RL78/G23
I2C Supporting Multiple Slave Address (Master)

Introduction

This application note describes how to use the master function of the I2C bus by using the IICCA serial interface. In the procedure described, you will operate four serial memory areas (256 bytes x 4) specified by different slave addresses.

This application note is associated with the application note of "I2C Supporting Multiple Slave Addresses (Slave)".

Target Device

RL78/G23

When applying the sample program covered in this application note to another microcomputer, modify the program according to the specifications for the target microcomputer and conduct an extensive evaluation of the modified program.
Contents

1. Specifications .......................................................................................................................... 3
  1.1 Basic Specifications of the I2C Bus (Master) ................................................................. 3
  1.2 Outline of Operation ................................................................................................. 5
  1.3 I2C Bus Control ....................................................................................................... 5
  1.4 Library for Communication to the Slave ................................................................. 5
  1.5 Serial Memory Control ............................................................................................ 6

2. Operation Confirmation Conditions .................................................................................. 7

3. Hardware Descriptions ........................................................................................................ 8
  3.1 Example of Hardware Configuration ........................................................................ 8
  3.2 List of Pins to be Used ............................................................................................. 9

4. Software Explanation ........................................................................................................... 9
  4.1 Setting of Option Byte .............................................................................................. 9
  4.2 List of Constants ...................................................................................................... 10
  4.3 List of Variables ..................................................................................................... 11
  4.4 List of Functions .................................................................................................... 12
  4.5 Specification of Functions ....................................................................................... 13
  4.6 Flowcharts ............................................................................................................. 18
    4.6.1 Main Processing ................................................................................................. 18
    4.6.2 Variables Initialization Processing ....................................................................... 21
    4.6.3 IICA0 Interrupt Processing Function ............................................................... 22
    4.6.4 IICA0 Master Interrupt Processing Function .................................................. 23
    4.6.5 Function That Generates an IICA0 Start Condition ........................................... 27
    4.6.6 Function that Generates an IICA0 Stop Condition ............................................. 28
    4.6.7 IICA0 Master Transmission Start Function ....................................................... 29
    4.6.8 IICA0 Data Transmission Processing Function ............................................... 30
    4.6.9 IICA0 Master Reception Start Function ........................................................... 31
    4.6.10 IICA0 Data Reception Processing Function ..................................................... 32
    4.6.11 Communication Status Polling Function ......................................................... 33
    4.6.12 50-μs Wait Function ......................................................................................... 34
    4.6.13 Communication Completion Wait Function ....................................................... 35
    4.6.14 I2C Bus Status Check Function ...................................................................... 36
    4.6.15 Function That Handles a 10-ms Interval Timer Interrupt ................................. 37

5. Sample code ...................................................................................................................... 38

6. Reference Documents ....................................................................................................... 38

Revision History .................................................................................................................... 39
1. Specifications

1.1 Basic Specifications of the I2C Bus (Master)

The following lists the basic specifications of the I2C bus.

- I2C bus to be connected: Fast mode (up to 400 kbps)
- Target slave (4 serial memories of 256 bytes)
  - Slave address 1: 0010000B (First serial memory)
  - Slave address 2: 0100100B (Second serial memory)
  - Slave address 3: 1011010B (Third serial memory)
  - Slave address 4: 1101011B (Fourth serial memory)

Note: In the RL78 family, the own address (7 bits) is expressed by the upper 7 bits of the SVA0 register. The least significant bit (LSB) of the SVA0 register is always 0.

For address transmission, the slave address together with the transfer direction (R/W) is written as 8-bit data to IICA shift register 0 (IICA0).

Table 1-1 lists peripheral functions to be used.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>IICA0</td>
<td>Master function of the I2C bus</td>
</tr>
<tr>
<td>TM01</td>
<td>50 μs interval timer interrupt</td>
</tr>
<tr>
<td>TM03</td>
<td>500 ms interval timer interrupt</td>
</tr>
<tr>
<td>TM07</td>
<td>10 ms interval timer interrupt</td>
</tr>
<tr>
<td>P52</td>
<td>Drives the operation status display LED.</td>
</tr>
<tr>
<td>P53</td>
<td>Drives the error display LED.</td>
</tr>
<tr>
<td>P137</td>
<td>Input to the operation start switch</td>
</tr>
</tbody>
</table>

The following describes the main settings.

1) Initial settings for IICA0

Table 1-2 lists Initial Settings for IICA0.

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PM6</td>
<td>03H</td>
<td>The pin used for both SCL and SDA signals is set for input during the initial setup period.</td>
</tr>
<tr>
<td>P6</td>
<td>00H</td>
<td>The pin used for both SCL and SDA signals is set to 0.</td>
</tr>
<tr>
<td>IICCTL01</td>
<td>0DH</td>
<td>Fast mode and digital filter enabled for the IICA0 operating clock fCLK/2</td>
</tr>
<tr>
<td>IICWL0</td>
<td>15H</td>
<td>SCLA0 low width</td>
</tr>
<tr>
<td>IICWH0</td>
<td>14H</td>
<td>SCLA0 high width</td>
</tr>
<tr>
<td>SVA0</td>
<td>10H</td>
<td>Sets the slave address to 10H.</td>
</tr>
<tr>
<td>IICF0</td>
<td>03H</td>
<td>In the initial status, the bus is idle and reservation for communication is prohibited.</td>
</tr>
<tr>
<td>IICCTL00</td>
<td>8CH</td>
<td>SPD interrupt disabled, wait at the 9th clock cycle, ACK response enabled</td>
</tr>
</tbody>
</table>
(2) Initial settings for TM0

Table 1-3 lists the initial settings for TM01.

**Table 1-3 Initial Settings for TM01**

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPS0</td>
<td>0018H</td>
<td>CK00: 125 kHz, CK01/CK02/CK03: 32 MHz</td>
</tr>
<tr>
<td>TT0</td>
<td>0002H</td>
<td>Channel 1 disabled</td>
</tr>
<tr>
<td>TMPR001</td>
<td>0</td>
<td>Interrupt priority: Level 2</td>
</tr>
<tr>
<td>TMR01</td>
<td>8000H</td>
<td>TM01 count clock: CK01 (32 MHz)</td>
</tr>
<tr>
<td>TDR01</td>
<td>063F</td>
<td>Interval time: 50 μs</td>
</tr>
<tr>
<td>TOM0</td>
<td>00H</td>
<td>TM01: Master mode</td>
</tr>
<tr>
<td>TOL0</td>
<td>00H</td>
<td>TM01 output: Positive logic</td>
</tr>
<tr>
<td>TO0</td>
<td>00H</td>
<td>TM01 output: 0</td>
</tr>
<tr>
<td>TOE0</td>
<td>00H</td>
<td>TM01 output: Disabled</td>
</tr>
</tbody>
</table>

(3) Initial settings for TM03

Table 1-3 lists the initial settings for TM03.

**Table 1-4 Initial Settings for TM03**

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPS0</td>
<td>0018H</td>
<td>CK00: 125 kHz, CK01/CK02/CK03: 32 MHz</td>
</tr>
<tr>
<td>TT0</td>
<td>0008H</td>
<td>Channel 3 disabled</td>
</tr>
<tr>
<td>TMPR003</td>
<td>0</td>
<td>Interrupt priority: Level 2</td>
</tr>
<tr>
<td>TMR03</td>
<td>8000H</td>
<td>TM03 count clock: CK00 (125 kHz)</td>
</tr>
<tr>
<td>TDR03</td>
<td>F423</td>
<td>Interval time: 500 ms</td>
</tr>
<tr>
<td>TOM0</td>
<td>00H</td>
<td>TM03: Master mode</td>
</tr>
<tr>
<td>TOL0</td>
<td>00H</td>
<td>TM03 output: Positive logic</td>
</tr>
<tr>
<td>TO0</td>
<td>00H</td>
<td>TM03 output: 0</td>
</tr>
<tr>
<td>TOE0</td>
<td>00H</td>
<td>TM03 output: Disabled</td>
</tr>
</tbody>
</table>

(4) Initial settings for TM07

Table 1-3 lists the initial settings for TM07.

**Table 1-5 Initial Settings for TM07**

<table>
<thead>
<tr>
<th>Register Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPS0</td>
<td>0008H</td>
<td>CK00: 125 kHz, CK01/CK02/CK03: 32 MHz</td>
</tr>
<tr>
<td>TT0</td>
<td>0080H</td>
<td>Channel 7 disabled</td>
</tr>
<tr>
<td>TMPR007</td>
<td>0</td>
<td>Interrupt priority: Level 2</td>
</tr>
<tr>
<td>TMR07</td>
<td>0000H</td>
<td>TM07 count clock: CK00 (125 kHz)</td>
</tr>
<tr>
<td>TDR07</td>
<td>04E1</td>
<td>Interval time: 10 ms</td>
</tr>
<tr>
<td>TOM0</td>
<td>00H</td>
<td>TM07: Master mode</td>
</tr>
<tr>
<td>TOL0</td>
<td>00H</td>
<td>TM07 output: Positive logic</td>
</tr>
<tr>
<td>TO0</td>
<td>00H</td>
<td>TM07 output: 0</td>
</tr>
<tr>
<td>TOE0</td>
<td>00H</td>
<td>TM07 output: Disabled</td>
</tr>
</tbody>
</table>
1.2 Outline of Operation

The I2C bus sends data to, or receives data from, four serial memory areas, each of which has a 1-byte register address that indicates a slave internal address.

The I2C bus performs the following operations sequentially in the following order for the four serial memory areas:
- Reads 256 bytes sequentially.
- Checks whether the data that was read is equal to the expected value (for the second and the following cycles).
- Writes data (0x00 to 0xFF) in units of 16 bytes.
- Reads 16-byte data 16 times.
- Checks whether the data that was read is equal to the expected value.
- Writes data (0xFF to 0x00) in units of 256 bytes.

1.3 I2C Bus Control

Table 1-1 shows the relationship between the status of the I2C bus and the value of g_status.

<table>
<thead>
<tr>
<th>I2C Bus Status</th>
<th>g_status</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>The I2C bus is idle.</td>
<td>0x00</td>
<td>IICBSY0 = 0</td>
</tr>
<tr>
<td>The I2C bus is being used by another master.</td>
<td>0x0F</td>
<td>IICBSY0 = 1, MSTS0 = 0</td>
</tr>
<tr>
<td>The I2C bus is being used in master operation.</td>
<td>0x20</td>
<td>MSTS0 = 1</td>
</tr>
<tr>
<td>The I2C bus is sending a slave address in master operation.</td>
<td>0x18</td>
<td>MSTS0 = 1</td>
</tr>
<tr>
<td>The I2C bus is sending a slave register address in master operation.</td>
<td>0x16</td>
<td>MSTS0 = 1</td>
</tr>
<tr>
<td>The I2C bus is receiving data in master operation.</td>
<td>0x14</td>
<td>MSTS0 = 1</td>
</tr>
<tr>
<td>Data reception ended normally.</td>
<td>0x12</td>
<td>MSTS0 = 1</td>
</tr>
<tr>
<td>Data transmission ended normally.</td>
<td>0x24</td>
<td>MSTS0 = 1</td>
</tr>
<tr>
<td>Processing of IICA0 is in progress or the I2C bus is being used by another master.</td>
<td>0x8F</td>
<td></td>
</tr>
<tr>
<td>NACK was detected for the slave address.</td>
<td>0x80</td>
<td></td>
</tr>
<tr>
<td>NACK from the slave was detected.</td>
<td>0xC0</td>
<td></td>
</tr>
</tbody>
</table>

1.4 Library for Communication to the Slave

This application note supports the following functions:

1. Function that starts writing data with the specified size (in bytes) to the specified slave
2. Function that writes data with the size (in bytes) specified from the specified address of the specified slave
3. Function that starts reading the specified data from the specified slave
4. Function that reads data with the size (in bytes) specified from the specified address of the specified slave
5. Function that performs polling to check whether the communication started by function 1 or 3 has ended
6. Function that waits for completion of the communication started by function 1 or 3
7. Function that generates a stop condition
1.5 Serial Memory Control

The I2C bus can write data to, or read data from, the address of the serial memory area specified by the combination of the slave address and the next 1-byte data (register address).

Figure 1-1 shows the sequence of writing data in succession to a slave serial memory area by specifying register addresses.

The master first sends a start condition (ST) followed by an 8-bit slave address that consists of a 7-bit slave address and the transfer direction (W). The master then sends a register address that specifies the internal address of the serial memory area. After that, the master sequentially sends pieces of write data. After sending the final data piece, the master generates a stop condition (SP) to complete the communication.

**Figure 1-1 Continuous Data Write to Designated Register Addresses**

<table>
<thead>
<tr>
<th></th>
<th>Slave address + W</th>
<th>Register address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>Write data 1</td>
<td>Write data n</td>
</tr>
<tr>
<td></td>
<td>Continued below</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1-2 shows the sequence of reading data in succession from a slave serial memory area by specifying the register address.

The master first sends a start condition (ST) followed by an 8-bit slave address that consists of a 7-bit slave address and the transfer direction (W). The master then sends a register address that specifies the internal address of the serial memory area. Next, the master sends a restart condition (ST) followed by an 8-bit slave address that consists of a 7-bit slave address and the transfer direction (R). After that, the master sequentially sends pieces of data from the specified register address (sequential reads). When the master returns NACK in response to the received data, the slave stops transmission. Finally, the master generates a stop condition (SP) to complete the communication.

**Figure 1-2 Continuous Data Read from Designated Register Addresses**

<table>
<thead>
<tr>
<th></th>
<th>Slave address + W</th>
<th>Register address</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST</td>
<td>Read data 1</td>
<td>Read data n</td>
</tr>
<tr>
<td></td>
<td>Continued below</td>
<td></td>
</tr>
<tr>
<td>▼: ACK response (from master)</td>
<td>▼: NACK response (from master)</td>
<td></td>
</tr>
</tbody>
</table>
2. **Operation Confirmation Conditions**

The operation of the sample code provided with this application note has been tested under the following conditions.

**Table 2-1** Operation Confirmation Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MCU used</strong></td>
<td>RL78/G23 (R7F100GLG)</td>
</tr>
<tr>
<td><strong>Board used</strong></td>
<td>RL78/G23 Fast Prototyping Board (RTK7RLG230CLG000BJ)</td>
</tr>
<tr>
<td><strong>Operating frequency</strong></td>
<td>High-speed on-chip oscillator clock: 32 MHz</td>
</tr>
<tr>
<td></td>
<td>CPU/peripheral hardware clock: 32 MHz</td>
</tr>
<tr>
<td><strong>Operating voltage</strong></td>
<td>5.0 V (can be operated at 4.0 V to 5.5 V)</td>
</tr>
<tr>
<td></td>
<td>LVD0 detection voltage: Reset mode</td>
</tr>
<tr>
<td></td>
<td>At rising edge TYP. 3.96 V (3.84 V to 4.08 V)</td>
</tr>
<tr>
<td></td>
<td>At falling edge TYP. 3.88 V (3.76 V to 4.00 V)</td>
</tr>
<tr>
<td><strong>Integrated development environment (CS+)</strong></td>
<td>CS+ V8.05.00 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td><strong>C compiler (CS+)</strong></td>
<td>CC-RL V1.10.00 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td><strong>Integrated development environment (e2studio)</strong></td>
<td>e2 studio V2021-04 (21.4.0) from Renesas Electronics Corp.</td>
</tr>
<tr>
<td><strong>C compiler (e2studio)</strong></td>
<td>CC-RL V1.10.00 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td><strong>Integrated development environment (IAR)</strong></td>
<td>IAR Embedded Workbench for Renesas RL78 V4.21.1 from IAR Systems Corp.</td>
</tr>
<tr>
<td><strong>C compiler (IAR)</strong></td>
<td>IAR C/C++ Compiler for Renesas RL78 V4.21.1 from IAR Systems Corp.</td>
</tr>
<tr>
<td><strong>Smart configurator (SC)</strong></td>
<td>V1.0.1 from Renesas Electronics Corp.</td>
</tr>
<tr>
<td><strong>Board support package (BSP)</strong></td>
<td>V1.00 from Renesas Electronics Corp.</td>
</tr>
</tbody>
</table>
3. Hardware Descriptions

3.1 Example of Hardware Configuration

Figure 3-1 Hardware Configuration shows an example of the hardware configuration used in the application note.

Figure 3-1 Hardware Configuration

Note 1. This schematic circuit diagram is simplified to show the outline of connections. When creating actual circuits, design them using appropriate pin processing so that the circuits meet electrical characteristics. (Connect input-only ports to VDD or VSS individually through a resistor.)

Note 2. Connect pins (with a name beginning with EVSS), if any, to VSS, and connect pins (with a name beginning with EVDD), if any, to VDD.

Note 3. Set VDD to a voltage not less than the reset release voltage (VLVD0) set by the LVD.
### 3.2 List of Pins to be Used

Table 3-1 lists the pins to be used and their functions.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P60 (D15)</td>
<td>Input / Output</td>
<td>SCL signal</td>
</tr>
<tr>
<td>P61 (D14)</td>
<td>Input / Output</td>
<td>SDA signal</td>
</tr>
<tr>
<td>P53 (D16)</td>
<td>Output</td>
<td>Drives LED1 (in operation)</td>
</tr>
<tr>
<td>P52 (D17)</td>
<td>Output</td>
<td>Drives LED2 (in error)</td>
</tr>
<tr>
<td>P137 (D18)</td>
<td>Input</td>
<td>Switch (SW) input</td>
</tr>
</tbody>
</table>

### 4. Software Explanation

#### 4.1 Setting of Option Byte

Table 4-1 shows the option byte settings.

<table>
<thead>
<tr>
<th>Address</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000C0H / 010C0H</td>
<td>11101111B</td>
<td>Disables the watchdog timer. (Counting stopped after reset)</td>
</tr>
<tr>
<td>000C1H / 010C1H</td>
<td>11111010B</td>
<td>LVD0 detection voltage: reset mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>At rising edge TYP. 3.96 V</td>
</tr>
<tr>
<td></td>
<td></td>
<td>At falling edge TYP. 3.88 V</td>
</tr>
<tr>
<td>000C2H / 010C2H</td>
<td>11101000B</td>
<td>HS mode,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>High-speed on-chip oscillator clock (fH): 32 MHz</td>
</tr>
<tr>
<td>000C3H / 010C3H</td>
<td>10000100B</td>
<td>Enables on-chip debugging</td>
</tr>
</tbody>
</table>
### 4.2 List of Constants

Table 4-2 lists the constants that are used in the sample code.

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED1_pin</td>
<td>P5_bit.no3</td>
<td>Port corresponding to the D16 pin</td>
</tr>
<tr>
<td>LED2_pin</td>
<td>P5_bit.no2</td>
<td>Port corresponding to the D17 pin</td>
</tr>
<tr>
<td>SW_IN</td>
<td>P13_bit.no7</td>
<td>Port corresponding to the D18 pin. Input pin for the on-board switch.</td>
</tr>
<tr>
<td>LED_ON</td>
<td>0</td>
<td>Value of common data for turning the LED on</td>
</tr>
<tr>
<td>LED_OFF</td>
<td>1</td>
<td>Value of common data for turning the LED off</td>
</tr>
<tr>
<td>INC_DATA[16][16]</td>
<td>0x00 to 0xFF</td>
<td>Write data 1 for RAM areas</td>
</tr>
<tr>
<td>RAM1</td>
<td>00100000b</td>
<td>7-bit slave address for RAM1</td>
</tr>
<tr>
<td>RAM2</td>
<td>0100100b</td>
<td>7-bit slave address for RAM2</td>
</tr>
<tr>
<td>RAM3</td>
<td>1011010b</td>
<td>7-bit slave address for RAM3</td>
</tr>
<tr>
<td>RAM4</td>
<td>1101011b</td>
<td>7-bit slave address for RAM4</td>
</tr>
<tr>
<td>PAGE_SIZE</td>
<td>256</td>
<td>RAM page size</td>
</tr>
<tr>
<td>HIGH</td>
<td>0x01</td>
<td>High level</td>
</tr>
<tr>
<td>LOW</td>
<td>0x00</td>
<td>Low level</td>
</tr>
<tr>
<td>WAITTIME</td>
<td>1000</td>
<td>Timeout time (number of loops) for detection of a start condition, etc.</td>
</tr>
<tr>
<td>BUS_FREE</td>
<td>0x00</td>
<td>The I2C bus is idle.</td>
</tr>
<tr>
<td>BUS_BUSY</td>
<td>0x0F</td>
<td>The I2C bus is being used by another master.</td>
</tr>
<tr>
<td>BUS_HOLD</td>
<td>0x20</td>
<td>The I2C bus is being used in master operation.</td>
</tr>
<tr>
<td>TX_SADDR</td>
<td>0x18</td>
<td>The I2C bus is sending a slave address in master operation.</td>
</tr>
<tr>
<td>TX_ADDR_REG</td>
<td>0x16</td>
<td>The I2C bus is sending the value of a register address in master operation.</td>
</tr>
<tr>
<td>RX_MODE</td>
<td>0x14</td>
<td>I2C reception is in progress in master operation.</td>
</tr>
<tr>
<td>TX_MODE</td>
<td>0x12</td>
<td>I2C transmission is in progress in master operation.</td>
</tr>
<tr>
<td>OP_END</td>
<td>0x20</td>
<td>I2C communication was completed in master operation.</td>
</tr>
<tr>
<td>RX_END</td>
<td>0x24</td>
<td>I2C reception was completed in master operation.</td>
</tr>
<tr>
<td>TX_END</td>
<td>0x22</td>
<td>I2C transmission was completed in master operation.</td>
</tr>
<tr>
<td>SUCCESS</td>
<td>0x00</td>
<td>Normal operation</td>
</tr>
<tr>
<td>COM_ERROR</td>
<td>0xFF</td>
<td>Error occurred due to a command parameter.</td>
</tr>
<tr>
<td>BUS_ERROR</td>
<td>0x8F</td>
<td>Error occurred because the I2C bus was busy.</td>
</tr>
<tr>
<td>NO_SLAVE</td>
<td>0x80</td>
<td>NACK was detected for a slave address.</td>
</tr>
<tr>
<td>NO_ACK</td>
<td>0xC0</td>
<td>NACK was detected for send data.</td>
</tr>
</tbody>
</table>
### 4.3 List of Variables

Table 4-3 lists global variables.

#### Table 4-3 Global Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Description</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>g_Tx_buff[]</td>
<td>257-byte send buffer</td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_Rx_buff[]</td>
<td>256-byte receive buffer</td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_sl_addr_T[]</td>
<td>RAM1, RAM2, RAM3, RAM4</td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_status</td>
<td>Current communication status:</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x00: The I2C bus is idle.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x0F: The I2C bus is being used by another master.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x12: Sending data.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x14: Receiving data.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x16: Sending a register address.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x18: Sending a slave address.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x20: Operating as a master.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x22: Data transmission ended normally.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x24: Data reception ended normally.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x8F: Processing of IICA0 is in progress or the bus is busy.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x80: NACK was detected for a slave address.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0xC0: NACK was detected.</td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>*gp_rx_address</td>
<td>Pointer to the storage area of received data</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td>uint16_t</td>
<td>g_rx_len</td>
<td>Number of bytes to be received</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td>uint16_t</td>
<td>g_rx_cnt</td>
<td>Number of received bytes</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>*gp_tx_address</td>
<td>Pointer to the data to be sent</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td>uint16_t</td>
<td>g_tx_cnt</td>
<td>Number of bytes to be sent</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_sl_addr</td>
<td>8-bit slave address</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_adr_reg</td>
<td>Register address area</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_adr_flag</td>
<td>Flag indicating whether the register address is enabled:</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x00: Disabled. Data is being sent or received.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x01: Enabled. Data is being sent.</td>
<td>main()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0x10: Enabled. Data is being received.</td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_SW_state</td>
<td>Switch status</td>
<td>main()</td>
</tr>
</tbody>
</table>

---

*Note: The table above is a simplified representation of the actual content. The full table includes more detailed descriptions and function usage details.*
4.4 List of Functions

Table 4-4 shows a list of functions.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>_R_Config_IICA0_Create()</td>
<td>Performs IICA0 initialization</td>
</tr>
<tr>
<td>_r_Config_IICA0_interrupt()</td>
<td>Performs an IICA0 interrupt</td>
</tr>
<tr>
<td>_r_IICA0_master_handler()</td>
<td>Performs an IICA0 master interrupt</td>
</tr>
<tr>
<td>_R_IICA0_StartCondition()</td>
<td>Generates a start condition</td>
</tr>
<tr>
<td>_R_IICA0_StopCondition()</td>
<td>Generates a stop condition</td>
</tr>
<tr>
<td>_R_IICA0_Master_Send()</td>
<td>Starts data transmission to the slave</td>
</tr>
<tr>
<td>_R_IICA0_Tx()</td>
<td>Sends data to the specified slave address</td>
</tr>
<tr>
<td>_R_IICA0_Master_Receive()</td>
<td>Starts data reception from the slave</td>
</tr>
<tr>
<td>_R_IICA0_Rx()</td>
<td>Receives data from the specified slave address</td>
</tr>
<tr>
<td>_R_IICA0_poll()</td>
<td>Performs polling for the communication status</td>
</tr>
<tr>
<td>_wait_time()</td>
<td>Waits for 50 us</td>
</tr>
<tr>
<td>_R_IICA0_wait_comend()</td>
<td>Waits for completion of communication</td>
</tr>
<tr>
<td>_R_IICA0_check_comstate()</td>
<td>Checks the IICA0 communication status</td>
</tr>
<tr>
<td>_R_IICA0_bus_check()</td>
<td>Checks the status of the I2C bus</td>
</tr>
<tr>
<td>_r_Config_TAU0_7_interrupt()</td>
<td>Performs a 10-ms interval timer interrupt</td>
</tr>
</tbody>
</table>
4.5 Specification of Functions

The function specifications of the sample code are shown below.

R_Config_IICA0_Create()

Outline: IICA0 initialization
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: void R_Config_IICA0_Create(void)
Description: Specifies the IICA0 initial settings (fast mode, master).
Argument: None
Return Value: None

r_Config_IICA0_interrupt()

Outline: IICA0 interrupt
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: static void __near r_iica0_interrupt(void)
Description: Accepts an IICA0 interrupt request.
Argument: None
Return Value: None

r_IICA0_master_handler()

Outline: Master processing function for IICA0 interrupts
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: static void r_IICA0_master_handler(void)
Description: Performs an IICA0 interrupt while the I2C bus is operating as the master. If NACK is detected for a slave address, the I2C bus generates a stop condition and ends communication. In transmission mode, if ACK is detected for a slave address, the I2C bus sends the register address or data. In reception mode, the I2C bus receives and stores data.
Argument: None
Return Value: None

R_IICA0_StartCondition()

Outline: Generating a start condition
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: uint8_t R_IICA0_StartCondition(void)
Description: Generates a start condition and checks its result.
Argument: None
Return Value: uint8_t
0x00 (SUCCESS): Generation of a start condition was completed.
0x8F (BUS_ERROR): An error occurred.

R_IICA0_StopCondition()

Outline: Generating a stop condition
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: uint8_t R_IICA0_StopCondition(void)
Description: Generates a stop condition and detects its result.
Argument: None
Return Value: uint8_t
0x00 (SUCCESS): A stop condition was generated and communication was completed.
0x8F (BUS_ERROR): An error occurred.
R_IICA0_Master_Send()

Outline  Starting data transmission to the slave

Header  r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h

Declaration  uint8_t R_IICA0_Master_Send(uint8_t sladr8, uint8_t * const tx_buf, uint16_t tx_num)

Description  If data transmission to IICA0 or data reception from IICA0 has ended normally, this function performs the following:
- If the I2C bus is idle, the function generates a start condition.
- If IICA0 is operating as the master, the function copies the transmission parameters and sends the slave address.

In other cases, the function returns an error.

Argument  
- uint8_t sladr8  Slave address
- uint8_t * const tx_buf  Pointer to the storage area of send data
- uint16_t tx_num  Number of bytes to be sent

Return Value  
- uint8_t Status:
  - 0x00 (SUCCESS): Communication started normally.
  - 0x8F (BUS_ERROR): Processing of IICA0 is in progress or the I2C bus is busy.

R_IICA0_Tx()

Outline  Sending data to the specified slave address

Header  r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h

Declaration  uint8_t R_IICA0_Tx(uint8_t sladr7, uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num)

Description  ① This function checks the status of the I2C bus.
  - If the I2C bus is idle, this function generates a start condition.
  - If the I2C bus is busy, this function returns an error.

② After generating a start condition, the function enables the register address (g_adr_flag = 0x01).

③ The function converts the slave address from a 7-bit format to an 8-bit format, and starts sending data.

④ The function waits for completion of communication.

Argument  
- uint8_t sladr7  7-bit slave address
- uint8_t adr  Value to be set for the register address
- uint8_t * const tx_buf  Pointer to the storage area of send data
- uint16_t tx_num  Number of bytes to be sent

Return Value  
- uint8_t Result:
  - 0x00 (SUCCESS): Transmission was completed.
  - 0x8F (BUS_ERROR): Processing of IICA0 is in progress or the I2C bus is busy.
## R_IICA0_Master_Receive()

**Outline**
Starting sending of data to the specified slave

**Header**
r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h

**Declaration**
uint8_t R_IICA0_Master_Receive(uint8_t sladr8, uint8_t * const tx_buf, uint16_t tx_num)

**Description**
If data transmission to IICA0 or data reception from IICA0 has ended normally, this function performs processing as follows:
- If the I2C bus is idle, the function generates a start condition.
- If IICA0 is operating as the master, the function copies transmission parameters and sends the slave address.

In other cases, the function returns an error.

**Argument**
- uint8_t sladr8: Slave address
- uint8_t * const tx_buf: Pointer to the storage area of received data
- uint16_t tx_num: Number of bytes to be received

**Return Value**
- uint8_t: Status:
  - 0x00 (SUCCESS): Communication started normally.
  - 0x8F (BUS_ERROR): Processing of IICA0 is in progress or the I2C bus is busy.
  - 0xFF (COM_ERROR): Error in the number of received bytes

## R_IICA0_Rx()

**Outline**
Receiving data from the specified slave address

**Header**
r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h

**Declaration**
uint8_t R_IICA0_Rx(uint8_t sladr7, uint8_t adr, uint8_t * const tx_buf, uint16_t tx_num)

**Description**
① This function checks the status of the I2C bus.
   - If the I2C bus is idle, the function generates a start condition.
   - If the I2C bus is busy, the function returns an error.
② After generating a start condition, the function sets communication parameters.
③ The function enables the register address (g_adr_flag = 0x10).
④ The function converts the slave address from a 7-bit format to an 8-bit format, and starts receiving data.
⑤ The function waits for completion of communication.

**Argument**
- uint8_t sladr7: 7-bit slave address
- uint8_t adr: Value to be set for the register address
- uint8_t * const tx_buf: Pointer to the storage area of send data
- uint16_t tx_num: Number of bytes to be sent

**Return Value**
- uint8_t: Result:
  - 0x00 (SUCCESS): Reception was completed.
  - 0x8F (BUS_ERROR): Processing of IICA0 is in progress or the I2C bus is busy.
  - 0xFF (COM_ERROR): Error in the number of received bytes
R_IICA0_poll()
Outline: Performing polling for the IICA0 communication status
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: uint8_t R_IICA0_poll(void)
Description: This function checks the status of the communication that is in progress.
Argument: None
Return Value: uint8_t
Communication status:
0x00 (SUCCESS): Communication is complete (successful)
0x01 (ON_COMMU): Communication is in progress
0x8F (BUS_ERROR): Processing of IICA0 is in progress or the I2C bus is busy.
0x80 (NO_SLAVE): NACK was detected for the slave address.
0xC0 (NACK): NACK was detected.

wait_time()
Outline: Waiting for 50 us
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: void wait_time(void)
Description: Waits for 50 us.
Argument: None
Return Value: None

R_IICA0_wait_comend()
Outline: Waiting for completion of IICA0 communication
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: uint8_t R_IICA_wait_comend(uint8_t stop)
Description: This function waits for completion of the communication that is in progress. If an error is detected after communication is completed, the function generates a stop condition and ends communication.
Argument: uint8_t stop
0x00: Generate a stop condition in the case of normal end.
0x01: Generate a stop condition when communication is completed.
Return Value: uint8_t
Communication result:
0x00 (SUCCESS): Communication is complete (successful).
0x8F (BUS_ERROR): Processing of IICA0 is in progress or the I2C bus is busy.
0x80 (NO_SLAVE): NACK was detected for the slave address.
0xC0 (NACK): NACK was detected.

R_IICA0_check_comstate()
Outline: Checking the IICA0 communication status
Header: r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h
Declaration: uint8_t R_IICA0_check_comstate(void)
Description: This function checks the status of IICA0 (by reading g_status).
Argument: None
Return Value: uint8_t
Value of g_status
**R_IICA0_bus_check()**

**Outline**
Checking the status of the I2C bus

**Header**
r_cg_macrodriver.h, Config_IICA0.h, r_cg_userdefine.h

**Declaration**
uint8_t R_IICA0_bus_check(void)

**Description**
This function checks the status of the I2C bus. If IICA0 is processing data, the function returns an error.
In the case where the I2C bus is secured but communication has ended or where the I2C bus is idle, the function generates a start condition.

**Argument**
None

**Return Value**
uint8_t

Result:
0x00 (SUCCESS): Generation of a start condition was completed.
0x8F (BUS_ERROR): Processing of IICA0 is in progress or the I2C bus is busy.

---

**r_tau0_channel7_interrupt()**

**Outline**
Performs a 10-ms interval interrupt

**Header**
Config_TAU0_7.h, r_cg_macrodriver.h, r_cg_userdefine.h

**Declaration**
static void r_tau0_channel7_interrupt(void)

**Description**
This function samples the status of a switch.

**Argument**
None

**Return Value**
None
4.6 Flowcharts

4.6.1 Main Processing

Figure 4-1 to Figure 4-3 shows the flowchart of the main processing.

Figure 4-1  Main Processing (1/3)

Variable sladr is assigned to argument 1 (7-bit address).
0x00 is assigned to argument 2 (register address).
g_Rx_buff is assigned to argument 3 (receive data buffer).
256 is assigned to argument 4 (number of received bytes).

main()

HALT()

Initial setup is performed.
Communication starts when the switch is pressed.
Data transmission/reception and comparison are performed for RAM1 to RAM4.

Variable sladr is assigned to argument 1 (7-bit address).
0x00 is assigned to argument 2 (register address).
g_Rx_buff is assigned to argument 3 (receive data buffer).
256 is assigned to argument 4 (number of received bytes).

Error detected?

Yes

Display the error

No

Data compared?

Yes

Compare data:
i = 0, 256, +1

Compare data

Different?

Yes

Display the error

No

Check the data

No

Transmission/reception with RAM1 to RAM4:
j = 0, 4, +1

Set the slave address:

Receive data from the slave:
R_IICA0_Rx()

0 (LED_ON) is set for the LED2_pin bit.
The function waits for an interrupt in HALT mode.

If variable ram_valid is 0, data comparison is not performed.
The received data is compared with the expected value.

0 (LED_ON) is set for the LED2_pin bit.
Figure 4-2  Main Processing (2/3)

B

Send data:  
\[ i = 0, 16, +1 \]

Send data to the slave:  
\( \text{R\_IICA0\_Tx()} \)

Error detected?  
\( \text{No} \)

Display error

Send data

Initialize the register address value

Send data to the slave:  
\( \text{R\_IICA0\_Master\_Send()} \)

Started normally?  
\( \text{Yes} \)

Wait for completion of communication:  
\( \text{R\_IICA0\_wait\_comend()} \)

Communication error occurred?  
\( \text{No} \)

Display the error

Data reception:  
\[ i = 0, 16, +1 \]

Receive data from the slave:  
\( \text{R\_IICA0\_Master\_Receive()} \)

Wait for completion of communication:  
\( \text{R\_IICA0\_wait\_comend()} \)

Error occurred?  
\( \text{No} \)

Display the error

Receive data

sladr is assigned to argument 1 (slave address).  
\( i \times 16 \) is assigned to argument 2 (register address)  
&INC\_DATA[i][0] is assigned to argument 3 (buffer).  
16 is assigned to argument 4 (number of bytes to be sent)

The communication error that occurred is handled.

0 (LED\_ON) is set for the LED2\_pin bit.

Register address settings:
\( (g\_sl\_addr[i] << 1) \) is assigned to argument 1 (slave address).  
g\_Tx\_buff is assigned to argument 2 (register address)  
1 is assigned to argument 3 (number of bytes to be sent).

The function waits for completion of transmission.

0 (LED\_ON) is set for the LED2\_pin bit.

Data reception:
\( (g\_sl\_addr[i] << 1) \) is assigned to argument 1 (slave address)  
&g\_Rx\_buff[i*16] is assigned to argument 2 (buffer address).  
16 is assigned to argument 3 (number of received bytes).

0 (LED\_ON) is set for the LED2\_pin bit.
Figure 4-3 Main Processing (3/3)

If variable ram_valid is 0, comparison is not performed.

The received data is compared with the expected value.

0 (LED_ON) is set for the LED2_pin bit.

Transmission parameters:
- (g_sl_addr[i] << 1) is assigned to argument 1 (slave address).
- g_Tx_buff is assigned to argument 2 (register address).
- 257 is assigned to argument 3 (number of bytes to be sent).

0 (LED_ON) is set for the LED2_pin bit.

Variable ram_valid is set.

The LED1_pin bit is reversed.

The function waits until the TMIF03 bit is set to 1.
4.6.2 Variables Initialization Processing

Figure 4-4 shows the flowchart of the variables initialization function.

Legend:
- R_MAIN_UserInit(): The entry point of the user initialization function.
- 0 (clear the interrupt request) is set for the TMIF03 bit.
- 1 (disable interrupts) is set for the TMMK03 bit.
- 0x0008 (start TM03) is set for the TS0 register.
- Enable vector interrupts
- 1 is set for the IE bit.

Start a 10-ms timer: R_Config_TAU0_7_Start()

Start a 500-ms timer

return
4.6.3 IICA0 Interrupt Processing Function

Figure 4-5 shows the flowchart of the IICA0 interrupt processing function.

**Figure 4-5** IICA0 Interrupt Processing Function

- r_Config_iica0_interrupt()
- MSTSO = 1 ?
  - No
  - IICA0 master interrupt processing: iica0_master_handler()
- Yes
- RETI

If the I2C bus is operating as the master, the function continues processing.
In other cases, the function exits.
4.6.4 IICA0 Master Interrupt Processing Function

Figure 4-6 to Figure 4-9 show the flowchart of the IICA0 master interrupt processing function.

**Figure 4-6** IICA0 Master Interrupt Processing Function (1/4)

```
r_IICA0_master_handler()

STD0 = 1 ?
   No
   This function detects a start condition. (Response to the slave address.)
   Yes

ACKD0 = 1 ?
   No
   Checks whether the slave exists.
   Yes

TRC0 = 1 ?
   No
   Checks whether transmission is performed by the master.
   Yes

g_adr_flag = 0x00 ?
   No
   Checks whether data is sent or received without using a register address.
   Yes

Change the status

Does send data exist?
   No
   Processing to start sending data
   0x12 (TX_MODE) is assigned to variable g_status -- Transmission is in progress.
   If variable g_tx_cnt is 1 or greater, the function sends data.
   Yes

Send data

Update the send data

Change the status

Send the register address

Change the status

g_adr_reg is assigned to the IICA0 register.
```

0x22 (TX_END) is assigned to variable g_status. -- Transmission ends.
(Processing performed if the number of bytes to be sent is 0)
Figure 4-7  IICA0 Master Interrupt Processing Function (2/4)

**Processing to start receiving data**
- 0 is set for the WTIM0 bit.
- 1 is set for the ACKE0 bit.
- 0xFF (DUMMY_DATA) is assigned to the IICA0 register.
- 0x14 (RX_MODE) is assigned to variable g_status. -- Reception is in progress.

**NACK processing for the slave address**
- 1 is set for the SPT0 bit.
- 0x80 (NO_SLAVE) is assigned to variable g_status. -- No ACK response made to the slave address.

**Data transfer start timing processing**
- Checks whether the processing is data transmission.
- Checks ACK response for sent data

**Restart processing for data reception**
- If the data read operation (variable g_adr_flag = 0x10) is set in slave address transmission (variable g_status = 0x18), the function restarts to start the read operation.

- (g_sl_addr | 0x01) is assigned to the IICA0 register.
- 0x14 (RX_MODE) is assigned to variable g_status. -- Reception is in progress.
- 1 is assigned to the LREL0 register.
- 0x8F (BUS_ERROR) is assigned to variable g_status. -- The bus is in use.
Figure 4-8  IICA0 Master Interrupt Processing Function (3/4)

Continuation of data transmission processing

0x12 (TX_MODE) is assigned to variable g_status. -- Transmission is in progress.

If variable g_tx_cnt is 1 or greater, the function sends data.

*gp_tx_address is assigned to the IICA0 register. -- Send data
Variable gp_tx_address is incremented.
Variable g_tx_cnt is decremented.

0x22 (TX_END) is assigned to variable g_status. -- Transmission ended.
(Processing performed if the number of remaining bytes to be sent is 0.)

0xC0(NO_ACK) is assigned to variable g_status. -- There was no ACK response for sent data from the slave.

Change the status

Does send data exist?

Yes

Send data
Update the send data

Change the status

No

Change the status

Yes

Send data
Update the send data

Change the status

Change the status

G
Figure 4-9 IICA0 Master Interrupt Processing Function (4/4)

Data reception processing

If variable g_rx_cnt is less than or equal to variable g_rx_len, the function continues reception.

The receive buffer is assigned to the IICA0 register.
Variable gp_rx_address is incremented.
Variable g_rx_cnt is incremented.

0 is set for the ACKE0 bit. -- Response with ACK is prohibited.
1 is set for the WTIM0 bit. -- An interrupt occurs at the 9th clock cycle.
1 is set for the WREL0 bit. -- Wait is canceled (communication continues.)
1 is set for the WREL0 bit. -- Wait is canceled (communication continues.)

0x24 (RX_END) is assigned to variable g_status. -- Reception ended.
4.6.5  Function That Generates an IICA0 Start Condition
Figure 4-1 shows a flowchart of the function that generates an IICA0 start condition.

Figure 4-1  Function That Generates an IICA0 Start Condition

```
R_IICA0_StartCondition()

Set the dummy status

Generate a start condition

STD = 0 ?
  Yes
  Timed out?
    Yes
    Detected?
      Yes
      Change the status
        0x00 (SUCCESS) is assigned to variable "status".
        0x20 (BUS_HOLD) is assigned to variable g_status.
      No
      Set the dummy status
        0x8F (BUS_ERROR) is assigned to variable "status".

  No

No

Yes

This function detects a start condition.

Set a return value

return

Variable "status" is assigned to the return value.
```

1 is set for the STT0 bit.
4.6.6 Function that Generates an IIC A0 Stop Condition

Figure 4-2 shows the flowchart of the function that generates an IIC A0 stop condition.

Figure 4-2 Function That Generates an IIC A0 Stop Condition
4.6.7 IICA0 Master Transmission Start Function

Figure 4-3 shows the flowchart of the IICA0 master transmission start function.

**Figure 4-3 IICA0 Master Transmission Start Function**

```
R_IICA0_Master_Send()

Set the register address parameter

Check the communication status:
R_IICA0_bus_check()

No

Operating as the master?

Yes

Set the communication parameters

Set the slave address

Set a return value

return

0x00 is assigned to variable g_adr_flag.

Generates a start function if the IICA0 data transmission/reception has ended normally or the I2C bus is made idle.
The status is assigned to variable "status".

Argument 3 (tx_num) is assigned to variable g_iica0_tx_cnt.
Argument 2 (tx_buf) is assigned to variable gp_iica0_tx_address.

Argument 1 (sladr8) & 0xFE is assigned to the IICA0 register.

Variable "status" is assigned to the return value.
```
4.6.8  IICA0 Data Transmission Processing Function

Figure 4-4 shows the flowchart of the IICA0 data transmission processing function.

**Figure 4-4  IICA0 Data Transmission Processing Function**

This function checks the status of IICA0. If it is operating as the master, the function generates a start condition.

If variable "status" is 0x00, the I2C bus is available.

0x01 is assigned to variable g_adr_flag.
Argument 2 (adr) is assigned to variable g_adr_reg.
(Argument 1 (sladr7) << 1) is assigned to variable g_sl_addr.

Argument 4 (tx_num) is assigned to variable g_iica0_tx_cnt.
Argument 3 (tx_buf) is assigned to variable gp_iica0_tx_address.

Variable g_sl_addr is assigned to the IICA0 register.
(Communication over the I2C bus is started.)

Waits for completion of communication while performing polling at 50-µs intervals to check whether communication is completed.

The IICA0 communication status is assigned to variable "status".
4.6.9 IICA0 Master Reception Start Function

Figure 4-5 shows the flowchart of the IICA0 master reception start function.

**Figure 4-5 IICA0 Master Reception Start Function**

```
R_IICA0_Master_Receive()

Set the dummy status

No

Is the number of received bytes > 0?

Yes

Set the register address parameter

Check the communication status:
R_IICA0_bus_check()

No

Operating as the master?

Yes

Set the communication parameters

Set the slave address

Set a return value

return

0xFF (COM_ERROR) is assigned to variable "status".

0x00 is assigned to variable g_adr_flag.

This function checks the status of IICA0. If it is operating as the master, the function generates a start condition. The status is assigned to variable "status".

Argument 3 (rx_num) is assigned to variable g_iica0_rx_len. 0 is assigned to variable g_iica0_rx_len. Argument 2 (rx_buf) is assigned to variable gp_iica0_rx_address.

Argument 1 (sladr8) | 0x01 is assigned to the IICA0 register.

Variable "status" is assigned to the return value.
```
4.6.10 IICA0 Data Reception Processing Function

Figure 4-6 shows the flowchart of the IICA0 data reception processing function.

This function checks the status of IICA0. If it is operating as the master, the function generates a start condition. The status is assigned to variable "status".

If variable "status" is 0x00, the I2C bus is available.

0x10 is assigned to variable g_adr_flag.
Argument 2 (adr) is assigned to variable g_adr_reg.
(Argument 1 (sladr7) << 1) is assigned to variable g_sl_addr.
Argument 4 (rx_num) is assigned to variable g_iica0_rx_cnt.
0 is assigned to variable g_iica0_rx_cnt.
Argument 3 (rx_buf) is assigned to variable gp_iica0_rx_address.

Variable g_sl_addr is assigned to the IICA0 register.
(Communication over the I2C bus is started.)

Waits for completion of communication while performing polling at 50-µs intervals to check whether communication is completed.

The IICA0 communication status is assigned to variable "status".

0xFF (COM_ERROR) is assigned to variable "status".

Is the number of bytes 0?
Yes
Set a command error

Check the communication status
R_IICA0_bus_check()
### 4.6.11 Communication Status Polling Function

Figure 4-7 shows the flowchart of the communication status polling function.

**Figure 4-7  Communication Status Polling Function**

- **R_IICA0_poll()**
  - Read the communication status
  - **Is the status value smaller than 0x30?**
    - **No**
    - **Is the status value equal to or larger than OP_END?**
      - **Yes**
        - **Variable g_status is assigned to variable "status".**
        - If the value of variable "status" is larger than 0x30, the function exits due to an error.
        - If the value of variable "status" is 0x20 or larger, the bus is available.
        - If the value of variable "status" is smaller than 0x20, the status is "communication in progress" (ON_COMMU).
      - **No**
        - **Change the status to "bus available"**
  - **Set the status to "communication in progress"**
  - **Set a return value**
  - **return**

0x00 (SUCCESS) is assigned to variable "status".
Variable "status" is assigned to the return value.
4.6.12 50-μs Wait Function
Figure 4-8 shows the flowchart of the 50-μs wait function.

**Figure 4-8  50-μs Wait Function**

- **Wait_time()**
- **Set up INTTM01**
- **Start TM01**
- **Timed out?**
  - **Yes**
    - **Clear the INTTM01 interrupt request**
    - **Stop TM01**
    - **return**
  - **No**

0 is set for the TMIF01 bit.
1 is set for the TMMK01 bit. -- Interrupt mask

0x0002 is assigned to the TS0 register.

0 is set for the TMIF01 bit.
0x0002 is assigned to the TT0 register.
4.6.13 Communication Completion Wait Function

Figure 4-9 shows the flowchart of the communication completion wait function.

Figure 4-9  Communication Completion Wait Function

```
R_IICA0_wait_comend()

Set a dummy status

0x01 (ON_COMMU) is assigned to variable "status".

Perform polling to check the communication status:
R_IICA0_poll()

No

Communication completed?

Yes

Set a return value

return

If variable "status" is 0x20 or larger, the bus is available.

No

Generation of a stop condition needed?

Yes

Generate a stop condition:
R_IICA0_StopCondition()

If an error occurs, the function generates a stop condition.

No

Variable "status" is assigned to the return value.

Perform polling to check the communication status:
R_IICA0_poll()

Generation of a stop condition needed?

Yes

Wait for 50 µs
wait_time()

No

Set a return value
```
4.6.14 I2C Bus Status Check Function

Figure 4-10 shows the flowchart of the I2C bus status check function.

**Figure 4-10 I2C Bus Status Check Function**

```
R_IICA0_bus_check()

Set a dummy status

Is the status other than "communication in progress"?
  No
  Yes

Is the I2C bus available?
  No
  Yes

Generate a start condition:
R_IICA0_StartCondition()

Has the I2C bus been secured?
  No
  Yes

Update the bus status

Set a return value

return
```

0x8F (BUS_ERROR) is assigned to variable "status".

If the value of variable g_status is 1xH, the function ends processing due to an error because the communication is in progress (the bus is busy).

IICA0 is not performing communication.

The function checks whether IICA0 is operating as the bus master and whether the I2C bus is idle.

The function generates a start condition for the I2C bus.

The function checks whether generation of a start condition was successful.

0x20 (BUS_HOLD) is assigned to variable g_status.

Variable "status" is assigned to the return value.
4.6.15 Function That Handles a 10-ms Interval Timer Interrupt
Figure 4-11 shows the flowchart of the function that handles a 10-ms interval timer interrupt.

Figure 4-11  Function That Handles a 10-ms Interval timer Interrupt

- r_Config_TAU0_7_interrupt()
- Prepare for reading the switch status
- SW_IN = 1 ?
  - No
    - (g_SW_state << 1) is assigned to variable g_SW_state.
  - Yes
    - Process the switch status variable
    - Was the switch unpressed?
      - (g_SW_state + 1) is assigned to variable g_SW_state.
- RETI
5. **Sample code**

Sample code can be downloaded from the Renesas Electronics website.

6. **Reference Documents**

RL78/G23 User’s Manual: Hardware (R01UH0896)
RL78 family user’s manual software (R01US0015)
The latest versions can be downloaded from the Renesas Electronics website.

Technical update
The latest versions can be downloaded from the Renesas Electronics website.

---

**Website and Support**

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/contact/

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Jul.01.21</td>
<td>—</td>
<td></td>
<td>First Edition</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. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on
   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. 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 the 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.

5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.

6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between $V_{IL}$ (Max.) and $V_{IH}$ (Min.).

7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.
1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation 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 anti-counterfeiting or any other claims 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, drawings, charts, programs, algorithms, and 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 be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.

5. You shall not alter, modify, copy, or reverse engineer 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, copying or reverse engineering.

6. 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; industrial robots; etc.
   - “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 implantations, etc.), or may cause serious property damage (space system; underwater 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 any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.

7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to, to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVIOLABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.

8. When using 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 dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.

9. 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. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a 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. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using 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.

11. 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 comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.

12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.

13. This document shall not be reproduced, reprinted or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.

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

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

Corporate Headquarters
TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information
For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/

Trademarks
Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.