RX610 Group

Communication with EEPROM
Using the Renesas I²C Bus Module (RIIC)

Introduction

This application note presents a sample program that communicates with an EEPROM (in single master mode) using the Renesas MCU I²C bus interface module.

Target Device

The RX610 Group products

Other members of the RX Family that have the same I/O registers (peripheral unit control registers) as the RX610 Group products can also use the code from this application note. Note, however, that since certain aspects of the functions used may be changed in other devices due to function additions or other differences, the documentation for the device used must be checked carefully before using this code. When using this code in an end product or other application, its operation must be tested and evaluated thoroughly.

Contents

1. Specifications .................................................................................................................................... 2
2. Operation Test Environment ............................................................................................................. 3
3. Operation ........................................................................................................................................... 4
4. Software ............................................................................................................................................ 6
5. Reference Documents ..................................................................................................................... 28
1. Specifications

This sample program communicates with the EEPROM to write 8 bytes of data and then read the written data back. Between the write and read operations, it uses acknowledge polling to verify that the EEPROM write has completed.

1.1 Connection Diagram

Figure 1 shows the connections in the application example presented in this application note.

![Connection Diagram](image)

**Figure 1** Connection Diagram

1.2 RIIC Settings

Table 1 lists the RIIC settings described in this application note.

<table>
<thead>
<tr>
<th>Item</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating frequencies</td>
<td>• Input clock (EXTAL): 12.5 MHz</td>
</tr>
<tr>
<td></td>
<td>• System clock (ICLK): 100 MHz</td>
</tr>
<tr>
<td></td>
<td>• Peripheral module clock (PCLK): 50 MHz</td>
</tr>
<tr>
<td></td>
<td>• External bus clock (BCLK): 25 MHz</td>
</tr>
<tr>
<td></td>
<td>• Internal reference clock (IIC(\phi)): 50 MHz</td>
</tr>
<tr>
<td>Master/slave</td>
<td>Single master</td>
</tr>
<tr>
<td>Address format</td>
<td>7-bit address format</td>
</tr>
<tr>
<td>Transfer speed</td>
<td>1 Mbps</td>
</tr>
<tr>
<td>Timeout detection</td>
<td>• The detection function counts while the SCL(n) line is low.</td>
</tr>
<tr>
<td></td>
<td>• Long mode (16-bit counter (IIC(\phi)): about 1.31 ms)</td>
</tr>
</tbody>
</table>
1.3 EEPROM

Table 2 lists the specifications of the EEPROM used in the application example described in this application note.

Table 2  EEPROM Specifications

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catalog number</td>
<td>R1EX24512ASAS0A</td>
</tr>
<tr>
<td>Capacity</td>
<td>512k (64-kword x 8-bit)</td>
</tr>
<tr>
<td>Slave address</td>
<td>Slave address: A6h</td>
</tr>
<tr>
<td></td>
<td>Bit 0 is the R/W bit. Bits 1 and 2 depend on the values of the A0 and A1 pins, respectively.</td>
</tr>
<tr>
<td></td>
<td>• A0 pin: High</td>
</tr>
<tr>
<td></td>
<td>• A1 pin: High</td>
</tr>
<tr>
<td>Write protection</td>
<td>Always released.</td>
</tr>
<tr>
<td></td>
<td>• WP pin: low</td>
</tr>
</tbody>
</table>

2. Operation Confirmation Environment

Table 3 lists the environment used for confirming the operation of this application example.

Table 3  Operation Confirmation Environment

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>RX610 (R5F56108VNFP)</td>
</tr>
<tr>
<td>Board</td>
<td>Evaluation board</td>
</tr>
<tr>
<td>Power supply voltage</td>
<td>3.3 V (Supplied from E1)</td>
</tr>
<tr>
<td>Input clock</td>
<td>12.5 MHz (ICLK = 100 MHz, PCLK = 50 MHz, BCLK = 25 MHz)</td>
</tr>
<tr>
<td>Operating temperature</td>
<td>Room temperature</td>
</tr>
<tr>
<td>HEW</td>
<td>Version 4.07.01.004</td>
</tr>
<tr>
<td>Toolchain</td>
<td>RX Standard Toolchain (V.1.0.0.0)</td>
</tr>
<tr>
<td>Debugger/Emulator</td>
<td>E1 emulator</td>
</tr>
<tr>
<td>Debugger component</td>
<td>RX E1/E20 SYSTEM V.1.00.00.000</td>
</tr>
</tbody>
</table>
3. Operation

3.1 Writing to the EEPROM

This sample program uses master transmission for writing to an external EEPROM device. The RIIC module issues a start condition (S) and then sends the EEPROM’s slave address. Since the eighth bit at this time is the R/W bit, a 0 must be sent at write time (master transmission). After that, the memory address is sent as two 8-bit bytes, and then the data to be written is sent to the EEPROM in order. The 2-byte memory address transmitted at this time indicates the address for the write operation in EEPROM. After the transmission of all the data has completed, the RIIC module issues a stop condition (P) and releases the bus. Note that the write address in memory used in this application note is 0000h.

Figure 2 shows an example of the signals used when writing the EEPROM.

3.2 Reading from EEPROM

A compound format consisting of master transmission and master reception is used for reading data from EEPROM. First, the RIIC module issues a start condition (S) and then it transmits the EEPROM slave address and then a two byte (2 × 8 bits) memory address. At this time, the RIIC module sends 0 as the R/W bit in the EEPROM slave address transmission (master transmission). After that, it issues a restart condition (Sr) and sends the EEPROM slave address again. At this time, it transmits 1 as the R/W bit in the transmission to the EEPROM (master reception). After the EEPROM slave address has been sent, the data is read out from the EEPROM by the generation of the next clock cycle. During the read operation, the RIIC module transmits an ACK each time it receives a single byte. For the last data, however, it returns a NACK. After that, it generates a stop condition (P). Note that the memory address read by this sample program is 0000h.

Figure 3 shows an example of the signals used when reading the EEPROM.
3.3 Acknowledge Polling

Acknowledge polling is used as the method for determining whether or not the EEPROM is in the write in progress state. To perform acknowledge polling, the sample program issues a start condition and then sends the EEPROM slave address and then a stop condition. At this time, if the EEPROM is writing, it will return a 1 on the ACK clock (NACK). Inversely, if the write has completed, it will return 0 (ACK). This allows the sample program to determine whether or not a write is in progress.

Figure 4 shows the acknowledge polling signals.

![Acknowledge Polling Signals](image_url)
4. Software

4.1 Functions

Tables 4 and 5 list the functions in this sample program. The functions that are not in bold are static functions.

Table 4  Functions in File main.c

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
<td>Figure 7</td>
</tr>
<tr>
<td>SampleEepromWrite</td>
<td>EEPROM write processing example</td>
<td>Figure 11</td>
</tr>
<tr>
<td>SampleEepromRead</td>
<td>EEPROM read processing example</td>
<td>Figure 12</td>
</tr>
<tr>
<td>IICAckPolling</td>
<td>Acknowledge polling</td>
<td>Figure 13</td>
</tr>
<tr>
<td>CpuCreate</td>
<td>CPU initialization</td>
<td>Figure 8</td>
</tr>
<tr>
<td>CpuIntCreate</td>
<td>CPU interrupt setting</td>
<td>Figure 9</td>
</tr>
<tr>
<td>IICPortCreate</td>
<td>IIC port settings</td>
<td>Figure 10</td>
</tr>
</tbody>
</table>

Table 5  Functions in File iic.c

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Operation</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIC_Create</td>
<td>IIC initialization</td>
<td>Figure 14</td>
</tr>
<tr>
<td>IIC_Destroy</td>
<td>IIC termination processing</td>
<td>Figure 15</td>
</tr>
<tr>
<td>IIC_EepWrite</td>
<td>EEPROM write start processing</td>
<td>Figure 16</td>
</tr>
<tr>
<td>IIC_RandomRead</td>
<td>EEPROM read start processing</td>
<td>Figure 17</td>
</tr>
<tr>
<td>IIC_GetStatus</td>
<td>IIC status check</td>
<td>Figure 18</td>
</tr>
<tr>
<td>IIC_EEI_Int</td>
<td>Communication error or event interrupt</td>
<td>Figure 19</td>
</tr>
<tr>
<td>IIC_EEI_IntTimeOut</td>
<td>Timeout detection interrupt</td>
<td>Called from within IIC_EEI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 20</td>
</tr>
<tr>
<td>IIC_EEI_IntAL</td>
<td>Arbitration lost detected interrupt</td>
<td>Called from within IIC_EEI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 21</td>
</tr>
<tr>
<td>IIC_EEI_IntSP</td>
<td>Stop condition detected interrupt</td>
<td>Called from within IIC_EEI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 22</td>
</tr>
<tr>
<td>IIC_EEI_IntST</td>
<td>Start condition detected interrupt</td>
<td>Called from within IIC_EEI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 23</td>
</tr>
<tr>
<td>IIC_EEI_IntNack</td>
<td>NACK detected interrupt</td>
<td>Called from within IIC_EEI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 24</td>
</tr>
<tr>
<td>IIC_RXI_Int</td>
<td>Receive data full interrupt</td>
<td>Figure 25</td>
</tr>
<tr>
<td>IIC_RXI_IntEepRead</td>
<td>EEPROM read processing (master reception section)</td>
<td>Called from within IIC_RXI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 26</td>
</tr>
<tr>
<td>IIC_TXI_Int</td>
<td>Transmit data empty interrupt</td>
<td>Figure 27</td>
</tr>
<tr>
<td>IIC_TXI_IntEepWrite</td>
<td>EEPROM write processing</td>
<td>Called from within IIC_TXI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 28</td>
</tr>
<tr>
<td>IIC_TXI_IntEepRead</td>
<td>EEPROM read processing (master transmission section)</td>
<td>Called from within IIC_TXI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 29</td>
</tr>
<tr>
<td>IIC_TEI_Int</td>
<td>Transmission complete interrupt</td>
<td>Figure 30</td>
</tr>
<tr>
<td>IIC_TEI_IntEepWrite</td>
<td>Transmission end processing used after an EEPROM write</td>
<td>Called from within IIC_TEI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 31</td>
</tr>
<tr>
<td>IIC_TEI_IntEepRead</td>
<td>Transmission end processing used after an EEPROM read</td>
<td>Called from within IIC_TEI_Int()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 32</td>
</tr>
<tr>
<td>IIC_GenClkSP</td>
<td>Stop condition generation used when an error occurs</td>
<td>Called from within IIC_EEI_IntTimeOut() and IIC_EEI_IntAL()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Figure 33</td>
</tr>
<tr>
<td>IIC_Error</td>
<td>Error handling</td>
<td>Figure 34</td>
</tr>
</tbody>
</table>
4.2 Variables

4.2.1 Structures

Figure 5 shows the structure used as the argument to the functions IIC_EepWrite() and IIC_RandomRead(). Also, table 6 lists the members of this structure.

```c
struct str_IIC_API_T
{
    uint8_t   SlvAdr;  /* Slave Address, Don’t set bit0. It’s a Read/Write bit */
    uint16_t  PreCnt;  /* Number of Predata */
    uint8_t   *pPreData; /* Pointer for PreData (Memory Addr of EEPROM) */
    uint32_t  RWCnt;   /* Number of Data */
    uint8_t   *pRWData; /* Pointer for Data buffer */
};
typedef struct str_IIC_API_T IIC_API_T;
```

**Figure 5  Structure Uses as an Argument to IIC_EepWrite() and IIC_RandomRead()**

**Table 6 Members of the Structure IIC_API_T**

<table>
<thead>
<tr>
<th>Structure Member</th>
<th>Range of Values</th>
<th>Description</th>
</tr>
</thead>
</table>
| SlvAdr           | 00h to FEh     | Slave address
|                  |                | Since the low-order bit is the R/W bit, it should always be set to 0. |
| PreCnt           | 00h to FFh     | Memory address counter
|                  |                | This is always set to 2 in this sample program. |
| *pPreData        | —              | Memory address storage buffer pointer
|                  |                | On write: The address in EEPROM to write data to (write destination) |
|                  |                | On read: The address in EEPROM to read data from (write source) |
| RWCnt            | 0000 0000h to FFFF FFFFh | Data counter
|                  |                | On write: Number of data items to write to EEPROM |
|                  |                | On read: Number of data items to read from EEPROM |
| *pRWData         | —              | Data storage buffer pointer
|                  |                | On write: Storage source for data to write to EEPROM. |
|                  |                | On read: Storage destination for data read from EEPROM. |
### 4.2.2 Functions

Tables 7 and 8 list the functions in this sample program.

**Table 7 Functions in the File main.c**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t trm_buff[256]</td>
<td>Transmit data buffer</td>
</tr>
<tr>
<td>uint8_t rcv_buff[256]</td>
<td>Receive data buffer</td>
</tr>
<tr>
<td>uint8_t trm_eeprom_adr[2]</td>
<td>EEPROM slave address storage buffer (for write)</td>
</tr>
<tr>
<td>uint8_t rcv_eeprom_adr[2]</td>
<td>EEPROM slave address storage buffer (for read)</td>
</tr>
<tr>
<td>IIC_API_T iic_buff_prm[2]</td>
<td>Structure used as the argument to the functions IIC_EepWrite() and</td>
</tr>
<tr>
<td></td>
<td>IIC_RandomRead()</td>
</tr>
</tbody>
</table>

**Table 8 Functions in the File iic.c**

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static IIC_API_T iic_buff</td>
<td>Structure used as the argument to the functions IIC_EepWrite() and</td>
</tr>
<tr>
<td></td>
<td>IIC_RandomRead() (Used by both IIC_EepWrite() and IIC_RandomRead())</td>
</tr>
<tr>
<td>static int8_t iic_mode</td>
<td>Internal mode</td>
</tr>
<tr>
<td>static int8_t iic_status</td>
<td>IIC status</td>
</tr>
<tr>
<td>static uint32_t iic_trm_cnt</td>
<td>Internal IIC transmit counter</td>
</tr>
<tr>
<td>static uint32_t iic_rcv_cnt</td>
<td>Internal IIC receive counter</td>
</tr>
</tbody>
</table>
4.2.3 Enumerations

The IIC status, the IIC bus status, the internal mode, and the return value from the functions `IIC_EepWrite()` and `IIC_RandomRead()` are all declared as enumerations. The IIC status values are listed in table 9 and their state transition diagram are shown in figure 6. Also, table 10 lists the IIC bus status values, table 11 lists the internal modes, and table 12 lists the return values of the functions `IIC_EepWrite()` and `IIC_RandomRead()`.

The IIC status is stored at the address given by its first argument when the function `IIC_GetStatus()` is called. The internal mode is only used in the IIC-related functions in this sample program.

### Table 9  IIC Status Values (enum RiicStatus_t)

<table>
<thead>
<tr>
<th>Defined Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIIC_STATUS_IDLE</td>
<td>The idle state</td>
</tr>
<tr>
<td></td>
<td>The status transitions to this state after initialization in the function</td>
</tr>
<tr>
<td></td>
<td><code>IIC_Create()</code> The status also transitions to this state after either an</td>
</tr>
<tr>
<td></td>
<td>EEPROM write or an EEPROM read completes normally (after a stop condition is</td>
</tr>
<tr>
<td></td>
<td>detected).</td>
</tr>
<tr>
<td>RIIC_STATUS_ON_COMMUNICATION</td>
<td>Communication in progress</td>
</tr>
<tr>
<td></td>
<td>The status transitions to this state when communication is initiated by</td>
</tr>
<tr>
<td></td>
<td>either <code>IIC_EepWrite()</code> or <code>IIC_RandomRead()</code></td>
</tr>
<tr>
<td>RIIC_STATUS_NACK</td>
<td>NACK received</td>
</tr>
<tr>
<td></td>
<td>The status transitions to this state when a NACK is received.</td>
</tr>
<tr>
<td>RIIC_STATUS_FAILED</td>
<td>Communication failure</td>
</tr>
<tr>
<td></td>
<td>The status transitions to this state when a stop condition is detected</td>
</tr>
<tr>
<td></td>
<td>before either an EEPROM write or an EEPROM read completes.</td>
</tr>
<tr>
<td></td>
<td>In this sample program, since a stop condition is generated on</td>
</tr>
<tr>
<td></td>
<td>either a timeout or an arbitration lost, the status will transition to</td>
</tr>
<tr>
<td></td>
<td>this state on either of those events as well.</td>
</tr>
</tbody>
</table>

![Figure 6  IIC Status State Transition Diagram](image-url)
Table 10  IIC Bus Status (enum RiicBusStatus_t)

<table>
<thead>
<tr>
<th>Defined Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RiIC_BUS_STATUS_FREE</td>
<td>IIC bus busy</td>
</tr>
<tr>
<td>RiIC_BUS_STATUS_BBSY</td>
<td>IIC bus free</td>
</tr>
</tbody>
</table>

Table 11  Internal Modes (enum RiicInternalMode_t)

<table>
<thead>
<tr>
<th>Defined Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| IIC_MODE_IDLE                   | Idle mode
The internal mode transitions to idle mode on initialization by IIC_Create() or when a stop condition is detected. |
| IIC_MODE_EEP_READ               | EEPROM read mode
The internal mode transitions to this mode at the start of communication due to IIC_RandomRead(). |
| IIC_MODE_EEP_WRITE              | EEPROM write mode
The internal mode transitions to this mode at the start of communication due to IIC_EepWrite(). |

Table 12  IIC_EepWrite() and IIC_RandomRead() Return Value (enum RiicEepFnc_t)

<table>
<thead>
<tr>
<th>Defined Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RiIC_OK</td>
<td>This value is returned when communication starts up normally.</td>
</tr>
<tr>
<td>RiIC_BUS_BUSY</td>
<td>This value is returned when the I^2^C bus is busy.</td>
</tr>
<tr>
<td>RiIC_MODE_ERROR</td>
<td>This value is returned when the RIIC module has a communication operation in progress.</td>
</tr>
<tr>
<td>RiIC_PRM_ERROR</td>
<td>This value is returned when an illegal argument value is passed. (Only the function IIC_RandomRead() uses this value.)</td>
</tr>
</tbody>
</table>


4.3 Function Specifications

This section presents the specifications of the sample code functions that control the RIIC module.

<table>
<thead>
<tr>
<th>Function</th>
<th>Overview</th>
<th>Header</th>
<th>Declaration</th>
<th>Description</th>
<th>Arguments</th>
<th>Return value</th>
<th>Notes</th>
</tr>
</thead>
</table>
| IIC_Create    | Initializes the RIIC module.                  | r_apn_iic.h     | void IIC_Create(void)      | Performs the following settings.                                                                                                                                                                        | None      | None          | - Transfer speed setting: 1 Mbps  
- Interrupt settings  
- Timeout settings   |
|               |                                               |                 |                            | Arguments None                                                                                                                                                                                              | None      | None          | Notes                                                                 |

| IIC_Destroy   | Stops the RIIC module.                        | r_apn_iic.h     | void IIC_Destroy(void)     | Stops the RIIC module and clears all the RIIC module related registers.                                                                                                                                 | None      | None          | Notes If this function is called during a communication operation, it forcibly stops the RIIC module. |

| IIC_EepWrite  | Starts a write to the EEPROM.                | r_apn_iic.h     | int8_t IIC_EepWrite(IIC_API_T) | Uses master transmission to write to the EEPROM. If the I²C bus is busy or if the RIIC module is in the communication in progress state, it does not start master transmission. | IIC_API_T data1 | RIIC_OK, RIIC_BUS_BUSY, RIIC_MODE_ERROR  |
|               |                                               |                 |                            | Arguments | Return value | Notes See section 4.2.1, Structures, for details on the argument IIC_API_T data1.  
See section 4.2.3, Enumerations, for details on the return value.  
Bit 0 in the slave address (SlvAdr), which is a member of the argument structure, must be set to 0. |
### IIC_RandomRead

**Overview**
Starts a read from the EEPROM.

**Header**
r_apn_iic.h

**Declaration**
```c
int8_t IIC_RandomRead(IIC_API_T);
```

**Description**
This function reads data from the EEPROM using both master transmission and master reception. If the I²C bus is busy or the RIIC is already communicating, it does not start a master transmission.

**Arguments**
IIC_API_T data1

**Return value**
- If communication starts up normally: RIIC_OK
- If the I²C bus is busy: RIIC_BUS_BUSY
- If the RIIC module is communicating: RIIC_MODE_ERROR
- If the argument value is illegal: RIIC_PRM_ERROR

**Notes**
- See section 4.2.1, Structures, for details on the argument IIC_API_T data1.
- See section 4.2.3, Enumerations, for details on the return value.
- The argument is recognized as illegal if both the memory address counter and the data counter are 0.
- Bit 0 in the slave address (SlvAdr), which is a member of the argument structure, must be set to 0.

### IIC_GetStatus

**Overview**
Acquires the status of the RIIC module.

**Header**
r_apn_iic.h

**Declaration**
```c
void IIC_GetStatus(enum RiicStatus_t*, enum RiicBusStatus_t*);
```

**Description**
This function stores the IIC status in the area indicated by the first argument. It also stores the IIC bus state in the area indicated by the second argument.

**Arguments**
- enum RiicStatus_t *data1
- enum RiicBusStatus_t *data2

**Return value**
None

**Notes**
See section 4.2.3, Enumerations, for details on the arguments.
4.4 Flowchart

This section presents the flowcharts for the functions in this sample program.

**Figure 7** Main Processing

- **main**
  - CpuCreate
    - CPU initialization
  - IIC_Create
    - IIC initialization
  - SampleEepromWrite
    - Example of EEPROM write processing
  - IIC_AckPolling
    - Acknowledge polling
  - SampleEepromRead
    - Example of EEPROM read processing
  - IIC_Destroy
    - IIC termination processing

**Figure 8** CPU Initialization

- CpuCreate
  - Clock settings
    - ICLK = 100 MHz, BCLK = 25 MHz, PCLK = 50 MHz (EXTAL = 12.5 MHz)
  - CpuIntCreate
    - CPU interrupt settings
  - IICPortCreate
    - IIC port settings
  - Clear the module stop bit
    - Clears the module stop bit

**Figure 9** CPU Interrupt Settings

- CpuIntCreate
  - IPR settings
    - Sets the interrupt event priority to 4. (ICEX0, ICRX0, ICTX0, and ICTEX0)
  - Clear the IR flag
    - Clears the interrupt request flag.
  - IEN settings
    - Enables interrupts.
      (Note, however, that it is also necessary to set the interrupts enabled/disabled state with the ICIER register.)
  - End
Figure 10  IIC Port Settings

- **IICPortCreate**
  - Sets the data direction register
    - Sets the data direction register to the input direction.
  - Set the input buffer control register
    - Enables the input buffer.

- **End**

---

Figure 11  Sample EEPROM Write Processing

- **SampleEepromWrite**
  - Transmit data settings
    - Sets up the sample transmit data (0x00, 0x01, 0x02, ...).
  - Set the EEPROM memory address
    - Sets the EEPROM write address (0x0000).
  - Set up the IIC_EepWrite() argument buffer
    - Sets up the argument data for IIC_EepWrite().
      - EEPROM slave address, memory address length, memory address storage buffer pointer, transmit data count, and transmit data storage buffer pointer
  - IIC_EepWrite
    - Starts the write to EEPROM.
  - IIC_GetStatus
    - Waits for the completion of data transmission to EEPROM.
  - Has IIC communication completed?
    - Yes
      - IIC_GetStatus
    - No
      - IIC bus free?
        - Yes
          - End
        - No
          - IIC_GetStatus

- **End**
Sample EEPROM Read Processing

1. Clear the receive data buffer
   - Clears the receive data buffer.
2. Set the EEPROM memory address
   - Sets the address data for reading from EEPROM (0x0000)
3. Set up the IIC_RandomRead() argument buffer
   - Sets up the argument data for IIC_RandomRead(). (EEPROM slave address, memory address length, memory address storage buffer pointer, receive data count, and receive data storage buffer pointer)
4. IIC_RandomRead
   - Starts the read from EEPROM.
5. IIC_GetStatus
   - Waits for the completion of data reception from EEPROM.
6. Has IIC communication completed?
   - Yes
5. IIC_GetStatus
   - Waits for the IIC bus free state.
7. IIC bus free?
   - Yes
   - End
   - No

Figure 12  Sample EEPROM Read Processing
IICAckPolling

Set the EEPROM memory address

Set up the IIC_EepWrite() argument buffer

IIC_EepWrite

IIC_GetStatus

No

Has IIC communication completed?

Yes

NACK response?

No

Yes

Wait for the interval required for the next acknowledge polling operation

ACK response?

No

Yes

Have the specified number of iterations completed?

Yes

No

End

Sets the EEPROM write address (0x0000).

Sets up the argument data for IIC_EepWrite(). (EEPROM slave address)

Starts acknowledge polling.

Waits for IIC communication to complete.

Iterates acknowledge polling either until an ACK response is received or until the specified iteration count completes.

During each iteration, the function waits for the interval specified in the argument.

Figure 13 Acknowledge Polling
IIC_Create

RIIC reset
Reset all RIIC registers and its internal state by setting ICCR1.ICE to 0 and ICCR1.IICRST to 1.

Transfer speed setting
Sets the transfer speed to 1 Mbps.

Timeout setting
Sets up timeout operation. (Only count when SCL is high and use long mode.)

Invalidate slave address
Disables slave address detection.

Clear ACKBT protection
Enables writing to ACKBT. (To enable NACK responses from EEPROM during read processing.)

Interrupt settings
Interrupt settings
• Enable the timeout interrupt
• Enable the arbitration lost interrupt
• Disable the start condition detection interrupt
• Enable the stop condition detection interrupt
• Enable the NACK received interrupt
• Enable the receive data full interrupt
• Disable the transmission complete interrupt
• Enable the transmitted empty interrupt

Initialize RAM used in the RIIC module
Initializes the RIIC internal RAM.

Enable IIC transfer operations
Enables RIIC transfer operations.

End

**Figure 14  IIC Initialization**

IIC_Destroy

RIIC reset
Reset all RIIC registers and its internal state by setting ICCR1.ICE to 0 and ICCR1.IICRST to 1.

End

**Figure 15  IIC Termination Processing**
IIC_EepWrite

Is this device communicating?

Yes

Checks the internal mode. Cancels the write if a communication operation is in progress.

return RIIC_MODE_ERROR

Store the argument in the IIC buffer

No

Stores the argument in the IIC buffer.

Is the bus free?

No

Checks the IIC bus state. Cancels the write if the bus is busy.

return RIIC_BUS_BUSY

Yes

Sets up internal RAM mode. Sets the internal mode and the counter.

return RIIC_OK

Set internal RAM mode and the counter

Generate a start condition

Figure 16  EEPROM Write Start Processing

IIC_RandomRead

Is this device communicating?

Yes

Checks the internal mode. Cancels the read if a communication operation is in progress.

return RIIC_MODE_ERROR

No

Stores the argument in the IIC buffer.

Is the argument legal?

No

Checks the argument value. Cancels the read if the value is illegal.

return RIIC_PRM_ERROR

Yes

Is the bus free?

No

Checks the IIC bus state. Cancels the read if the bus is busy.

return RIIC_BUS_BUSY

Yes

Sets up internal RAM mode. Sets the internal mode and the counter.

return RIIC_OK

Set internal RAM to internal mode and set the counter

Generate a start condition

Figure 17  EEPROM Read Start Processing
Enables the timeout detection interrupt and furthermore, if a timeout detection interrupt has occurred, handles the timeout detection interrupt.

Enables the arbitration lost interrupt and furthermore, if an arbitration lost interrupt has occurred, handles the arbitration lost interrupt.

Enables the stop condition detection interrupt and furthermore, if a stop condition detection interrupt has occurred, handles the stop condition detection interrupt.

Enables the NACK detection interrupt and furthermore, if a NACK detection interrupt has occurred, handles the NACK detection interrupt.

Enables the start condition detection interrupt and furthermore, if a start condition detection interrupt has occurred, handles the start condition detection interrupt.
**Figure 20**  Timeout Detection Interrupt

```
IIC_EEI_IntTimeOut
  IIC_GenClkSP
  End
```

Stop condition generation processing used when an abnormality occurs

**Figure 21**  Arbitration Lost Detection Interrupt

```
IIC_EEI_IntAL
  IIC_GenClkSP
  End
```

Stop condition generation processing used when an abnormality occurs
Clear NACKF here when NACK detected.
Clear the stop flag.

Interrupt settings:
- Enable the timeout interrupt
- Enable the arbitration lost interrupt
- Disable the start condition detection interrupt
- Enable the stop condition detection interrupt
- Enable the NACK received interrupt
- Enable the receive data full interrupt
- Disable the transmission complete interrupt
- Enable the transmitted empty interrupt

Figure 22 Stop Condition Detection Interrupt

Figure 23 Start Condition Detected Interrupt
Figure 24  NACK Detection Interrupt

Figure 25  Receive Data Full Interrupt
Figure 26  EEPROM Read Processing (Master Reception Section)

Figure 27  Transmit Data Empty Interrupt
Figure 28  EEPROM Write Processing

Figure 29  EEPROM Read Processing (Master Transmission Section)

Figure 30  Transmission Complete Interrupt
**Figure 31** Transmission Complete Processing after EEPROM Write Processing

**Figure 32** Transmission Complete Processing after EEPROM Read Processing
IIC_GenClkSP

Internal reset

Set up timeout operation

Wait for a fixed period

Enable timeouts

SCL = High? No

Yes IIC_Error

Set up master mode

SDA = Low? No

Yes Loop

Cnt = 0; cnt < 10; cnt++

SDA = Low? No

Yes Generate 1 clock cycle

Has the one clock cycle completed? No

Yes Did a timeout occur? No

Yes IIC_Error

Loop

Is the bus busy? No

Yes IIC internal reset

Generate a stop condition

Enable MST/TRS protection

Set up timeout operation

End

- Stops output to SCL and SDA and resets the internal state.
- Generates timeouts on SCL = high or SCL = low.
- Waits for the SCL and SDA release times due to the internal reset.
- Enables timeouts.
- Even though an internal reset was performed, SCL was set high while another device was holding it low.
- Clears MST/TRS protection and sets up master transmission mode.
- Processing when the remote device is holding SDA low.
  - One possibility is that a bit displacement occurred between the RIIC module and the remote device. In that case, it may be possible to release SCL by generating a few clock cycles.
  - Use the ICCR1.CLO bit and generate one clock cycle at a time. Each time, check whether SDA is high or nine iterations have been performed.
- If the bus is busy, generates a stop condition. In all other cases, performs an internal reset and switches from master transmission mode to slave reception mode (idle mode).
- Enables MST/TRS protection.
- Generates timeouts when SCL is high.

Figure 33 Stop Condition Generation Processing when an Abnormal State Occurs
Figure 34  Error Handling
5. Reference Documents

- Hardware Manual
  RX610 Group User’s Manual: Hardware
  (The latest version can be downloaded from the Renesas Electronics Web site.)

- Software Manual
  RX Family User’s Manual: Software
  (The latest version can be downloaded from the Renesas Electronics Web site.)

- Development Environment Manual
  RX Family C/C++ Compiler Package User’s Manual
  (The latest version can be downloaded from the Renesas Electronics Web site.)

- Technical Updates
  (The latest information can be downloaded from the Renesas Electronics Web site.)
Website and Support

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

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

All trademarks and registered trademarks are the property of their respective owners.
<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Jan.18.11</td>
<td>—</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of MPU/MCU Products

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

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

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

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

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

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

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

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

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

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

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

6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.

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

- Standard: Computer, 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, train, ship, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for the support.
- Specific: Aircraft, aerospace equipment, submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implants, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.

8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.

9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products system manufactured by you.

10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.

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

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

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

(2) “Renesas Electronics products” means any product developed or manufactured by or for Renesas Electronics.