Contents

1. Abstract ............................................................................................................................ 3

2. Introduction ...................................................................................................................... 3

3. Application Example ......................................................................................................... 4
   3.1 Program Outline ............................................................................................................... 4
      3.1.1 Peripheral Functions ................................................................................................. 10
      3.1.2 Notes on Using the Attached Sample Program ......................................................... 10
   3.2 Memory .......................................................................................................................... 10

4. Software ......................................................................................................................... 11
   4.1 Usage Variables ............................................................................................................. 11
   4.2 Function Tables ............................................................................................................. 13
   4.3 Main Processing ............................................................................................................. 23
   4.4 System Clock Setting ................................................................................................... 25
   4.5 Initial Setting of I/O Ports ............................................................................................ 26
   4.6 Initial Setting of I2C Bus Interface ................................................................................. 27
   4.7 Initial Setting of Timer RA .......................................................................................... 28
   4.8 Initializing Color Sensor .............................................................................................. 29
   4.9 Copying Communication Parameters ......................................................................... 30
   4.10 Starting I2C Communication ...................................................................................... 30
   4.11 Issuing Start Condition ............................................................................................... 32
   4.12 Issuing Stop Condition ............................................................................................... 32
   4.13 Resetting Color Sensor ............................................................................................. 33
   4.14 Waiting for Light Intensity Measurement Completion .............................................. 35
   4.15 Starting Timer RA Operation ..................................................................................... 35
   4.16 Reading Measurement Data ....................................................................................... 36
   4.17 Processing Measurement Results ............................................................................... 38
   4.18 Waiting for Communication Restart ........................................................................ 39
   4.19 Processing I2C Communication Error ...................................................................... 39
   4.20 I2C bus Interface Interrupt Handling ........................................................................ 40
   4.21 Data Transmission ..................................................................................................... 41
   4.22 Continuous Data Transmission .................................................................................. 41
   4.23 Reception Settings ...................................................................................................... 42
   4.24 Data Reception ........................................................................................................... 43
   4.25 Communication Error End Processing ..................................................................... 44
   4.26 Timer RA Interrupt Handling ...................................................................................... 44

5. Sample Program ............................................................................................................. 45
6. Reference Documents

Reference Documents

...
1. Abstract

This document describes the I\(^2\)C bus single master control program (transmit/receive processes) that can issue the restart condition by using the R8C/35C Group I\(^2\)C bus Interface.

The sample code given here controls the digital color sensor (S11059-02 DT) supporting the I\(^2\)C bus Interface, for an example.

2. Introduction

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

- MCU : R8C/35C Group
- High-Speed On-Chip Oscillator Clock : 20MHz

This application note can be used with other R8C Family MCUs which have the same special function registers (SFRs) as the above group. Check the 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

To conform to the communication format of the digital color sensor (S11059-02DT), a slave address and 8-byte data are sent in master transmission, and a slave address is sent and then 8-byte data is received in master reception. Master transmission and master reception are repeated alternately.

The sample code given here supports issuance of the restart condition. In master transmission, after fifth byte data is transmitted, the restart condition is issued without issuing the stop condition, and then the sixth byte is transmitted. After eighth byte data is transmitted, the restart condition is issued without issuing the stop condition to switch to the master reception mode.

Issuing the restart condition enables data transmission conforming to the communication format of the connection destination device, and also switching of data transmission direction to allow data reception from the connection destination device after the data transmission.

This transmission procedure conforms to the I²C bus communication protocol when used under the following conditions:

- Slave address : 7 bits
- Transfer rate : Approximately 357 kHz (Standard-mode and Fast-mode supported)
- Transfer data length : 1 to 255 bytes (not including the slave address)
- Single master communication (multimaster is not supported)
Figure 3.1 shows the Communication Format, Figure 3.2 shows the Block Diagram, Figure 3.3 to Figure 3.4 show Outline Flowcharts, and Figure 3.5 to Figure 3.7 show Timing Diagrams.

---

Master transmission

<table>
<thead>
<tr>
<th>ST</th>
<th>SLAVE ADDRESS</th>
<th>W</th>
<th>ACK</th>
<th>Data0 (11100100b)</th>
<th>ACK</th>
<th>Data1 (00001100b)</th>
<th>ACK</th>
<th>Data2 (00110000b)</th>
<th>ACK</th>
<th>S</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS</td>
<td>SLAVE ADDRESS</td>
<td>W</td>
<td>ACK</td>
<td>Data3 (00000000b)</td>
<td>ACK</td>
<td>Data4 (10000100b)</td>
<td>ACK</td>
<td></td>
<td></td>
<td>S</td>
<td>P</td>
</tr>
<tr>
<td>ST</td>
<td>SLAVE ADDRESS</td>
<td>W</td>
<td>ACK</td>
<td>Data5 (00000000b)</td>
<td>ACK</td>
<td>Data6 (00000100b)</td>
<td>ACK</td>
<td></td>
<td></td>
<td>S</td>
<td>P</td>
</tr>
</tbody>
</table>

Master reception

<table>
<thead>
<tr>
<th>RS</th>
<th>SLAVE ADDRESS</th>
<th>R</th>
<th>ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data0</td>
<td>ACK</td>
<td>Data1</td>
<td>ACK</td>
</tr>
<tr>
<td>Data4</td>
<td>ACK</td>
<td>Data5</td>
<td>ACK</td>
</tr>
</tbody>
</table>

- ST : Start condition
- RS : Restart condition
- SP : Stop condition
- W : Write mode is “0”
- R : Read mode is “1”
- ACK : Acknowledge is “0”
- NACK : No Acknowledge is “1”

Figure 3.1  Communication Format
The numbers in Figure 3.3 and Figure 3.4 correspond to the numbers indicated in the program processing in the operating timing charts in Figure 3.5 to Figure 3.7.
To release color sensor reset, issue restart and transmit 2-byte data.

Wait a certain period of time before starting next communication.

Select address of structure where information of transmit data is stored.

During transmission

2-byte data transmission completed?

Transmission completed

Generate stop condition

Wait

Set transmit data

Select address of structure where information of transmit data is stored.

During transmission

1-byte data transmission completed?

Transmission completed

Start master control (master transmit)

Start master control (master receive)

Perform processing after transmission/reception completion

Wait

8-byte data reception completed?

Reception completed

Transmit 1-byte data (read address in the sensor) to receive measurement data from color sensor. (Stop condition is not issued on completion of transmission.)

To receive measurement data from color sensor, issue restart condition and make a transition to read mode. Subsequently, receive 8-byte data. (After reception of last byte, stop condition is issued in I²C bus interface interrupt processing)

Wait until color sensor completes measurement of light intensity.

Figure 3.4 Outline Flowchart (2/2)
A process outline is described as follows:

1. Initial setting
   Initialize the system clock, I^2^C bus Interface associated SFRs, and variables used.

2. Start master control
   Generate a start condition.
   Enable the I^2^C bus Interface interrupt (transmit end interrupt request) and transmit the slave address.

3. I^2^C bus Interface interrupt (transmit end interrupt request)
   An interrupt is generated at the rising edge of the ninth bit of the SCL clock.
   - At master transmit
     Determine ACK/NACK and set the next byte transmit data.
   - At master receive
     Disable the transmit end interrupt request and enable the receive data full interrupt request.

4. I^2^C bus Interface interrupt (receive data full interrupt request)
   An interrupt is generated at the rising edge of the ninth bit of the SCL clock at master receive.
   Set the next byte ACK/NACK and read the receive data.
   To end communication, issue the stop condition and disable the receive data full interrupt request.

---

**Figure 3.5  Master Transmit Timing**

- **SDA (master output)**
- **SDA (slave output)**
- **SCL (master output)**
- **TEND bit in the ICSR register**
- **STOP bit in the ICSR register**
- **Program processing**
- **Start condition**
- **Stop condition**
- **Slave address**
- **I^2^C bus Interface interrupt (transmit end interrupt request)**
- **ACK**
- **Set to "0" by a program.**
- **Becomes "0" when writing data to the ICDRT register.**

---

R8C/35C Group  
I^2^C Communication for Issuing Restart Condition
R8C/35C Group

I2C Communication for Issuing Restart Condition

Figure 3.6 Master Receive Timing (1/2)

Figure 3.7 Master Receive Timing (2/2)
3.1.1 Peripheral Functions
The I2C bus Interface mode of the I2C bus Interface is used under the following setting conditions:

- I2C bus format is used.
- f1/56 is used for the transfer clock (approximately 357 kHz is set as the transfer rate).
- No wait states are set (data and the acknowledge bit are transferred consecutively).
- MSB first is used for the transfer format.
- 3 × f1 cycles are used for the SDA digital delay value.
- The receive acknowledge bit (ACKBR bit) is used to determine an acknowledge signal.
- The receive data full interrupt request is used.
- The transmit end interrupt request is used.
- The stop condition detection interrupt request is not used.
- The transmit data empty interrupt request is not used.
- The NACK receive interrupt request and arbitration lost/overrun error interrupt request are not used.

Calculating the transfer rate
Transfer rate = Bits CKS3 to CKS0 in the ICCR1 register setting
= 20 MHz (f1) / 56
≈ 357.142 kHz

Table 3.1 Pins Used and Their Functions

<table>
<thead>
<tr>
<th>Pin</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3_5/SCL</td>
<td>I/O</td>
<td>I2C bus clock I/O pin</td>
</tr>
<tr>
<td>P3_7/SDA</td>
<td>I/O</td>
<td>I2C bus 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 20 MHz High-Speed On-Chip Oscillator Clock, change the setting value of bits CKS3 to CKS0 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>2146 bytes</td>
<td>—</td>
</tr>
<tr>
<td>RAM</td>
<td>317 bytes</td>
<td>—</td>
</tr>
<tr>
<td>Maximum user stack</td>
<td>37 bytes</td>
<td>—</td>
</tr>
<tr>
<td>Maximum interrupt stack</td>
<td>25 bytes</td>
<td>—</td>
</tr>
</tbody>
</table>

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

C compiler : M16C Series, R8C Family Compiler V.5.45 Release 01
Compile options : -c -finfo -dir "$(CONFIGDIR)" -R8C
4. Software

This section shows the program example to set the example described in section 3, Application Example. Refer to the latest "R8C/35C Group User’s Manual Hardware" for details on individual registers.

4.1 Usage Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iic_stat_fcb iic_comstat</td>
<td>5 bytes in total</td>
<td>Structure for storing communication status</td>
</tr>
</tbody>
</table>

**Structure members**

- **iic_status**
  - 1 byte
  - Communication status
    - **b0**: Communication direction flag
      - 0: Master transmission
      - 1: Master reception
    - **b1**: Address mismatch flag
      - 0: No address mismatch
      - 1: Address mismatch
    - **b2 and b3**: Only used on command error occurrence (undefined)
    - **b4**: Error flag
      - 0: Normal completion
      - 1: Error
    - **b5**: Not used (undefined)
    - **b6**: Only used for command request (undefined)
    - **b7**: Communication in progress flag
      - 0: Communication completed
      - 1: Communication in progress

- **iic_slave_addr**
  - 1 byte
  - Slave address (7-bit address, value of b0 is "0")

- **iic_data_addr**
  - 2 bytes
  - Transmit/receive buffer address

- **iic_num_byte**
  - 1 byte
  - Number of transmit/receive data (in bytes)

**Functional Description of Structure**

Acquires values set to structures iic_set_str1 to iic_set_str5 (see Table 4.3) and uses the values for communication.

**Functional Description of Structure Member iic_status**

<table>
<thead>
<tr>
<th>&lt;Commands&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40 (CWRITEMODE)</td>
<td>Master transmission command</td>
</tr>
<tr>
<td>0x41 (CREADMODE)</td>
<td>Master reception command</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;Status&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80 (CINWRITE)</td>
<td>Transmission is in progress</td>
</tr>
<tr>
<td>0x81 (CINREAD)</td>
<td>Reception is in progress</td>
</tr>
<tr>
<td>0x00 (SUCCESSW)</td>
<td>Transmission has completed normally</td>
</tr>
<tr>
<td>0x01 (SUCCESWR)</td>
<td>Reception has completed normally</td>
</tr>
<tr>
<td>0x11 (CBUSBUSY)</td>
<td>I2C bus error</td>
</tr>
<tr>
<td>0x12 (CSLADEVBUSY)</td>
<td>Slave busy (NACK in address)</td>
</tr>
<tr>
<td>0x13 (CNOACK)</td>
<td>Data transmission error (NACK in data)</td>
</tr>
<tr>
<td>0x1f (CCOMERROR)</td>
<td>Command error</td>
</tr>
</tbody>
</table>
### Table 4.2  Definition File Name: r01an3378_src.c (2/4)

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static unsigned char iic_tx1[TX1_BUFSIZE]</td>
<td>3 bytes</td>
<td>Transmit buffer 1</td>
</tr>
<tr>
<td>static unsigned char iic_tx2[TX2_BUFSIZE]</td>
<td>2 bytes</td>
<td>Transmit buffer 2</td>
</tr>
<tr>
<td>static unsigned char iic_tx3[TX3_BUFSIZE]</td>
<td>2 bytes</td>
<td>Transmit buffer 3</td>
</tr>
<tr>
<td>static unsigned char iic_tx4[TX4_BUFSIZE]</td>
<td>1 byte</td>
<td>Transmit buffer 4</td>
</tr>
<tr>
<td>static unsigned char iic_rx1[RX1_BUFSIZE]</td>
<td>8 bytes</td>
<td>Receive buffer 1</td>
</tr>
</tbody>
</table>

### Table 4.3  Definition File Name: r01an3378_src.c (3/4)

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static iic_set_fcb iic_set_str1, iic_set_str2, iic_set_str3, iic_set_str4, iic_set_str5</td>
<td>5 bytes in total</td>
<td>Structure for storing communication setting parameters</td>
</tr>
<tr>
<td>Structure members</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsigned char iic_str_command</td>
<td>1 byte</td>
<td>Transmission/reception commands</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x40 (CWRITEMODE) : Master transmission command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x41 (CREADMODE) : Master reception command</td>
</tr>
<tr>
<td>unsigned char iic_str_slave_addr</td>
<td>1 byte</td>
<td>Slave address (7-bit address, value of b0 is &quot;0&quot;)</td>
</tr>
<tr>
<td>unsigned char far *iic_str_data_addr</td>
<td>2 bytes</td>
<td>Transmit/receive buffer address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(See Table 4.2 for transmit/receive buffer.)</td>
</tr>
<tr>
<td>unsigned char iic_str_num_byte</td>
<td>1 byte</td>
<td>Number of transmit/receive data (in bytes)</td>
</tr>
</tbody>
</table>

### Table 4.4  Definition File Name: r01an3378_src.c (4/4)

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static unsigned int iic_red_data</td>
<td>2 bytes</td>
<td>Red light data</td>
</tr>
<tr>
<td>static unsigned int iic_green_data</td>
<td>2 bytes</td>
<td>Green light data</td>
</tr>
<tr>
<td>static unsigned int iic_blue_data</td>
<td>2 bytes</td>
<td>Blue light data</td>
</tr>
<tr>
<td>static unsigned int iic_ir_data</td>
<td>2 bytes</td>
<td>Infrared light data</td>
</tr>
</tbody>
</table>

### Table 4.5  Definition File Name: iic.c

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static unsigned char iic_status_buf</td>
<td>1 byte</td>
<td>Status buffer</td>
</tr>
<tr>
<td>static unsigned char far *iic_data_pointer</td>
<td>2 bytes</td>
<td>Pointer to transmit/receive buffer</td>
</tr>
<tr>
<td>static unsigned char iic_num_byte_buf</td>
<td>1 byte</td>
<td>Transmit/receive data count buffer</td>
</tr>
<tr>
<td>static unsigned char iic_start_cond_flag</td>
<td>1 byte</td>
<td>Start condition flag</td>
</tr>
</tbody>
</table>

### Table 4.6  Definition File Name: tra.c

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static unsigned int tra_wait_dwncnt</td>
<td>2 bytes</td>
<td>Variable for timer counting</td>
</tr>
</tbody>
</table>
### 4.2 Function Tables

#### Declaration
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>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
After the initial setting of the system clock, I²C bus Interface, and timer RA, executes master transmission and reception. After master transmission and reception are completed, resumes communication after inserting a given length of wait time, and repeats master transmission and reception.

---

#### Declaration
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
This function is called from the Main Processing. Sets the system clock (High-Speed On-Chip Oscillator Clock).

---

#### Declaration
void port_init (void)

#### Outline
Initial Setting of I/O Ports

#### 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
This function is called from the Main Processing. Initializes I/O Ports to use I²C bus Interface.

---

#### Declaration
void iic_init (void)

#### Outline
Initial Setting of I²C Bus Interface

#### 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
This function is called from the Main Processing. Initializes SFRs to use I²C bus Interface.
### Declaration
```
void tra_init (void)
```

### Outline
Initial Setting of Timer RA

### 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
This function is called from the Main Processing. Initializes SFRs to use Timer RA.

---

### Declaration
```
unsigned char iic_sensor_init (unsigned char far *iic_set_str)
```

### Outline
Initializing Color Sensor

### Argument
<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char far *iic_set_str</td>
<td>Pointer to structure for storing communication setting parameters</td>
</tr>
</tbody>
</table>

### Variable (global)
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(structure) iic_comstat</td>
<td>Communication status</td>
</tr>
</tbody>
</table>

### Returned value
<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>0</td>
<td>Communication error</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Successful communication</td>
</tr>
</tbody>
</table>

### Function
This function is called from the Main Processing. Performs processing to initialize the color sensor. Calls the iic_copy_parameter function to acquire the transmit data, and calls the iic_start function to start communication. During communication, checks the communication status of the structure iic_comstat and waits for completion of data transmission. To end communication, calls the iic_stop_cond function to perform the communication end processing. If the status checking shows a communication error, calls the iic_stop_cond function to stop the communication.

---

### Declaration
```
static void iic_copy_parameter (unsigned char far *iic_set_str)
```

### Outline
Copying Communication Parameters

### Argument
<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char far *iic_set_str</td>
<td>Pointer to structure for storing communication setting parameters</td>
</tr>
</tbody>
</table>

### Variable (global)
<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(structure) iic_comstat</td>
<td>Communication status</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
This function is called before starting communication. Copies the communication data to (structure) iic_comstat to acquire the parameters required for communication.
### Declaration

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>iic_start (void)</td>
<td></td>
</tr>
</tbody>
</table>

### Outline

#### Starting I2C Communication

#### 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) iic_comstat</td>
<td>Communication status</td>
</tr>
<tr>
<td>static unsigned char iic_status_buf</td>
<td>Buffer for storing communication status during communication</td>
</tr>
<tr>
<td>static unsigned char far *iic_data_pointer</td>
<td>Pointer to indicate transmit/receive data during communication</td>
</tr>
<tr>
<td>static unsigned char iic_num_byte_buf</td>
<td>Buffer for indicating number of remaining data during communication</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

This function is called to start communication. Performs processing to start communication. After calling the iic_start_cond function to issue the start condition, transmits the slave address. In the function header, confirms that the bus is not being used by any external device and checks the transmit/receive commands and variable iic_num_byte_buf. If the checking shows an error, does not perform the communication start processing.

### Declaration

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>static void</td>
<td>iic_start_cond (void)</td>
<td></td>
</tr>
</tbody>
</table>

### Outline

#### Issuing Start Condition

#### 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>static unsigned char iic_start_cond_flag</td>
<td>Start condition 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

This function is called from the Starting I2C Communication processing. Issues the start condition and sets the start condition flag.
### Declaration

static void iic_stop_cond (void)

### Outline

Issuing Stop Condition

### 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

This function is called to stop communication. Performs processing to issue the stop condition.
- In the function header, checks the bus status.
  - When the bus is busy
    Performs processing to issue the stop condition.
  - When the bus is free
    Does not perform processing to issue the stop condition.

### Declaration

unsigned char iic_sensor_reset (unsigned char far *iic_set_str,
                                 unsigned char far *iic_set_str_2nd)

### Outline

Resetting Color Sensor

### Argument

<table>
<thead>
<tr>
<th>Argument name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char far *iic_set_str</td>
<td>Pointer to structure for storing communication setting parameters</td>
</tr>
<tr>
<td>unsigned char far *iic_set_str_2nd</td>
<td>Pointer to structure for storing communication setting parameters</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(structure) iic_comstat</td>
<td>Communication status</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>0</td>
<td>Communication error</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Successful communication</td>
</tr>
</tbody>
</table>

### Function

This function is called from the Main Processing. Performs processing to reset the color sensor.
- Calls the iic_copy_parameter function to acquire the transmit data, and calls the iic_start function to start communication. During communication, checks the communication status of the structure iic_comstat and waits for completion of data transmission. To end communication, calls the iic_stop_cond function to perform the communication end processing.
- If the status checking shows a communication error, calls the iic_stop_cond function to stop the communication.
### declaration
```c
void tra_measure_wait (void)
```

### Outline
- **Waiting for Light Intensity Measurement Completion**

### 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>static unsigned int tra_wait_dwncnt</td>
<td>Timer count value</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
This function is called from the Main Processing. Performs processing to wait for the color sensor to complete measurement of the light intensity. Sets the timer count to the variable `tra_wait_dwncnt` and calls the `tra_start` function to start the timer RA. While the timer RA is running, checks the timer RA count status flag (TCSTF) and waits for the timer RA to stop. The variable `tra_wait_dwncnt` is updated and timer RA is stopped in the timer RA interrupt handling.

### declaration
```c
static void tra_start (void)
```

### Outline
- **Starting Timer RA Operation**

### 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
This function is called to run the timer RA. Performs processing to run the timer RA.
### Declaration

```c
unsigned char iic_sensor_read (unsigned char far *iic_set_str,
    unsigned char far *iic_set_str_2nd)
```

### Outline

**Reading Measurement Data**

### Argument

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char far *iic_set_str</td>
<td>Pointer to structure for storing communication setting parameters</td>
</tr>
<tr>
<td>unsigned char far *iic_set_str_2nd</td>
<td>Pointer to structure for storing communication setting parameters</td>
</tr>
</tbody>
</table>

### Variable (global)

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(structure) iic_comstat</td>
<td>Communication status</td>
</tr>
</tbody>
</table>

### Returned value

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char</td>
<td>0</td>
<td>Communication error</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Successful communication</td>
</tr>
</tbody>
</table>

### Function

This function is called from the Main Processing. Performs processing to read out the measurement data of the color sensor.

1. Calls the `iic_copy_parameter` function to acquire the transmit data, and calls the `iic_start` function to start communication.
2. During communication, checks the communication status of the structure `iic_comstat` and waits for completion of data transmission/reception. After receiving the last byte, issues the stop condition in the I2C bus Interface Interrupt Handling to end communication.
3. If the status checking shows a communication error, calls the `iic_stop_cond` function to stop the communication.

---

### Declaration

```c
void iic_measure_result (unsigned char far *iic_rx,
    unsigned char iic_rx_size,
    unsigned int far *iic_red_data,
    unsigned int far *iic_green_data,
    unsigned int far *iic_blue_data,
    unsigned int far *iic_ir_data)
```

### Outline

**Processing Measurement Results**

### Argument

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned char far *iic_rx</td>
<td>Pointer to receive buffer</td>
</tr>
<tr>
<td>unsigned char iic_rx_size</td>
<td>Size of receive buffer</td>
</tr>
<tr>
<td>unsigned int far *iic_red_data</td>
<td>Pointer to variable for storing measurement data of red light</td>
</tr>
<tr>
<td>unsigned int far *iic_green_data</td>
<td>Pointer to variable for storing measurement data of green light</td>
</tr>
<tr>
<td>unsigned int far *iic_blue_data</td>
<td>Pointer to variable for storing measurement data of blue light</td>
</tr>
<tr>
<td>unsigned int far *iic_ir_data</td>
<td>Pointer to variable for storing measurement data of infrared light</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

This function is called from the Main Processing. Performs processing to store the receive data in the receive buffer into the variables indicating the measurement data of each color light.
### Declaration
void tra_next_com_wait (void)

### Outline
Waiting for Communication Restart

### Argument
- **Argument name**: None
- **Meaning**: __

### Variable (global)
- **Variable name**: tra_wait_dwncnt
- **Contents**: Timer count value

### Returned value
- **Type**: None
- **Value**: __
- **Meaning**: __

### Function
This function is called from the Main Processing. Performs processing to wait for the next communication to be resumed.

Sets the timer count to the variable `tra_wait_dwncnt` and calls the `tra_start` function to start the timer RA. While the timer RA is running, checks the timer RA count status flag (TCSTF) and waits for the timer RA to stop.

The variable `tra_wait_dwncnt` is updated and timer RA is stopped in the timer RA interrupt handling.

### Declaration
void iic_error (void)

### Outline
Processing I2C Communication Error

### Argument
- **Argument name**: None
- **Meaning**: __

### Variable (global)
- **Variable name**: iic_status_buf
- **Contents**: Buffer for storing communication status during communication
- **Variable name**: iic_start_cond_flag
- **Contents**: Start condition flag

### Returned value
- **Type**: None
- **Value**: __
- **Meaning**: __

### Function
This function is called from the Main Processing when a communication error is detected. Ends without any processing. To process an error, add the appropriate program.

### Declaration
void iic_int (void)

### Outline
I2C bus Interface Interrupt Handling

### Argument
- **Argument name**: None
- **Meaning**: __

### Variable (global)
- **Variable name**: iic_comstat
- **Contents**: Communication status
- **Variable name**: iic_status_buf
- **Contents**: Buffer for storing communication status during communication
- **Variable name**: iic_start_cond_flag
- **Contents**: Start condition flag

### Returned value
- **Type**: None
- **Value**: __
- **Meaning**: __

### Function
An interrupt is generated at the rising edge of the 9th bit of the SCL clock.

In the function header, checks the variable `iic_status_buf` to see if the start condition has been issued immediately before data transmission.

When the start condition has been issued, calls the `iic_data_trs_int` function in transmit mode, and calls the `iic_set_rcv_int` function in receive mode.

When the start condition has not been issued, calls the `iic_cont_data_trs_int` function in transmit mode, and calls the `iic_data_rcv_int` function in receive mode.

When a communication error is detected, calls the `iic_error_exit_int` function to end communication.
### Declaration

```c
static void iic_data_trs_int (void)
```

#### Outline
Data Transmission

#### 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>static unsigned char far *iic_data_pointer</td>
<td>Pointer to indicate transmit/receive data during communication</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
This function is called from the I2C bus Interface Interrupt Handling. Transmits the data in the transmit buffer pointed to by the pointer iic_data_pointer. Also updates the pointer iic_data_pointer to perform the next transmission.

---

### Declaration

```c
static void iic_cont_data_trs_int (void)
```

#### Outline
Continuous Data Transmission

#### 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) iic_comstat</td>
<td>Communication status</td>
</tr>
<tr>
<td>static unsigned char iic_status_buf</td>
<td>Buffer for storing communication status during communication</td>
</tr>
<tr>
<td>static unsigned char iic_num_byte_buf</td>
<td>Buffer for indicating number of remaining data during communication</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
This function is called from the I2C bus Interface Interrupt Handling. In the function header, checks if ACK has been detected.

- **ACK detected**
  - After updating the variable iic_num_byte_buf, checks the number of remaining transmit data bytes. When there is any data to be transmitted, calls the iic_data_trs_int function to perform the data transmit processing. When there is no data to be transmitted, updates the communication status of the structure iic_comstat to indicate the end of communication.

- **NACK detected**
  - Calls the iic_error_exit_int function to end the communication.
### Declaration

```c
static void iic_set_rcv_int (void)
```

### Outline

**Reception Settings**

<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></td>
<td>static unsigned char iic_num_byte_buf</td>
<td>Buffer for indicating number of remaining data during communication</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

This function is called from the I2C bus Interface Interrupt Handling. Switches from master transmit mode to master receive mode and enables the receive data full interrupt request.

In the function, checks the variable `iic_num_byte_buf` to see the number of receive data bytes to be received.

- When the number of data bytes to be received is 1
  Sets so that "1" (NACK) is transmitted at the acknowledge timing and disables reception following the next data reception.

- When the number of data bytes to be received is 2 or more
  Sets so that "0" (ACK) is transmitted at the acknowledge timing and enables reception following the next data reception.

---

### Declaration

```c
static void iic_data_rcv_int (void)
```

### Outline

**Data Reception**

<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></td>
<td>(structure) iic_comstat</td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td>static unsigned char *iic_data_pointer</td>
<td>Pointer to indicate transmit/receive data during communication</td>
</tr>
<tr>
<td></td>
<td>static unsigned char iic_num_byte_buf</td>
<td>Buffer for indicating number of remaining data during communication</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

This function is called from the I2C bus Interface Interrupt Handling. Stores the receive data into the receive buffer area pointed to by the pointer `iic_data_pointer` and updates the pointer `iic_data_pointer` to perform the next reception.

Also updates the variable `iic_num_byte_buf` and checks the number of remaining receive data bytes.

- When the number of data bytes to be received is 0
  To end communication, issues the stop condition, disables the receive data full interrupt request, and updates the communication status of the structure `iic_comstat`.

- When the number of data bytes to be received is 1
  Sets so that "1" (NACK) is transmitted at the acknowledge timing and disables reception following the next data reception.
<table>
<thead>
<tr>
<th>Declaration</th>
<th>static void iic_error_exit_int (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Communication Error End 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>(structure) iic_comstat</td>
</tr>
<tr>
<td></td>
<td>static unsigned char iic_status_buf</td>
</tr>
<tr>
<td></td>
<td>static unsigned char iic_num_byte_buf</td>
</tr>
<tr>
<td>Returned value</td>
<td>Type</td>
</tr>
<tr>
<td></td>
<td>None</td>
</tr>
<tr>
<td>Function</td>
<td>This function is called if a communication error is detected during execution of the I2C bus Interface Interrupt Handling. To end the communication, updates the communication status of the structure iic_comstat and the number of transmit/receive data bytes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Declaration</th>
<th>void tra_int (void)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outline</td>
<td>Timer RA 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>static unsigned int tra_wait_dwncnt</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 due to a timer RA underflow at 25-ms intervals. This function decrements the variable tra_wait_dwncnt and checks the result. When the variable tra_wait_dwncnt is &quot;0&quot;, stops Timer RA counting operation.</td>
</tr>
</tbody>
</table>
4.3 Main Processing

```
main()
asm("FCLR I")

System clock setting
mcu_init()
(System clock setting
(High-Speed On-Chip Oscillator Clock setting)).

Initial setting of I/O Ports
port_init()
(Initial setting of I/O Ports).

Initial setting of I²C bus interface
iic_init()
(Initial setting of I²C bus interface associated SFRs).

Initial setting of Timer RA
tra_init()
(Initial setting of Timer RA associated SFRs).

Disable interrupts.

Initialize registers TRAPRE and TRA,
and bits TSTART and TCSTF in the TRACR register.

Disable the timer RA interrupt.
Clear the interrupt request bit.
```

Figure 4.1 Main Processing (1/2)
Initializing color sensor
\[ \text{iic_sensor_init()} \]

Resetting color sensor
\[ \text{iic_sensor_reset()} \]

Waiting for light intensity measurement completion
\[ \text{tra_measure_wait()} \]

Reading measurement data
\[ \text{iic_sensor_read()} \]

Processing measurement results
\[ \text{iic_measure_result()} \]

Waiting for communication restart
\[ \text{tra_next_com_wait()} \]

Master transmission and reception
\[ \text{teie_iicier} \leftarrow 1 \] (Enable transmit end interrupt request.)
\[ \text{iicic} \leftarrow 0x02 \] (Enable I2C bus interface interrupt.)
(Set priority level to "2").
\[ \text{asm("FSET I")} \] (Enable interrupts.)

\[ \text{bsy_iicr2} = 1 ? \]
\[ = 1 \text{ (bus busy)} \]
\[ = 0 \text{ (bus free)} \]
Wait for bus release.

\[ \text{iicr1} \leftarrow \text{iicr1} | 0x30 \]
Set to master transmit mode.

Master transmission
\[ \Rightarrow \]

\[ \text{Returned value} = 1 ? \]
\[ = 0 \text{ (communication error)} \]
\[ = 1 \text{ (successful communication)} \]

Master transmission
\[ \Rightarrow \]

\[ \text{Returned value} = 1 ? \]
\[ = 0 \text{ (communication error)} \]
\[ = 1 \text{ (successful communication)} \]

Wait for communication restart
\[ \Rightarrow \]

\[ \text{Processing I2C communication error} \]
\[ \text{iic_error()} \]

Wait until next communication starts.

**Figure 4.2** Main Processing (2/2)
4.4 System Clock Setting

```
mcu_init()
asm("FCLR I")
prc0 ← 1
Enable interrupts.

cm14 ← 0
Start Low-Speed On-Chip Oscillator.

fra2 ← 0x00
Select High-Speed On-Chip Oscillator Clock divide-by-2 mode.

fra00 ← 1
Start High-Speed On-Chip Oscillator.

Repeat
(i <= 255)

i++;
Wait until oscillation stabilizes.

fra01 ← 1
Select High-Speed On-Chip Oscillator.

cm16 ← 0
Select CPU clock no division.

cm17 ← 0

cm06 ← 0
Enable bits CM16 and CM17.

prc0 ← 0
Set system control register protect.

return
```

Figure 4.3 System Clock Setting
4.5 Initial Setting of I/O Ports

Figure 4.4 Initial setting of I/O Ports
4.6 Initial Setting of I2C Bus Interface

```c
iic_init()
asm("FCLR I")
pd3 ← pd3 & 0x5f
iicic ← 0x00
msiic ← 0
iicsel ← 1
icier ← 0x00
stop_icsr ← 0
iccr1 ← 0x88
iccr2 ← 0xf0
icmr ← 0x00
pmsr ← pmsr & 0x09
icsr ← icsr & 0x97
sar ← 0x00
```

- Disable interrupts.
- Set PD3_5 (SCL) to input mode. Set PD3_7 (SDA) to input mode.
- Disable I²C bus interface interrupt.
- Set I²C bus to active.
- Select I²C bus interface function.
- Disable stop condition detection interrupt request.
- Disable receive data full interrupt request.
- Disable transmit end interrupt request.
- Clear stop condition detection flag.
- Select f1/56 as transfer clock. Set to slave receive mode. Continue next receive operation. This module is enabled for transfer operations.
- Initialize I²C bus control register 2.
- Set 000b (9 bits) for bit counter. Set no wait states (data and the acknowledge bit are transferred consecutively). Use MSB first for data transfer.
- Select digital delay of 3 × f1 cycles.
- Set slave address.

Figure 4.5 Initial Setting of I²C Bus Interface
4.7 Initial Setting of Timer RA

```c
tra_init()
asm("FCLR I")
tstart_tracr ← 0
= 1 (during count)
= 0 (count stopped)
tcstf_tracr = 1 ?
traic ← 0x00
Disable timer RA interrupt.
trasr ← 0x00
The TRAIO pin is not used.
tstop_tracr ← 1
Initialize registers TRAPRE and TRA, and bits TSTART and TCSTF in the TRACR register.
trapre ← 250 - 1
Underflow period: Set to 25ms
[ {1 / (20MHz / 8)} × 250 × 250 = 25ms ]
tra ← 250 - 1
traioc ← 0x00
Set to "0x00" in timer mode.
tramr ← 0x10
Select timer mode.
f8 is selected as the timer RA count source.
Provide count source.
return
```

Figure 4.6 Initial setting of Timer RA
4.8 Initializing Color Sensor

```c
iic_sensor_init()
ret ← 0

Copying communication parameters
iic_copy_parameter()

Starting I2C communication
iic_start()

asm("FSET I")

≠ 0x00 (error)
= 0x00 (successful completion)

(iic_status & 0x80) ≠ 0x00 ?
= 0x00 (communication completed)

(iic_status & 0x10) ≠ 0x00 ?
= 0x00 (successful completion)

(iic_status & 0x10) = 0x00 ?

= 0x00 (successful completion)

Issuing stop condition
iic_stop_cond()

ret ← 1

return (ret)
```

Figure 4.7 Initializing Color Sensor
### 4.9 Copying Communication Parameters

- \texttt{iic\_copy\_parameter()}
- \texttt{iic\_status \leftarrow *(\texttt{iic\_set\_str})} \quad \text{Read command.}
- \texttt{iic\_slave\_addr \leftarrow *(\texttt{iic\_set\_str} + 1)} \quad \text{Read slave address.}
- \texttt{iic\_data\_addr \leftarrow \left(\left(\texttt{iic\_set\_str} + 3\right) \ll 8\right) + \left(\texttt{iic\_set\_str} + 2\right)} \quad \text{Read transmit/receive buffer address.}
- \texttt{iic\_num\_byte \leftarrow *(\texttt{iic\_set\_str} + 4)} \quad \text{Read transmit/receive data count.}

### 4.10 Starting \textit{i}^2\text{C} Communication

- \texttt{iic\_start()}
- \texttt{asm("FCLR I")} \quad \text{Disable interrupts.}
- \texttt{iic\_status\_buf \leftarrow \text{CBUSBUSY}} \quad \text{Set "CBUSBUSY" to status buffer.}
- \texttt{((\text{mst\_iccr1} = 0) \&\& (\text{bbsy\_iccr2} = 1)) = 0 ?} \quad \text{1 (bus occupied by external device.)}
- \texttt{= 0 \text{ (bus not occupied by external device.)}}
- \texttt{iic\_status\_buf \leftarrow \text{iic\_status}} \quad \text{Acquire command from FCB for storing communication status.}
- \texttt{iic\_status\_buf \leftarrow \text{iic\_status\_buf} \& \text{0xe}} \quad \text{Mask communication direction (transmission/reception) indication bit.}
- \texttt{\text{iic\_status\_buf} \neq \text{0x40} \text{ (invalid command)}}
- \texttt{\text{iic\_status\_buf} = \text{0x40} \text{ (valid command)}}
- \texttt{iic\_status\_buf \leftarrow \text{CCOMERROR}}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{diagram}
\caption{Starting \textit{i}^2\text{C} Communication (1/2)}
\end{figure}
Issuing start condition
iic_start_cond()

slave_addr_rw ← iic_status & 0x01

icdrt ← iic_slave_addr | slave_addr_rw

iic_data_pointer ← iic_data_addr

iic_num_byte_buf ← iic_num_byte

iic_status_buf ← iic_status + 0x40

Generate status from command.

iic_status_buf = CCOMERROR (error)

≠ CCOMERROR (no error)

iic_num_byte_buf = 0x00 ?

= 0x00 (no transmit/receive data remaining)

≠ 0x00 (transmit/receive data remaining)

iic_status_buf ← CCOMERROR

iic_status ← iic_status_buf

Update communication status.

return

Acquire transfer direction.

Transmit slave address with transfer direction.

Copy transmit/receive buffer address.

Copy transmit/receive data count.

Figure 4.10   Starting I^2C Communication (2/2)
4.11 Issuing Start Condition

```c
iic_start_cond()
```

Generate start condition.

```c
iccr2 ← 0xb0
```

Set start condition flag.

```c
iic_start_cond_flag ← 1
```

```c
return
```

Figure 4.11 Issuing Start Condition

4.12 Issuing Stop Condition

```c
iic_stop_cond()
```

```c
bbsy_iccr2 = 1 ?
```

= 0 (bus free)

= 1 (bus busy)

```c
stop_icr ← 0
```

Clear stop condition detection flag.

```c
iccr2 ← 0x30
```

Generate stop condition.

```c
stop_icr = 0 ?
```

= 0 (stop condition not detected)

= 1 (stop condition detected)

```c
stop_icr ← 0
```

Clear stop condition detection flag.

```c
return
```

Figure 4.12 Issuing Stop Condition
4.13 Resetting Color Sensor

Transmit 2-byte data to reset color sensor. (Stop condition is not issued on completion of transmission.)

Starting I2C communication
   iic_start()

Enable interrupts.
   asm("FSET I")

Set sensor reset data.
   iic_sensor_reset()

Wait for communication completion.

(iic_status & 0x010) ≠ 0x00 (error)
   = 0x00 (successful completion)

(iic_status & 0x080) ≠ 0x00 (during communication)
   = 0x00 (communication completed)

ret ← 0

Copying communication parameters
   iic_copy_parameter()
Issuing stop condition

\[ \text{iic\_stop\_cond()} \]

\[ \text{return (ret)} \]

Starting \(\text{I}^2\text{C}\) communication

\[ \text{iic\_start()} \]

\(\text{asm}(\text{"FSET I")} \)

\(\text{(iic\_status \& 0x10) \neq 0x00 \?} \)

\(\text{= 0x00 (successful completion)} \)

\(\neq 0x00 \text{ (error)} \)

\(\text{(iic\_status \& 0x80) \neq 0x00 \?} \)

\(\text{= 0x00 (communication completed)} \)

\(\neq 0x00 \text{ (during communication)} \)

\(\text{(iic\_status \& 0x10) = 0x00 \?} \)

\(\text{= 0x00 (successful completion)} \)

\(\neq 0x00 \text{ (error)} \)

\(\text{ret} \leftarrow 1 \)

Copying communication parameters

\[ \text{iic\_copy\_parameter()} \]

To release color sensor reset, issue restart condition and transmit 2-byte data

Figure 4.14   Resetting Color Sensor (2/2)
4.14 Waiting for Light Intensity Measurement Completion

```
tra_measure_wait()
tra_wait_dwncnt ← 100
Starting Timer RA operation
tra_start()
```

Set count value to the variable for timer counting.
Measurement time 2.5 s [25 ms × 100 = 2500 ms]

```
tcstf_tracr = 1 ?
= 1 (during count)
= 0 (count stopped)
traic ← 0x00
return
```

Wait until time measurement is complete and timer RA stops.
Disable timer RA interrupt.

---

4.15 Starting Timer RA Operation

```
tra_start()
stop_tracr ← 1
trapre ← 250 - 1
tra ← 250 - 1
traic ← 0x01
tstart_tracr ← 1
tcstf_tracr = 0 ?
= 0 (count stopped)
= 1 (during count)
return
```

Initialize registers TRAPRE and TRA, and bits TSTART and TCSTF in the TRACR register.

```
Underflow period: Set to 25ms
   = {1 / (20MHz / 8)} × 250 × 250 = 25ms
```

Enable the timer RA interrupt (Set interrupt priority level to "1").

Timer RA operation starts.
### 4.16 Reading Measurement Data

```c
iic_sensor_read()
ret ← 0

Copying communication parameters
iic_copy_parameter()

Starting I2C communication
iic_start()

asm("FSET I")
Enable interrupts.

(iic_status & 0x10)
≠ 0x00 (error)
= 0x00 (successful completion)

(iic_status & 0x80)
≠ 0x00 (during communication)
= 0x00 (communication completed)

To receive measurement data from color sensor,
transmit 1-byte data (read address in sensor).
(Stop condition is not issued on completion of transmission.)
```

Figure 4.17 Reading Measurement Data (1/2)
return (ret)

Starting I²C communication
iic_start()

asm("FSET I")

Enable interrupts.

(iic_status & 0x10) ≠ 0x00 (error)

(iic_status & 0x10) = 0x00 ?

= 0x00 (successful completion)

(iic_status & 0x80) ≠ 0x00 (during communication)

(iic_status & 0x80) = 0x00 ?

= 0x00 (communication completed)

(iic_status & 0x10) ≠ 0x00 (error)

(iic_status & 0x10) = 0x00 ?

= 0x00 (successful completion)

ret ← 1

ret = 0 ?

= 0 (communication error)

= 1 (successful communication)

Issuing stop condition
iic_stop_cond()

return (ret)

To receive measurement data from color sensor, issue restart condition and make a transition to read mode. Subsequently, receive 8-byte data. (After reception of last byte, stop condition is issued in I²C bus interface interrupt handling.)

Figure 4.18   Reading Measurement Data (2/2)
4.17 Processing Measurement Results

```c
iic_measure_result()

i ← 0

iic_rx_buf[i] ← *(iic_rx + (i + 1))
i++
iic_rx_buf[i] ← *(iic_rx + (i - 1))
i++

Yes (i < iic_rx_size)
iic_result_pointer ← iic_rx_buf
*iic_red_data ← *iic_result_pointer
*iic_green_data ← *(iic_result_pointer + 1)
*iic_blue_data ← *(iic_result_pointer + 2)
*iic_ir_data ← *(iic_result_pointer + 3)

No (i >= iic_rx_size)
i < iic_rx_size ?
iic_rx_buf[i] ← *(iic_rx + (i + 1))
iic_rx_buf[i] ← *(iic_rx + (i + 1))
i++
i++
i < iic_rx_size ?

return
```

Rearrange receive data of each color stored in receive buffer so that lower byte precedes upper byte, and store them in array iic_rx_buf[].

Variable iic_rx_size: Indicates receive data count

Set address of array iic_rx_buf[] to pointer iic_result_pointer.

Store red light measurement data in variable iic_red_data.

Store green light measurement data in variable iic_green_data.

Store blue light measurement data in variable iic_blue_data.

Store infrared light measurement data in variable iic_ir_data.

Figure 4.19 Processing Measurement Results
4.18 Waiting for Communication Restart

Set count value to the variable for timer counting.
Measurement time 1 s [25 ms × 40 = 1000 ms]

Starting Timer RA operation
tra_start()

tcstf_tracr = 1 ?
= 1 (during count)
= 0 (count stopped)

traic ← 0x00

Wait until time measurement is complete and timer RA stops.
Disable timer RA interrupt.

Figure 4.20 Waiting for Communication Restart

4.19 Processing I^2C Communication Error

iic_error()

return

Figure 4.21 Processing I^2C Communication Error
4.20 I2C bus Interface Interrupt Handling

```plaintext
iic_int()

  tend_icsr ← 0
  Clear transmission end flag.

  iic_status_buf ← CBUSBUSY
  Set "CBUSBUSY" to status buffer.
  = 0 (slave mode)
  = 1 (master mode)

  mst_iccr1 = 1 ?
  = 0 (slave mode)
  = 1 (master mode)

  iic_startCond_flag = 1 ?
  = 0 (start condition not issued)
  = 1 (start condition issued)

  iic_status_buf ← CBUSBUSY
  Set "CBUSBUSY" to status buffer.

  mst_iccr1 = 1 ?
  = 0 (slave mode)
  = 1 (master mode)

  iic_startCond_flag = 1 ?
  = 0 (start condition not issued)
  = 1 (start condition issued)

  iic_startCond_flag ← 0
  Clear start condition flag.

  iic_status_buf ← CSLAVEBUSY
  Set "CSLAVEBUSY" to status buffer.

  ackbr_icier = 0 ?
  = 1 (NACK detected)
  = 0 (ACK detected)

  slave_addr_rw ← iic_status & 0x01
  Acquire transfer direction.
  = 0 (transmission mode)
  = 1 (reception mode)

  slave_addr_rw = 0 ?
  = 0 (transmission mode)
  = 1 (reception mode)

  Communication error end processing
  iic_error_exit_int()

  Data reception
  iic_data_rcv_int()

  Continuous data transmission
  iic_cont_data_trs_int()

return

Data transmission
iic_data_trs_int()

Processing when interrupt is generated by address transmission

Processing when interrupt is generated by data transmission/reception
```

Figure 4.22  I2C bus Interface Interrupt Handling
4.21 Data Transmission

```
iic_data_trs_int()

icdrt ← *iic_data_pointer

iic_data_pointer++

return
```

Transmit data stored in transmit buffer.

Increment pointer to transmit buffer by 1.

Figure 4.23 Data Transmission

4.22 Continuous Data Transmission

```
iic_cont_data_trs_int()

iic_status_buf ← CNOACK

ackbr_iicer = 0 ?
= 1 (NACK detected)
= 0 (ACK detected)

iic_num_byte_buf --
Update remaining transmit data count.

iic_num_byte_buf ≠ 0 ?
= 0 (no transmit data remaining)
≠ 0 (transmit data remaining)

Data transmission
iic_data_trs_int()

iic_status ← iic_status & 0x7f
Clear communication-in-progress flag.

Communication error end processing
iic_error_exit_int()

return
```

Set "CNOACK" to status buffer.

Figure 4.24 Continuous Data Transmission
4.23 Reception Settings

- `iic_set_rcv_int()`: Call the function to set the reception mode.
- `tend_icsr ← 0`: Clear transmission end flag.
- `scl_o_iccr2 = 1 ?`: SCL pin is "H"?
  - `= 1` (SCL pin is "H".)
  - `= 0` (SCL pin is "L".)
- `iccr1 ← iccr1 & 0xef`: Set reception mode.
- `iic_num_byte_buf ≠ 1 ?`: Data to be received is one byte?
  - `= 1` (Data to be received is one byte.)
  - `≠ 1` (Data to be received is two bytes or more.)
- `ackbt_icier ← 0`: Set to send "0" (ACK) at acknowledge timing.
- `rcvd_iccr1 ← 0`: Set to continue reception after next data reception.
- `tdre_icsr ← 0`: Clear transmit data empty flag.
- `teie_icier ← 0`: Disable transmission end interrupt request.
- `rie_icier ← 1`: Enable receive data full interrupt request.
- `dummy_data ← icdrr`: Start reception by dummy read.
- `return`: Return from the function.

Figure 4.25  Reception Settings
4.24 Data Reception

```c
iic_data_rcv_int()

iic_num_byte_buf-- Update remaining receive data count.

# 0 (no receive data remaining)

# 0 (receive data remaining)

# 1 (Data to be received is one byte.)

*iic_data_pointer ← icdrr

Store data read from ICDRR register in receive buffer.

iic_data_pointer++

Increment pointer to receive buffer by 1.

*iic_data_pointer ← icdrr

Store data read from ICDRR register in receive buffer.

ackbt_icier ← 1

rcvd_iccr1 ← 1

Set to send "1"(NACK) at acknowledge timing.

iic_num_byte_buf -- Update remaining receive data count.

iic_num_byte_buf ≠ 0 ?

= 1 (Data to be received is two bytes or more.)

ackbt_icier ← 1

rcvd_iccr1 ← 1

Set to send "1"(NACK) at acknowledge timing.

iic_num_byte_buf ≠ 0 ?

= 0 (no receive data remaining)

= 1 (Data to be received is one byte.)

*ic_data_pointer ← icdrr

_store data read from ICDRR register in receive buffer.

iic_data_pointer++

Increment pointer to receive buffer by 1.

iic_num_byte_buf ≠ 0 ?

= 0 (bus free)

= 1 (bus busy)

stop_icsr ← 0

Clear stop condition detection flag.

icr2 ← 0x30

Issue stop condition.

stop_icsr = 0 ?

= 0 (stop condition not detected)

= 1 (stop condition detected)

stop_icsr ← 0

Clear stop condition detection flag.

*iic_data_pointer ← icdrr

Store data read from ICDRR register in receive buffer.

rie_icier ← 0

Disable receive data full interrupt request.

rcvd_iccr1 ← 0

Set to continue reception after next data reception.

iccr1 ← iccr1 & 0xcf

Set slave reception mode.

iic_status ← iic_status & 0x7f

Clear communication-in-progress flag.

iic_status & 0x7f

Clear communication-in-progress flag.

*ic_data_pointer ← icdrr

Store data read from ICDRR register in receive buffer.

rie_icier ← 0

Disable receive data full interrupt request.

rcvd_iccr1 ← 0

Set to continue reception after next data reception.

iccr1 ← iccr1 & 0xcf

Set slave reception mode.

iic_status ← iic_status & 0x7f

Clear communication-in-progress flag.

iic_status & 0x7f

Clear communication-in-progress flag.

Figure 4.26 Data Reception
```
4.25 Communication Error End Processing

```
void iic_error_exit_int()
{
    iic_status ← iic_status_buf
    Update communication status.
    iic_num_byte ← iic_num_byte_buf
    Update transmit/receive data count.
    iic_num_byte_buf ← 0x00
    Clear transmit/receive data count.
    iic_status ← iic_status & 0x7f
    Clear communication-in-progress flag.
    return
}
```

Figure 4.27 Communication Error End Processing

4.26 Timer RA Interrupt Handling

```
void tra_int()
{
    tra_wait_dwncnt ← −
    Update variable for timer counting.
    if tra_wait_dwncnt = 0
    = 0 (time measurement completed)
    ≠ 0 (time measurement in progress)
    tstart_tracr ← 0
    Stop timer RA count.
    return
}
```

Figure 4.28 Timer RA Interrupt Handling
5. Sample Program
Sample code can be downloaded from the Renesas Electronics website.

6. Reference Documents
R8C/35C Group User’s Manual Hardware Rev.1.00
The latest version can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

Website and Support
Renesas Electronics website
http://japan.renesas.com/

Inquiries
http://japan.renesas.com/inquiry

All trademarks and registered trademarks are the property of their respective owners.
## Revision History

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>May 11, 2017</td>
<td>—</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins
   Handle unused pins in accordance 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 a product with a different part number, confirm that the change will not lead to problems.
   — The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
Notice

1. Descriptions of circuits, software, and 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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.

2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other disputes involving 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, including but not limited to, the product data, drawing, chart, program, algorithm, application examples.

3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.

4. You shall not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics products.

5. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.

   - "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment, and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.), traffic control (traffic lights), large-scale communication equipment, key financial terminal systems, safety control equipment, etc.

   Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantation devices, or any cause serious property damages (space and undersea repeaters, nuclear power control systems, aircraft control systems, key plant systems, military equipment, etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for which the product is not intended by Renesas Electronics.

6. When using the Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat radiation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions or failure or accident arising out of the use of Renesas Electronics products beyond such specified ranges.

7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics 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 ensure to implement safety measures to guard them against the possibility of bodily injury, injury or damage caused by fire, and social damage in the event of failure or malfunction of Renesas Electronics products, 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 by your own responsibility as warranty for your product/system. Because the evaluation of microcomputer software alone is very difficult and not practical, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please investigate applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive carefully and sufficiently and use Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall not use Renesas Electronics products or technologies for (1) any purpose relating to the development, design, manufacture, use, stocking, etc., of weapons of mass destruction, such as nuclear weapons, chemical weapons, or biological weapons, or missiles (including unmanned aerial vehicles (UAVs)) for delivering such weapons, (2) any purpose relating to the development, design, manufacture, or use of conventional weapons, or (3) any other purpose of disturbing international peace and security, and you shall not sell, export, lease, transfer, or release Renesas Electronics products or technologies to any third party whether directly or indirectly with knowledge or reason to know that the third party or any other party will engage in the activities described above. When exporting, selling, transferring, etc., Renesas Electronics products or technologies, you shall comply with any applicable export control laws and regulations promulgated and administered by the governments of the countries asserting jurisdiction over the parties or transactions.

10. Please acknowledge and agree that you shall bear all the losses and damages which are incurred from the misuse or violation of the terms and conditions described in this document, including this notice, and hold Renesas Electronics harmless, if such misuse or violation results from your resale or making Renesas Electronics products available any third party.

11. This document shall not be reprinted, 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.

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

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