1. Abstract

This document describes the master transmit/receive processes in I²C-bus interface single master communication using the R32C/100 Series serial interface (UART2) special mode 1 (I²C mode).

Seven channels (UART0 to UART6) can be used in special mode 1 in the R32C/118 Group. If channels other than UART0 to UART6 are used, refer to the hardware user's manual and modify the registers associated with UARTi (i = 0 to 6).

2. Introduction

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

• MCU: R32C/118 Group
• XIN Clock: 16 MHz

This application note can be used with other R32C/100 Series 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

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

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

Note:

1. The setting value is 378 kbps.

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

In this application example, the actual transfer rate becomes approximately 350 kbps since the clock synchronous function is enabled (reference value: pull up voltage 5 V, pull up resistance 1 kΩ).

Standard-mode and Fast-mode are supported.

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

![Figure 3.1 Communication Format](image-url)
I²C-bus Interface Using UARTi Special Mode 1
(Master Transmit/Receive)

Figure 3.2 Block Diagram
Figure 3.3 Outline Flowchart

The numbers in Figure 3.3 correspond to the numbers indicated in the program processing in the operating timing charts in Figure 3.4 to Figure 3.6.

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

2. Start master control
   Enable the start/stop condition generation interrupt and generate a start condition.

3. Start/stop condition generation interrupt
   An interrupt request is generated when start condition generation is completed and a stop condition is detected. When start condition generation is completed, the UART2 transmit interrupt is enabled and the slave address is transmitted. When a stop condition is detected, SFR values which changed during communication are returned to their initial values.

4. UART2 transmit interrupt
   A UART2 transmit interrupt is generated at the falling edge of the ninth bit of the SCL clock. When transmitting, set the next byte transmit data. When receiving, set ACK/NACK for the next byte. When communication is completed, generate a stop condition.
I2C-bus Interface Using UARTi Special Mode 1
(Master Transmit/Receive)

Figure 3.4 Master Transmit Timing
I²C-bus Interface Using UARTi Special Mode 1
(Master Transmit/Receive)

Figure 3.5  Master Receive Timing (1)

Figure 3.6  Master Receive 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 internal clock source.
- f1 is used as U2BRG count source.
- SDA2 and SCL2 pins are N-channel open drain.
- Transfer format uses MSB first.
- Transmission completed (TXEPT is 1) is selected as the UART2 transmit interrupt source.
- Clock phase setting is clock delay.
- Seven to eight cycles of U2BRG count source is selected as SDA2 digital delay value.
- Clock synchronization is enabled.
- SCL2 wait output function 2 is not used.
- SDA2 output disable function is not used.
- Start/stop condition generation interrupt is used.
- UART2 transmit interrupt is used.
- UART2 receive interrupt is not used.
- PLL clock is 100 MHz.
- Base clock is 50 MHz.
- CPU clock is 50 MHz.
- Peripheral bus clock is 25 MHz.
- Peripheral clock is 25 MHz.
- Transfer rate is approximately 378 kbps.

Calculating the transfer rate:

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

\[
= \frac{25 \text{ MHz (f1)}}{2 \times (32 + 1)}
\]

\[\approx 378.788 \text{ kbps}\]

### Table 3.1 Pins Used and Their Function

<table>
<thead>
<tr>
<th>Pin</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P7_1/SCL2</td>
<td>I/O</td>
<td>I²C mode clock I/O pin</td>
</tr>
<tr>
<td>P7_0/SDA2</td>
<td>I/O</td>
<td>I²C mode data I/O pin</td>
</tr>
</tbody>
</table>

### 3.1.2 Notes on Using the Attached Sample Program

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

- Do not use multiple interrupts.
- When setting the system clock to anything other than the XIN clock (16 MHz), change the setting value of the U2BRG count source and the U2BRG register according to the transfer rate calculation shown in **3.1.1 Peripheral Functions**.
3.2 Memory

### Table 3.2 Memory

<table>
<thead>
<tr>
<th>Memory</th>
<th>Size</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROM</td>
<td>496 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>RAM</td>
<td>8 bytes</td>
<td>In the iic.c module</td>
</tr>
<tr>
<td>Maximum user stack</td>
<td>24 bytes</td>
<td></td>
</tr>
<tr>
<td>Maximum interrupt stack</td>
<td>64 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: R32C/100 Series C Compiler V.1.02 Release 01
- 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: rej05b1395.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 retry_counter</td>
<td>1 byte</td>
<td>Count number of communication retries</td>
</tr>
</tbody>
</table>

Definition file name: iic.c

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static byte_dt iic_str1</td>
<td></td>
<td>- Structure to store slave address</td>
</tr>
<tr>
<td>iic_slave_addr</td>
<td>1 byte</td>
<td>Slave address</td>
</tr>
<tr>
<td>iic_rw</td>
<td>b0</td>
<td>R/W flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Write (W)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Read (R)</td>
</tr>
<tr>
<td></td>
<td>b7 to b1</td>
<td>7-bit address</td>
</tr>
<tr>
<td>static byte_dt iic_str2</td>
<td></td>
<td>- Structure to store status</td>
</tr>
<tr>
<td>iic_status</td>
<td>1 byte</td>
<td>All statuses</td>
</tr>
<tr>
<td>iic_start</td>
<td>b0</td>
<td>Mid-communication flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: Communication completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Mid-communication</td>
</tr>
<tr>
<td>iic_err_par</td>
<td>b1</td>
<td>Parameter error flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: No error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: Parameter error</td>
</tr>
<tr>
<td>iic_err_nack</td>
<td>b2</td>
<td>NACK detection error flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: No error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: NACK detection error</td>
</tr>
<tr>
<td>iic_err_addr</td>
<td>b3</td>
<td>No address match error flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0: No error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1: No address match error</td>
</tr>
<tr>
<td></td>
<td>b7 to b4</td>
<td>Not used (undefined)</td>
</tr>
<tr>
<td>unsigned char iic_length</td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td>unsigned char iic_index</td>
<td>1 byte</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>unsigned char far*iic_pointer</td>
<td>4 bytes</td>
<td>Transmit/receive buffer pointer</td>
</tr>
</tbody>
</table>
## 4.2 Function Tables

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void main (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Main processing</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>unsigned char iic_tx[BUFSIZE]</td>
</tr>
<tr>
<td></td>
<td>unsigned char iic_rx[BUFSIZE]</td>
</tr>
<tr>
<td></td>
<td>unsigned char retry_counter</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>After initializing the system clock and UART2, master transmission and reception are repeated alternately. After calling the iic_master_start function to start master control, call the iic_master_end function and wait for master control to be completed. When the iic_master_end function returns ADD_ERR (communication stop because of address not matched error), communication is retried.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void SetPLLClock (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>PLL mode setting</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Call this function from the main processing. Process for transition to PLL mode. Set the peripheral clock source to 25 MHz.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void uart2_init (unsigned char ini)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>UART2 initial setting</td>
</tr>
<tr>
<td>Argument</td>
<td>Argument name</td>
</tr>
<tr>
<td></td>
<td>unsigned char ini</td>
</tr>
<tr>
<td>Variable (global)</td>
<td>Variable name</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>Call this function from the main processing. Initialize the SFRs used for UART2 in special mode 1 (I^2C mode).</td>
</tr>
</tbody>
</table>
### I2C-bus Interface Using UARTi Special Mode 1

#### Master Transmit/Receive

#### Declaration

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

#### Outline

Master control start processing

#### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char addr</td>
<td>0x00 to 0x7F: Specify slave address</td>
</tr>
<tr>
<td>unsigned char rw</td>
<td>0x00: Master transmit 0x01: Master receive</td>
</tr>
<tr>
<td>unsigned char far *buf</td>
<td>Transmit or receive buffer pointer</td>
</tr>
<tr>
<td>unsigned char len</td>
<td>0x01 to 0xFF: Transfer data length</td>
</tr>
</tbody>
</table>

#### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(structure member) iic_status</td>
<td>All statuses</td>
</tr>
<tr>
<td>(structure member) iic_start</td>
<td>Mid-communication flag</td>
</tr>
<tr>
<td>(structure member) iic_err_par</td>
<td>Parameter error flag</td>
</tr>
<tr>
<td>(structure member) iic_slave_addr</td>
<td>Slave address</td>
</tr>
<tr>
<td>unsigned char iic_length</td>
<td>Transfer data length</td>
</tr>
<tr>
<td>unsigned char far *iic_pointer</td>
<td>Transmit/receive buffer pointer</td>
</tr>
</tbody>
</table>

#### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char BUSY</td>
<td>Bus busy</td>
<td></td>
</tr>
<tr>
<td>unsigned char RDY</td>
<td>Bus free</td>
<td></td>
</tr>
<tr>
<td>PAR_ERR</td>
<td>Parameter error</td>
<td></td>
</tr>
</tbody>
</table>

#### Function

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

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

### Start/stop condition detection

#### Declaration

```
void _start_stop_condition_detection (void)
```

#### Outline

Start/stop condition generation interrupt handling

#### Function

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

### Outline
Start condition detection processing

### Argument
<table>
<thead>
<tr>
<th>Variable 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_slave_addr</td>
<td>Slave address</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>None</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### Function
Called from the start/stop condition generation interrupt handling. UART2 transmit/receive interrupt is enabled. Transmit the slave address.

---

### Declaration
```
static void stp_int (void)
```

### Outline
Stop condition detection processing

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

### Variable (global)
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>(structure member) iic_start</td>
<td>Mid-communication flag</td>
</tr>
</tbody>
</table>

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

### Function
Called from the start/stop condition generation interrupt handling. SFR associated with UART2 values which changed during communication are returned to their values, and the mid-communication flag is set to 0.

---

### Declaration
```
void _uart2_trans (void)
```

### Outline
UART2 transmit interrupt handling

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

### Variable (global)
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>(structure member) iic_err_addr</td>
<td>No address match error flag</td>
</tr>
<tr>
<td>(structure member) iic_RW</td>
<td>R/W flag</td>
</tr>
</tbody>
</table>

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

### Function
An interrupt is generated at the falling edge of the ninth bit of the SCL clock. Read the U2RB register in the function header. When a NACK is detected during slave address transmission, set the no address match error flag to 1. At all other times, the master_trn_int function is called during master transmission and the master_rcv_int function is called during master reception. When communication is completed, generate a stop condition.
### Declaration

| static unsigned char master_trn_int (unsigned short rb_data) |

### Outline

Master transmit processing

### Argument

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

### Variable (global)

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

### Returned value

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

#### Function

Called from the UART2 transmit interrupt handling.

- IIC_SP_OFF is returned in the following case:
  - ACK is detected and not the last byte (starts the next transmission).
- IIC_SP_ON is returned in the following cases:
  - NACK is detected (NACK detect error flag is set to 1).
  - When the last byte transmission is completed.

### Declaration

| static unsigned char master_rcv_int (unsigned short rb_data) |

### Outline

Master receive processing

### Argument

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

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char iic_index</td>
<td>Number of transmit/receive bytes</td>
</tr>
<tr>
<td>unsigned char iic_length</td>
<td>Transfer data length</td>
</tr>
<tr>
<td>unsigned char far *iic_pointer</td>
<td>Transmit/receive buffer pointer</td>
</tr>
</tbody>
</table>

### Returned value

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

#### Function

Called from the UART2 transmit interrupt handling.

The argument value is stored in the receive buffer (except for the slave address data).

- NACK is set to the transmit register when the following data is the last byte. ACK is set to the transmit register when the following data is not the last byte. After setting ACK or NACK to the transmit register, the next receive operation starts.
- IIC_SP_OFF is returned in the following case:
  - The following data is not the last byte data.
- IIC_SP_ON is returned in the following case:
  - The last byte receive operation is completed.
### Declaration

```
unsigned char iic_master_end (void)
```

### Outline

Master control completed processing

### Argument

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

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(structure member) iic_status</td>
<td>All statuses</td>
</tr>
<tr>
<td>(structure member) iic_start</td>
<td>Mid-communication flag</td>
</tr>
<tr>
<td>(structure member) iic_err_par</td>
<td>Parameter error flag</td>
</tr>
<tr>
<td>(structure member) iic_err_nack</td>
<td>NACK detection error flag</td>
</tr>
<tr>
<td>(structure member) iic_err_addr</td>
<td>No address match error flag</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>BUSY</td>
<td>Mid-communication</td>
</tr>
<tr>
<td></td>
<td>RDY</td>
<td>Communication completed</td>
</tr>
<tr>
<td></td>
<td>ADDR_ERR</td>
<td>Address not matched</td>
</tr>
</tbody>
</table>

### Function

Called from the main function. Informs the user of the master control state completed. During communication, this function returns BUSY. When communication is completed, this function returns RDY. Additional processing after communication is completed can be added as needed.
4.3 Main Processing

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

Disabling interrupts.

PLL mode setting
SetPLLClock()

PLL mode setting \(^{(1)}\)
Set peripheral clock source to 25 MHz.

UART2 initial setting
uart2_init()

UART2 special mode 1 (I^2C mode) enabled

asm("fset I")

Enable interrupts.

R/W mode initial setting (master transmit)

ret ← BUSY

Initialize RAM for returned value of iic_master_end.

Initialize retry counter.

switch(ret)

= default (BUSY: mid-communication)
= ADDR_ERR (address not matched error)

retry_counter ← 0

Initialize retry counter.

= RDY (communication completed)

Increment retry counter

mode = WRITE?

= WRITE (master receive)

≠ WRITE (master transmit)

= WRITE (master transmit)

≠ WRITE (master receive)

mode = WRITE?

master control start processing
iic_master_start()

Master control start processing

iic_master_start()

= 0
(bus free or parameter error)

≠ 0
(bus free or parameter error)

= 0
(bus busy)

≠ 0
(bus busy)

Master control completed processing
iic_master_end()

= ADDR_ERR
(address not matched error)

= ADDR_ERR
(address not matched error)

switch(ret)

ret = BUSY?

Yes

No

Yes

No

Notes:
1. Refer to the hardware user's manual.
2. Additional processing can be added as needed.

```

Notes:
1. Refer to the hardware user's manual.
2. Additional processing can be added as needed.
4.4 UART2 Initial Setting

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

if ini = 1
  = 1 (I2C mode enabled)

uart2_init()

P7_1S ← 0x43
BCN2IC ← 0x00
S2TIC ← 0x00
U2C1 ← 0x00
U2MR ← 0x02
U2SMR2 ← 0x03
U2SMR3 ← 0xE0
U2SMR4 ← 0x70
U2C0 ← 0x90
U2SMR ← IIC_BRG
U2C1 ← 0x10
BCN2IC ← 0x00
S2TIC ← 0x00
P7_0S ← 0x43
P7_1S ← 0x43
PD_IIC ← PD_IIC & PD_IIC_INIT_INPUT
PD_IIC ← PD_IIC_INIT_INPUT
return
```

- Disable transmission/reception.
- Set the port direction: PD_IIC = PD_IIC_INIT
- Set SCL2 pin to N-channel open drain output.
- Set pins P7_0 (SDA2) and P7_1 (SCL2) to output mode.

- Select I2C mode.
- Disable start/stop condition generation interrupt.
- Disable UART2 transmit interrupt.
- Disable transmission/reception.
- Select I2C mode, select internal clock.
- Select UART2 transmit interrupt, enable clock synchronization.
- SDA2 digital delay: Seven or eight cycles of U2BRG count source
- Enable NACK output (release SDA2 pin), enable SCL output stop.
- U2BRG count source: f1
- Transfer format: MSB first
- Set 378 kbps transfer rate.
- Select transmission completed (TXEPT is 1) as UART2 transmit interrupt source.
- Set the IR bit to 0.
- Set SDA2 pin to N-channel open drain output.
- Set SCL2 pin to N-channel open drain output.
4.5 Master Control Start Processing

[Argument]
unsigned char addr: specify slave address
unsigned char rw: master transmit, master receive
unsigned char far *buf: transmit or receive buffer pointer
unsigned char len: transfer data length

iic_master_start()

iic_status ← 0x00
Clear all statuses.

(len = 0) || (rw > 1) || (addr > 0x7F)?
No parameter error

BBS = 1?
= 1 (bus busy)
≠ 1 (bus free)

iic_slave_addr ← addr << 1
Set slave address.

iic_slave_addr ← iic_slave_addr + rw
Set R/W.

(iic_length ← len
Set transfer data length.

iic_pointer ← buf
Set buffer address.

iic_start ← 1
Set mid-communication flag to 1.

BCN2IC ← 0x01
Enable start/stop condition generation interrupt.

U2SMR4 ← 0x070
Set STSPSEL bit to 0.

U2MR ← 0x02
Select I2C mode, select internal clock.

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

More than half an SCL clock cycle (40.0 ns) is needed to execute this instruction.

U2SMR2 ← 0x03

U2BRG ← IIC_BRG

U2SMR4 ← 0x71

U2SMR4 ← 0x09

Generate start condition.

U2BRG setting to target transfer rate.

return(RDY)
4.6 Start/Stop Condition Generation Interrupt Handling

```
_start_stop_condition_detection()

BBS = 1?
= 1 (stop condition detection)
= 1 (start condition detection)

Start condition detection processing
sta_int()

Stop condition detection processing
stp_int()

return
```

4.7 Start Condition Detection Processing

```
sta_int()

U2SMR3 ← 0xE2
Clock phase: With clock delay
Enable transmission/reception.

U2C1 ← 0x15
Do not generate start/stop conditions.

U2SMR4 ← 0x00
Clear ABT bit.

U2RB ← 0x0000
Set slave address.

temp.byte.byte0 ← iic_slave_addr
Set data to release SDA2 pin at the ninth bit.

U2TB ← temp.all
Start first byte (slave address) transmission.

BCN2IC ← 0x01
Set IR bit to 0 after changing CKPH bit.

S2TIC ← (S2TIC & 0x08) | 0x01
Enable UART2 transmit interrupt.

iic_index ← 0x00
Initialize number of transmit/receive bytes.

return
```
4.8 Stop Condition Detection Processing

```plaintext
stp_int()

U2C1 ← 0x10
  Disable transmission/reception.
U2MR ← 0x00
  Disable serial interface.
U2SMR3 ← 0xE0
  Clock phase: No clock delay
U2SMR4 ← 0x70
  Enable NACK output (release SDA2 pin), enable SCL output stop.
U2MR ← 0x02
  Select I2C mode, select internal clock.
S2TIC ← 0x00
  Disable UART2 transmit interrupt.
BCN2IC ← 0x00
  Disable start/stop condition generation interrupt.
iic_index ← 0x00
  Initialize number of transmit/receive bytes.
iic_start ← 0
  Set mid-communication flag to 0.
return
```
4.9 UART2 Transmit Interrupt Handling

```c
_uart2_trans()

temp.all ← U2RB  // Read from receive buffer register.

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

ACK detection in slave address or from the second byte on

iic_rw = 0?

= 0 (master transmit)
≠ 0 (master receive)

Master receive processing
master_rcv_int()
stop_req ← Returned value

Master transmit processing
master_trn_int()
stop_req ← Returned value

stop_req = IIC_SP_ON?

= IIC_SP_ON (stop condition generated)
≠ IIC_SP_ON (stop condition not generated)

U2SMR4 ← 0x04
U2SMR4 ← 0x3C

return

iic_err_addr ← 1  // Set address not matched error flag to 1.

stop_req ← IIC_SP_ON  // Generate stop condition.
```
4.10 Master Transmit Processing

```
master_trn_int() [Argument]
unsigned short rb_data: Data read from the U2RB register

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

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

iic_index < iic_length?
≥ iic_length (last byte)
< iic_length (not last byte)

iic_err_nack ← 1 Set NACK detection error flag to 1.

master_trn_int()

return(IIC_SP_OFF)

return(IIC_SP_ON)
```
### 4.11 Master Receive Processing

**[Argument]**

- `master_rcv_int()`
- `unsigned short rb_data`: Data read from the U2RB register

1. **iic_index = 0x00?**
   - `iic_index = 0x00? ≠ 0x00 (from the second byte on)`
   - `*iic_pointer ← (unsigned char)rb_data`
     - Store data read from the receive buffer register to the receive buffer.
     - `iic_pointer++`
     - Pointer to the receive buffer + 1

2. **iic_index ≥ iic_length?**
   - `≥ iic_length (last byte)`
   - `return(IIC_SP_ON)`
   - `U2TB ← 0x01FF`
     - Set NACK for next byte (prepare for next receive).
   - `iic_index ≥ iic_length?`<br>`< iic_length (next received data is not the last byte)`
     - `iic_pointer++`
     - `iic_index ≥ iic_length?`
       - `< iic_length (not last byte)`
       - `U2TB ← 0x01FF`
         - Set NACK for next byte (prepare for next receive).
     - `≥ iic_length (not last byte)`
       - `U2TB ← 0x00FF`
         - Set ACK for next byte (prepare for next receive).

3. **return(IIC_SP_OFF)**

### 4.12 Master Control Completed Processing

**iic_master_end()**

1. **iic_start = 1?**
   - `= 1 (mid-communication)`
   - `= 1 (communication completed)`
   - `(iic_status & 0x0E) = 0x00 (error)`
   - `= 0x00 (normal end)`
     - `iic_addr_err = 1?`
       - `Yes`
         - `return(ADDR_ERR)`
       - `No`
         - `return(RDY)`

2. **See Note 1.**
   - `return(RDY)`

3. **See Note 2.**
   - `return(ADDR_ERR)`

**Notes:**

1. Additional processing of communication completed normally can be added as needed.
2. Additional processing of communication completed with error can be added as needed.
5. Sample Program
A sample program can be downloaded from the Renesas Electronics website.

6. Reference Documents
R32C/118 Group User’s Manual: Hardware Rev.1.00
The latest version can be downloaded from the Renesas Electronics website.

Technical News/Technical Update
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 31, 2010</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of MPU/MCU Products

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

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

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

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

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

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

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

2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

3. You should neither modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.

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

5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

6. 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.

7. 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.

8. 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.

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 possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, the control and malfunction prevention, appropriate treatment for aging-degradation or any other appropriate measures. Because the evaluation of reconfigurable 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 if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

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.

(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.