/receive bytes.
  - Set status completed.
  - `temp.byte.byte0 ← IIC_ERR`
- `(iic_end = 0) or (iic_index ≤ 1)` (other than mid-communication and with transmit/receive data)
- `temp.byte.byte0 ← IIC_BUSY`
- Set status (mid-communication).
- `temp.byte.byte0 ← IIC_ERR`
- Set status (overrun error).
5. **Sample Program**

A sample program can be downloaded from the Renesas Electronics website. To download, click “Application Notes” in the left-hand side menu of the M16C Family Series page.

6. **Reference Documents**

**Application Note**
M16C Family I2C-bus Interface Using UARTi Special Mode 1
(REJ05B1349-0100)
The latest version can be downloaded from the Renesas Electronics website.

M16C/63 Group User’s Manual: Hardware Rev.1.00
M16C/64 Group User’s Manual: Hardware Rev.1.05
M16C/64A Group User’s Manual: Hardware Rev.1.10
M16C/64C Group User’s Manual: Hardware Rev.1.00
M16C/65 Group User’s Manual: Hardware Rev.1.10
M16C/65C Group User’s Manual: Hardware Rev.1.00
M16C/6C Group User’s Manual: Hardware Rev.1.00
M16C/5LD Group, M16C/56D Group User’s Manual: Hardware Rev.1.10
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.

**C Compiler Manual**
M16C Series, R8C Family C Compiler Package V.5.45
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 31, 2010</td>
<td>— First edition issued</td>
</tr>
<tr>
<td>1.01</td>
<td>Jan 31, 2011</td>
<td>— Add: M16C/64, M16C/64C, M16C/65C, M16C/56D, M16C/56, M16C/57</td>
</tr>
<tr>
<td>1.02</td>
<td>Mar 10, 2011</td>
<td>8 Modify: ROM size is 718 byte in Table3.2.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>18 Add: “Wait releasing SDA2 pin” in “4.8 UART2 Receive Interrupt Handling”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>19 Add: “Wait releasing SDA2 pin” in “4.10 UART2 Transmit Interrupt Handling”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>20 Add: “U2SMR4 ← 0xB0”, “Wait releasing SDA2 pin” in “4.11 Slave Receive Processing”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>21 Add: “U2SMR4 ← 0xB0”, “Wait releasing SDA2 pin” in “4.12 Slave Transmit Processing”</td>
</tr>
</tbody>
</table>

All trademarks and registered trademarks are the property of their respective owners.
General Precautions in the Handling of MPU/MCU Products

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

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

2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
     In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
     In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products
   Before changing from one product to another, i.e. to one with a different part number, confirm that the change will not lead to problems.
   - The characteristics of MPU/MCU in the same group but having different part numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different part numbers, implement a system-evaluation test for each of the products.
Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.

2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No 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 manufacturer, 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 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.

9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of recompute 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 non-compliance 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 Electronic" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

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