Introduction

This application note describes how to control an external EEPROM by using the I²C bus features. The sample code shown in this application note writes and reads 4-byte (word) data to and from the EEPROM connected to the I²C bus. In addition, the sample code compares the written data and read data and uses the result of comparison to control the LED display.

Target Devices

V850ES/JC3-L
V850ES/JE3-L
V850ES/JF3-L
V850ES/JG3-L
V850ES/JG3-L USB

When applying this application note to other microcontrollers, make the necessary changes according to the specifications of the microcontroller and verify them thoroughly.
Contents

1. Specifications ................................................................................................................. 3

2. Conditions Under Which Operation Has Been Verified ........................................... 4
   2.1 Notes on using CubeSuite+ code generator ................................................................. 4

3. Related Application Notes ............................................................................................. 4

4. Hardware .......................................................................................................................... 5
   4.1 Hardware configuration ............................................................................................... 5
   4.2 Pins used ..................................................................................................................... 5

5. Software .......................................................................................................................... 6
   5.1 Operation overview ..................................................................................................... 6
      5.1.1 EEPROM write processing .................................................................................... 7
      5.1.2 EEPROM read processing ................................................................................... 8
   5.2 Option byte settings ................................................................................................... 9
   5.3 Constants ................................................................................................................... 9
   5.4 Variables ..................................................................................................................11
   5.5 Functions ..................................................................................................................12
   5.6 Function specifications .............................................................................................13
   5.7 Flowcharts ...............................................................................................................18
      5.7.1 IIC0 initialization function ...................................................................................19
      5.7.2 main function .....................................................................................................20
      5.7.3 Selection of EEPROM .......................................................................................22
      5.7.4 IIC0 start function .............................................................................................22
      5.7.5 Start condition generation function .................................................................23
      5.7.6 Stop condition generation function ...................................................................23
      5.7.7 EEPROM write processing ...............................................................................24
      5.7.8 EEPROM address check processing .................................................................25
      5.7.9 Slave address calculation ..................................................................................26
      5.7.10 Waiting for writing to EEPROM to finish .......................................................27
      5.7.11 EEPROM read processing ..............................................................................28
      5.7.12 Waiting for reading from EEPROM to finish ................................................29
      5.7.13 Finishing slave address transmission .............................................................29
      5.7.14 Finishing upper address transmission ............................................................30
      5.7.15 Restart processing ............................................................................................30
      5.7.16 Starting data reception .....................................................................................31
      5.7.17 Data reception ..................................................................................................32
      5.7.18 Reception of final data .....................................................................................33
      5.7.19 Starting data transmission ...............................................................................33
      5.7.20 Data transmission .............................................................................................34
      5.7.21 MD_INTIIC0 interrupt servicing ....................................................................35

6. Sample Code ....................................................................................................................38

7. Reference Documents ....................................................................................................38
1. Specifications

The sample code shown in this application note uses the I²C bus features to write and read data to and from an external EEPROM connected to the I²C bus. In addition, the sample code uses the result of comparing the written data and read data to control the LED display.

- Interrupt functions are used to implement API operations for controlling communication by using the I²C bus features.
- Multiple I²C communication control parameters are provided to enable control of different EEPROM sizes. The EEPROM size can be selected by the user, from 2 to 512 Kb. In this sample code, a 16 Kb EEPROM, R1EX24016A, and the communication control parameters for it are used to execute communication.
- The sample code writes 4-byte data to a specified EEPROM address by using I²C communication and then reads the written data. The EEPROM can be accessed in block units, where one block consists of 4 bytes.
- Finally, the sample code compares the written data and read data to judge whether they match. If the data matches, the sample code turns on LED1 by controlling the relevant port. If the data does not match, the sample code turns on LED2.

Table 1.1 shows the peripheral functions used and their applications, and Figure 1.1 shows an overview of I²C communication.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>I²C00</td>
<td>I²C master transmission/reception implemented by using the SCL00 and SDA00 pins</td>
</tr>
</tbody>
</table>

Table 1.1 Peripheral functions used and their applications

V850ES/Jx3-L (master) | EEPROM (slave)
=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=~
2. Conditions Under Which Operation Has Been Verified

The operation of the sample code shown in this application note has been verified under the conditions shown below.

Table 2.1  Conditions under which operation has been verified

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcontroller used</td>
<td>V850ES/JG3-L USB (μPD70F3796GC)</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>• CPU clock: 16 MHz</td>
</tr>
<tr>
<td></td>
<td>• Peripheral clock: 16 MHz</td>
</tr>
<tr>
<td></td>
<td>• Main clock oscillation frequency: 6 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>3.3 V (2.7 V to 3.6 V, when the CPU operates on 16 MHz)</td>
</tr>
<tr>
<td>Integrated development environment</td>
<td>CubeSuite+ V1.03.00 made by Renesas Electronics</td>
</tr>
<tr>
<td>C compiler</td>
<td>CA850 V3.50 made by Renesas Electronics</td>
</tr>
<tr>
<td>Board used</td>
<td>QB-V850ESJG3LUSB-TB + external EEPROM (R1EX24016)</td>
</tr>
</tbody>
</table>

Caution This sample code can be used with V850ES/Jx3-L devices. If you plan to use a device that does not conform to the conditions under which operation has been verified, be sure to choose the operating frequency and operating voltage that suit your device, by using features such as the code generator.

2.1 Notes on using CubeSuite+ code generator

This sample code uses the features of the CubeSuite+ code generator, but some source files have been added or deleted. When generating code for the project described in this document, perform the following:

- Remove CG_serial_user.c from the project.
- Add IIC_EEPROM.c and CG_lk.dir (existing files) to the project.

3. Related Application Notes

Related application notes are shown below. Also refer to these documents when using this application note.

V850ES/JG3-L Initialization (U19479EJ) Application Note
4. Hardware

4.1 Hardware configuration

Figure 4.1 shows an example of the hardware configuration described in this application note.

Cautions
1. This circuit diagram is simplified to show an overview of the circuit connections. When designing your actual circuit, connect pins appropriately so as to satisfy the electrical specifications.
2. Make the potential of the EV_{DD} pin and AV_{REF} pin the same as V_{DD}.
3. Make the potential of the EV_{SS} pin the same as GND.
4. Connect REGC to GND via a capacitor (recommended value: 4.7 μF).
5. Connect the FLMD0 pin to GND in normal operating mode.

4.2 Pins used

Table 4.1 shows the pins used and their roles.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P39/SCL00</td>
<td>I/O</td>
<td>Serial clock output pin for I²C00</td>
</tr>
<tr>
<td>P38/SDA00</td>
<td>I/O</td>
<td>Serial data transmission/reception pin for I²C00</td>
</tr>
<tr>
<td>PDH1</td>
<td>I/O</td>
<td>LED1 pin (LED1 turns on when data comparison results in match)</td>
</tr>
<tr>
<td>PDH0</td>
<td>I/O</td>
<td>LED2 pin (LED2 turns on when data comparison results in mismatch)</td>
</tr>
</tbody>
</table>
5. Software

5.1 Operation overview

The sample code in this application note controls (reads and writes) the EEPROM by using \( \text{I}^2\text{C} \) master transmission/reception via \( \text{I}^2\text{C}00 \). In addition, the sample code compares the data written to and read from the EEPROM and lights the LEDs accordingly.

Setting conditions:

- Dividing the \( \text{I}^2\text{C}0 \) clock is enabled and the division factor is \( f_{CX}/2 \).
- The operating mode is high-speed mode. The digital filter is enabled.
- \( f_{CLK} \) is specified as the transfer clock.
- The local address is 80H.
- Generating a start condition is enabled after operation is enabled.
- Generating an interrupt request upon detection of a stop condition is disabled.
- An interrupt request is set to be generated at the falling edge of the 9th clock cycle.
- The acknowledge signal is enabled.
- The P39/SCL00 pin is used to output the transfer clock and the P38/SDA00 pin is used to transmit and receive data.

1. Specify the \( \text{I}^2\text{C}00 \) initial settings.
2. Set the interval counted by timer TMM0 to 100 \( \mu \text{s} \). TMM0 is used to count the wait cycles when writing to the EEPROM.
3. Specify the communication control parameters for the EEPROM used (16 Kb EEPROM).
4. Enable \( \text{I}^2\text{C}00 \).
5. Create 4-byte data (00H, 01H, 02H, and 03H).
6. Specify the slave address (A0H) and the address to be accessed in the EEPROM (5th block (014H)) for the EEPROM access parameter (structure g_PARAT).
7. Write data to the EEPROM.
8. Read the data written to the EEPROM.
9. Compare the written data and read data and light the LEDs according to the result (match or mismatch).

Caution  This sample code simply shows an example of controlling the EEPROM (R1EX24016) connected to the \( \text{I}^2\text{C} \) bus by using \( \text{I}^2\text{C}00 \) of the V850ES/JG3-L. If you use another channel or EEPROM, thoroughly evaluate it before use.

Note  If an attempt is made to release the bus by generating a stop condition on the \( \text{I}^2\text{C} \) bus, if the EEPROM, which is serving as a slave, is pulling the SDA signal low, the stop condition cannot be generated.

In this case, control the SCL signal by using the program to generate a pseudo \( \text{I}^2\text{C} \) bus clock for up to nine clock cycles; this stops the EEPROM driving the SDA signal (normal operation) and the SDA signal can then be pulled high.
5.1.1 EEPROM write processing

Figure 5.1 shows the status transitions during EEPROM write processing. The statuses are defined by the values of the variable $g\_comstatus$ (such as TRANSMIT (0x80)). During I²C communication, the current status is determined by referencing the value of $g\_comstatus$ during the processing of the transfer complete interrupt function, and the program branches to the next processing based on this value.

![Status transition diagram](image)

**Caution** Error handling is omitted in the above figure. The parameter below each state is the value of $g\_comstatus$.

During I²C communication, the current status is determined during processing of the interrupt function and the program branches to the next processing based on this value.

### Transfer complete interrupt function

$g\_comstatus = \text{TxHiADDR}$

- Transmits the lower address ($g\_comstatus \rightarrow \text{TxLoADDR}$)
- Transmits data ($g\_comstatus \rightarrow \text{TxData}$)

**Figure 5.1 Status transition during EEPROM write processing**
5.1.2 EEPROM read processing

Figure 5.2 shows the status transitions during EEPROM read processing. Like EEPROM write processing, the statuses are defined by the values of the variable `g_comstatus` (such as RECEIVE (0xC0)). EEPROM read processing consists of two main sections: specifying the cell address (by informing EEPROM of the EEPROM address to be read) and reading data (by specifying the master as the agent to receive data and generating a restart condition).

**Caution** Error handling is omitted in the above figure. The parameter below each state is the value of `g_comstatus`.

![Figure 5.2 Status transition during EEPROM read processing](image)

**Specifying the cell address**

1. **Transmission ready**
   - Starts reception, calculates the slave address, generates the start condition, and then transmits the slave address.

2. **Wait for slave address transmission to finish**
   - RECEIVED (0xC0)
   - Receives ACK and transmits the address if the EEPROM size is 16 Kb or less.

3. **Wait for upper address transmission to finish**
   - RxHiADDR (0xC1)
   - Receives ACK and transmits the upper address if the EEPROM size is 32 Kb or more.

4. **Wait for lower address transmission to finish**
   - RxLoADDR (0xC2)
   - Finishes transmission of the upper address and then transmits the lower address.

**Reading data**

5. **Wait for slave address transmission to finish**
   - ReStart (0xC3)
   - Finishes transmission of the slave address and then starts data reception.

6. **Wait for data reception to finish**
   - RxData (0xE0)
   - Finishes data reception, sets NACK, and starts the next data reception if there is one more data unit to receive.

7. **Wait for reception of the final data to finish**
   - RxLAST (0xE1)
   - Finishes data reception and finalizes completion, and then generates the stop condition.

8. **Wait for slave address transmission to finish**
   - RECEIVED (0xC0)
   - Finishes data reception, sets NACK, and starts the next data reception if there is more than one data unit left to receive.
5.2 Option byte settings

Table 5.1 shows the option byte settings.

<table>
<thead>
<tr>
<th>Address</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000007AH</td>
<td>00000110B</td>
<td>Disable stopping the on-chip oscillator by using software (oscillation stabilization time: 10.92 ms (f_X = 6 MHz))</td>
</tr>
<tr>
<td>0000007BH to 0000007FH</td>
<td>00000000B</td>
<td>Specify signal for addresses 007BH to 007FH.</td>
</tr>
</tbody>
</table>

5.3 Constants

Table 5.2 shows the constants used in sample code.

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOCK_SIZE</td>
<td>4</td>
<td>EEPROM data block access unit (bytes)</td>
</tr>
<tr>
<td>MEMORY_2K</td>
<td>0x0001</td>
<td>Capacity of 2 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_4K</td>
<td>0x0002</td>
<td>Capacity of 4 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_8K</td>
<td>0x0004</td>
<td>Capacity of 8 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_16K</td>
<td>0x0008</td>
<td>Capacity of 16 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_32K</td>
<td>0x0010</td>
<td>Capacity of 32 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_64K</td>
<td>0x0020</td>
<td>Capacity of 64 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_128K</td>
<td>0x0040</td>
<td>Capacity of 128 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_256K</td>
<td>0x0080</td>
<td>Capacity of 256 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>MEMORY_512K</td>
<td>0x0100</td>
<td>Capacity of 512 Kb EEPROM (in 256-byte units)</td>
</tr>
<tr>
<td>BLOCK_2K</td>
<td>See right</td>
<td>MEMORY_2K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_4K</td>
<td>See right</td>
<td>MEMORY_4K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_8K</td>
<td>See right</td>
<td>MEMORY_8K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_16K</td>
<td>See right</td>
<td>MEMORY_16K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_32K</td>
<td>See right</td>
<td>MEMORY_32K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_64K</td>
<td>See right</td>
<td>MEMORY_64K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_128K</td>
<td>See right</td>
<td>MEMORY_128K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_256K</td>
<td>See right</td>
<td>MEMORY_256K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>BLOCK_512K</td>
<td>See right</td>
<td>MEMORY_512K * 256 / BLOCK_SIZE</td>
</tr>
<tr>
<td>ADDR0BIT</td>
<td>0b00000000</td>
<td>Specifies that slave addresses are not be used as cell addresses</td>
</tr>
<tr>
<td>ADDR1BIT</td>
<td>0b00000001</td>
<td>Specify A8 by using bit 1 of the slave address</td>
</tr>
<tr>
<td>ADDR2BIT</td>
<td>0b00000011</td>
<td>Specify A9 and A8 by using bits 2 and 1 of the slave address</td>
</tr>
<tr>
<td>ADDR3BIT</td>
<td>0b00000111</td>
<td>Specify A10 to A8 by using bits 3 to 1 of the slave address</td>
</tr>
<tr>
<td>I2C_OK</td>
<td>0x00</td>
<td>Successful</td>
</tr>
<tr>
<td>PARA_ERR</td>
<td>0x20</td>
<td>Parameter error</td>
</tr>
<tr>
<td>COMP_ERR</td>
<td>0x21</td>
<td>Compare match error in read or written data</td>
</tr>
<tr>
<td>NO_ACK1</td>
<td>0x40</td>
<td>No ACK response for the slave address</td>
</tr>
<tr>
<td>NO_ACK2</td>
<td>0x41</td>
<td>No ACK response for the EEPROM address</td>
</tr>
<tr>
<td>NO_ACK3</td>
<td>0x42</td>
<td>No ACK response for the transmitted data</td>
</tr>
<tr>
<td>SVAMSK</td>
<td>0b11111110</td>
<td>Data for masking bit 0 of the slave address</td>
</tr>
</tbody>
</table>
Table 5.2 Constants used in sample code (2/2)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1EX24002A</td>
<td>0x00</td>
<td>Specify the EEPROM to be used</td>
</tr>
<tr>
<td>R1EX24004A</td>
<td>0x01</td>
<td>These constants are defined by the enumeration constant</td>
</tr>
<tr>
<td>R1EX24008A</td>
<td>0x02</td>
<td>eeprom_name and are used to allow the eeprom_info</td>
</tr>
<tr>
<td>R1EX24016A</td>
<td>0x03</td>
<td>structure EEPROM_ADDRESS to reference EEPROM</td>
</tr>
<tr>
<td>R1EX24032A</td>
<td>0x04</td>
<td>parameters.</td>
</tr>
<tr>
<td>R1EX24064A</td>
<td>0x05</td>
<td></td>
</tr>
<tr>
<td>R1EX24128B</td>
<td>0x06</td>
<td></td>
</tr>
<tr>
<td>R1EX24256B</td>
<td>0x07</td>
<td></td>
</tr>
<tr>
<td>R1EX24512B</td>
<td>0x08</td>
<td></td>
</tr>
</tbody>
</table>
## 5.4 Variables

Table 5.3 shows the global variables.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Variable Name</th>
<th>Description</th>
<th>Function That Uses This Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure</td>
<td>g_PARAI</td>
<td>Parameter for specifying EEPROM access</td>
<td>main()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td>Operation information/result flag</td>
<td>check_EEPROM_Addr()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_EEPROM_R()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_EEPROM_wait_read()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_Tx_addr1()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_Tx_addr2()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_Rx_RST()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_RxData_ST()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_RxData()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_Rx_LastData()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_EEPROM_W()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_EEPROM_wait_write()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_TxDataST()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_EEPROM_W()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>R_IIC0_TxDataST()</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_comstatus</td>
<td></td>
<td>MD_INTIIC0()</td>
</tr>
<tr>
<td>uint8_t array</td>
<td>g_data_buffer</td>
<td>Write data buffer</td>
<td>main()</td>
</tr>
<tr>
<td>(BLOCK_SIZE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint8_t array</td>
<td>g_data_buffer</td>
<td>Read data buffer</td>
<td>main()</td>
</tr>
<tr>
<td>(BLOCK_SIZE)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_data_counte</td>
<td>Data counter</td>
<td></td>
</tr>
<tr>
<td>rter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_celladdr</td>
<td>EEPROM internal address</td>
<td></td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_eeprom_type</td>
<td>Number of EEPROM used</td>
<td></td>
</tr>
<tr>
<td>Structure</td>
<td>g_PARAA</td>
<td>Parameter for accessing EEPROM</td>
<td></td>
</tr>
<tr>
<td>uint8_t</td>
<td>slaveaddr;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td>block_num;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structure</td>
<td>g_PARAC</td>
<td>Parameter for accessing work EEPROM (a copy of g_PARAA)</td>
<td>check_EEPROM_Addr()</td>
</tr>
<tr>
<td>uint16_t</td>
<td>rom_size;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint16_t</td>
<td>total_block;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint8_t</td>
<td>addr_mask;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Structure</td>
<td>EEPROM_Info</td>
<td>Variable for saving parameters for the EEPROM used (for processing)</td>
<td>R_device_select()</td>
</tr>
<tr>
<td>uint16_t</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 5.4 shows the functions.

### Table 5.4 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_device_select</td>
<td>This function specifies the EEPROM to be used.</td>
</tr>
<tr>
<td>R_EEPROM_R</td>
<td>This function reads data from the EEPROM based on the structure pointed to by the pointer passed by the EEPROM access parameter.</td>
</tr>
<tr>
<td>R_EEPROM_wait_read</td>
<td>This function is used to wait for EEPROM reading to finish.</td>
</tr>
<tr>
<td>R_EEPROM_W</td>
<td>This function writes data to the EEPROM based on the structure pointed to by the pointer passed by the EEPROM access parameter.</td>
</tr>
<tr>
<td>R_EEPROM_wait_write</td>
<td>This function is used to wait for EEPROM writing to finish.</td>
</tr>
<tr>
<td>check_EEPROM_Addr</td>
<td>This function checks whether the address specified by the parameter is within the EEPROM size. If the address is within the EEPROM size, this function copies the access parameter.</td>
</tr>
<tr>
<td>get_slave_Addr</td>
<td>This function incorporates the upper address of a cell into the slave address when using a 4 Kb to 16 Kb EEPROM.</td>
</tr>
<tr>
<td>R_IIC0_Tx_addr1</td>
<td>This function ends transmission to the slave address and transmits the EEPROM cell address.</td>
</tr>
<tr>
<td>R_IIC0_Tx_addr2</td>
<td>This function transmits the lower address of an EEPROM cell if it is 2 bytes.</td>
</tr>
<tr>
<td>R_IIC0_Rx_RST</td>
<td>After transmitting the EEPROM cell address, this function restarts communication to read the data.</td>
</tr>
<tr>
<td>R_IIC0_RxData_ST</td>
<td>This function writes dummy data to the IIC0 register and starts data reception.</td>
</tr>
<tr>
<td>R_IIC0_RxData</td>
<td>This function stores the received data in the buffer and starts the next data reception. In addition, this function disables the acknowledge signal before receiving the final data.</td>
</tr>
<tr>
<td>R_IIC0_Rx_LastData</td>
<td>This function stores the data received last, generates the stop condition, and then ends reception processing.</td>
</tr>
<tr>
<td>R_IIC0_TxDataST</td>
<td>This function starts transmission of data to be written to the EEPROM.</td>
</tr>
<tr>
<td>R_IIC0_TxData</td>
<td>If there is more data to send when transmission finishes, this function starts data transmission again. When all data units have been transmitted, this function generates the stop condition and instructs the EEPROM to write data.</td>
</tr>
<tr>
<td>R_IIC00_StartCondition</td>
<td>This function generates a start condition and waits for the start condition to be detected.</td>
</tr>
<tr>
<td>R_IIC00_StopCondition</td>
<td>This function generates a stop condition and waits for the stop condition to be detected.</td>
</tr>
<tr>
<td>MD_INTIIC0</td>
<td>This function checks the ACK response from the slave by using the IIC00 transfer complete interrupt and assigns the next processing according to the communication status.</td>
</tr>
<tr>
<td>R_IIC00Start</td>
<td>This function enables IIC00.</td>
</tr>
</tbody>
</table>
## 5.6 Function specifications

This section shows the specifications of the functions used in the sample code.

### R_device_select

<table>
<thead>
<tr>
<th>Overview</th>
<th>Specification of EEPROM to be used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_macrodriver.h</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>MD_STATUS R_device_select(enum eeprom_name);</td>
</tr>
<tr>
<td>Description</td>
<td>This function copies the EEPROM parameter specified by the parameter to the EEPROM_Info structure.</td>
</tr>
<tr>
<td>Parameters</td>
<td>Name of EEPROM Name defined by enumeration constant eeprom_name</td>
</tr>
<tr>
<td>Return value</td>
<td>I2C_OK: Successful</td>
</tr>
<tr>
<td></td>
<td>PARA_ERR: Incorrect name was specified</td>
</tr>
<tr>
<td>Remark</td>
<td>None</td>
</tr>
</tbody>
</table>

### R_IIC00_Start

<table>
<thead>
<tr>
<th>Overview</th>
<th>Enabling IIC00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_macrodriver.h</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void R_IIC00_Start(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function enables IIC00.</td>
</tr>
<tr>
<td>Parameters</td>
<td>None</td>
</tr>
<tr>
<td>Return value</td>
<td>None</td>
</tr>
<tr>
<td>Remark</td>
<td>None</td>
</tr>
</tbody>
</table>

### R_EEPROM_R

<table>
<thead>
<tr>
<th>Overview</th>
<th>Reading data from the EEPROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_macrodriver.h</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void R_EEPROM_R(struct eeprom_paraA16 *PARA);</td>
</tr>
<tr>
<td>Description</td>
<td>This function reads data from the address specified by the structure pointed to by the parameter.</td>
</tr>
<tr>
<td>Parameters</td>
<td>*PARA Pointer to structure eeprom_paraA16</td>
</tr>
<tr>
<td>Return value</td>
<td>None</td>
</tr>
<tr>
<td>Remark</td>
<td>Processing to wait for reading to finish is performed by R_EEPROM_wait_read.</td>
</tr>
</tbody>
</table>

### R_EEPROM_wait_read

<table>
<thead>
<tr>
<th>Overview</th>
<th>Waiting for reading from EEPROM to finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_macrodriver.h</td>
</tr>
<tr>
<td></td>
<td>r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>MD_STATUS R_EEPROM_wait_read (void);</td>
</tr>
<tr>
<td>Description</td>
<td>This function waits for reading started by R_EEPROM_R to finish.</td>
</tr>
<tr>
<td>Parameters</td>
<td>None</td>
</tr>
<tr>
<td>Return value</td>
<td>I2C_OK: Data was read successfully.</td>
</tr>
<tr>
<td></td>
<td>PARA_ERR: The address specified by the parameter was out of the EEPROM address range.</td>
</tr>
<tr>
<td></td>
<td>NO_ACK1: No ACK response for the slave address</td>
</tr>
<tr>
<td></td>
<td>NO_ACK2: No ACK response for the EEPROM address</td>
</tr>
<tr>
<td>Remark</td>
<td></td>
</tr>
</tbody>
</table>
R_EEPROM_W
Overview Writing data to EEPROM
Header r_cg_macrodriver.h
r_cg_userdefine.h
Declaration void R_EEPROM_W(struct eeprom_paraA16 *PARA);
Description This function writes data to the address specified by the structure pointed to by the parameter.
Parameters *PARA Pointer to structure eeprom_paraA16
Return value None
Remark Processing to wait for writing to finish is performed by R_EEPROM_wait_write.

R_EEPROM_wait_write
Overview Waiting for writing to EEPROM to finish
Header r_cg_macrodriver.h
r_cg_userdefine.h
Declaration MD_STATUS R_EEPROM_wait_write(void);
Description This function waits for writing started by R_EEPROM_W to finish.
Parameters None
Return value I2C_OK: Data was written successfully.
PARA_ERR: The address specified by the parameter was out of the EEPROM address range.
NO_ACK1: No ACK response for the slave address
NO_ACK2: No ACK response for the EEPROM address
NO_ACK3: No ACK response for the transmitted data
Remark None

check_EEPROM_Addr
Overview Checking the EEPROM access area (block number)
Header r_cg_macrodriver.h
r_cg_userdefine.h
Declaration static MD_STATUS check_EEPROM_Addr(struct eeprom_paraA16 *PARA);
Description This function checks whether the area to be read or written specified by the EEPROM access parameter is within the EEPROM.
Parameters *PARA Pointer to structure eeprom_paraA16
Return value I2C_OK: The area to be accessed is within the EEPROM size.
PARA_ERR: The area to be accessed is not within the EEPROM size.
Remark None

get_slave_Addr
Overview Calculating the EEPROM slave address
Header r_cg_userdefine.h
Declaration static void get_slave_Addr(void);
Description This function modifies the slave address by using the upper 3 bits of the cell address when using a 4 Kb to 16 Kb EEPROM.
Parameters None
Return value None
Remark This function modifies slaveaddr, a member of the g_PARAC structure, by using the value of another member eepromaddr.
### R_IIC0_Tx_addr1

**Overview**
Transmitting the EEPROM address

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_Tx_addr1(void);

**Description**
This function transmits the upper bytes of the cell address when using an EEPROM of 32 Kb or more. When using an EEPROM of 16 Kb or less, this function transmits a one-byte cell address.

**Parameters**
None

**Return value**
None

**Remark**
Used during MD_INTIIC0 interrupt servicing

### R_IIC0_Tx_addr2

**Overview**
Transmitting the lower address of the EEPROM cell

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_Tx_addr2(void);

**Description**
This function transmits the lower bytes of the cell address when using an EEPROM of 32 Kb or more.

**Parameters**
None

**Return value**
None

**Remark**
Used during MD_INTIIC0 interrupt servicing

### R_IIC0_Rx_RST

**Overview**
Restarting reception to read data from EEPROM

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_Rx_RST(void);

**Description**
This function restarts transfer in reception mode in order to read data.

**Parameters**
None

**Return value**
None

**Remark**
Used during MD_INTIIC0 interrupt servicing

### R_IIC0_RxData_ST

**Overview**
Starting data reception

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_RxData_ST(void);

**Description**
This function starts receiving data (by writing dummy data to the IIC0 register) once the slave address has been transmitted in reception mode.

**Parameters**
None

**Return value**
None

**Remark**
Used during MD_INTIIC0 interrupt servicing

### R_IIC0_RxData

**Overview**
Data reception

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_RxData (void);

**Description**
This function stores the received data in the buffer and starts the next data reception.

**Parameters**
None

**Return value**
None

**Remark**
Used during MD_INTIIC0 interrupt servicing
### R_IIC0_Rx_LastData

**Overview**
Finishing reception of final data

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_Rx_LastData (void);

**Description**
This function stores the data received last, generates the stop condition, and then ends reception processing.

**Parameters** None

**Return value** None

**Remark** Used during MD_INTIIC0 interrupt servicing

### R_IIC0_TxDataST

**Overview**
Starting data transmission

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_TxDataST (void);

**Description**
This function starts data transmission once the EEPROM address has been transmitted.

**Parameters** None

**Return value** None

**Remark** Used during MD_INTIIC0 interrupt servicing

### R_IIC0_TxData

**Overview**
Data transmission

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC0_TxData (void);

**Description**
Once 1-byte data has been transmitted, this function transmits the next data. When all data units to be written have been transmitted, this function generates the stop condition and instructs the EEPROM to write data. If there is data left to write, this function transmits the data.

**Parameters** None

**Return value** None

**Remark** Used during MD_INTIIC0 interrupt servicing

### R_IIC00_StartCondition

**Overview**
Generating a start condition

**Header**
r_cg_userdefine.h

**Declaration**
static void R_IIC00_StartCondition(void);

**Description**
This function generates a start condition and waits for the start condition to be detected.

**Parameters** None

**Return value** None

**Remark** None

### R_IIC00_StopCondition

**Overview**
Generating a stop condition

**Header**
r_cg_macrodriver.h
r_cg_userdefine.h

**Declaration**
Void R_IIC00_StopCondition(void)

**Description**
This function generates a stop condition and waits for the stop condition to be detected.

**Parameters** None

**Return value** None

**Remark** None
**MD_INTIIC0**

<table>
<thead>
<tr>
<th>Overview</th>
<th>IIC0 transfer complete interrupt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>r_cg_userdefine.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>__interrupt void MD_INTIIC0(void);</td>
</tr>
<tr>
<td>Description</td>
<td>Triggered by the INTIIC0 interrupt request, this function executes the appropriate processing according to the communication status. When a NACK response from the slave is detected in a state other than finishing read from the EEPROM, this function sets the error flag of the corresponding g_comstatus bit.</td>
</tr>
<tr>
<td>Parameters</td>
<td>None</td>
</tr>
<tr>
<td>Return value</td>
<td>None</td>
</tr>
<tr>
<td>Remark</td>
<td></td>
</tr>
</tbody>
</table>
5.7 Flowcharts

Figure 5.3 shows an overview of the processing flow used in this application note.

Remark Processing of functions other than those for IIC00 is omitted in the flowchart of the initialization function.

![Flowchart]

Figure 5.3 Overview of processing flow
5.7.1 IIC00 initialization function

Figure 5.4 shows the flow of the initialization function processing.

```plaintext
IIC0_Init()

Setup for initialization

Specify IIC00 clock division ratio

Specify IIC00 transfer clock

Specify local address

Enable initial start

Disable communication reservation

Disable interrupts when stop condition detected

Set interrupt request at 9th clock edge

Enable acknowledge signal

Unmask interrupts

Set up port output

return
```

IICE0 bit ← 0: Disable IIC0
IICMK0 bit ← 1: Mask interrupts
IICIF0 bit ← 0: Clear the interrupt flag
IICPR0n bit ← 1: Interrupt priority 7

OCKS0 register ← 10H: \( f_{\text{I}^2\text{C}} \) division ratio (fxx/2)

IICCL0 register ← 0CH
IICX0 register ← 01H
Operating mode: High-speed mode
Digital filter: On
Transfer clock: 333 kHz

SVA0 register ← 80H: Specify local address

STCEN0 bit ← 1:
Enables generating a start condition even if no stop condition has been detected

IICRSV0 bit ← 1

SPIE0 bit ← 0

WTIM0 bit ← 1

ACHE0 bit ← 1

IICMK0 bit ← 0

Specifies that the pins used for SDA and SCL are to be used as their alternate function in N-ch open-drain output mode.

Figure 5.4 IIC0 initialization function
5.7.2 main function

Figures 5.5 and 5.6 show the flow of the main processing. The main function tests reading from and writing to a 16 Kb EEPROM.

```
main()

Select EEPROM
    R_device_select()

Enable IIC00
    R_IIC00_Start()

Set data to transmit

Specify EEPROM writing parameter

Write data
    R_EEPROM_W()

Wait for writing to finish
    R_EEPROM_wait_write()

Error detected?
    Yes
        NOP instruction
    No

Specify EEPROM reading parameter

Read data
    R_EEPROM_R()

Wait for reading to finish
    R_EEPROM_wait_read()
```

- **Specify the EEPROM to be used.**
- **Enables IIC00.**
- Sets the address increment pattern (00H, 01H, ...) for 4 bytes (equivalent to BLOCK_SIZE) to the buffer (g_data_bufferW).
- Specifies the parameter used to write data in the buffer (g_data_bufferW) to the EEPROM in the g_PARAI structure.
- Calls the function to write data to the EEPROM by using the pointer to the g_PARAI structure as a parameter.
- Waits for EEPROM writing to finish.
- **If the write results in an error, the NOP instruction loops.**
- Specifies the parameter used to read and copy the written data to the buffer (g_data_bufferR) in the g_PARAI structure.
- Calls the function to read data to the EEPROM by using the pointer to the g_PARAI structure as a parameter.
- **Waits for EEPROM read to finish.**
If the read results in an error, the NOP instruction loops.

- Error detected? (If YES, proceed to NOP instruction)
- If NO, successively compares data to judge whether the data in the write buffer \( g_{\text{data_bufferW}} \) and read buffer \( g_{\text{data_bufferR}} \) match. If the data does not match, an error flag \( \text{COMP_ERR} \) is set.

- Apply the result of comparison to LED
  - Turns on LED1 if the data matches, and LED2 if the data does not match.
5.7.3 Selection of EEPROM

Figure 5.7 shows the flow of EEPROM selection.

```
R_device_select()

Set the initial status to "error"

EEPROM number is within valid range?

Yes

Save the EEPROM number

Copy the EEPROM parameter

Set the status to "successful"

return(status)

No

status ← PARA_ERR: Parameter error
status: Local variable for saving the execution result

Checks whether the EEPROM number specified by the parameter corresponds to the number of the EEPROM of 2 Kb to 512 Kb.

g_eeprom_type ← EEPROM number

g_eeprom_type: Variable for saving the EEPROM number

Copies the parameter for the specified EEPROM to the work structure (EEPROM_Info).

EEPROM_Info = EEPROM_ADDRESS[g_eeprom_type]

status ← I2C_OK: Successful

Specifies the value of the local variable status to be returned
```

Figure 5.7 Selection of EEPROM

5.7.4 IIC0 start function

Figure 5.8 shows the flow of the IIC0 start function processing.

```
R_IIC00_Start()

Enable IIC00

IICE0 bit ← 1

return()
```

Figure 5.8 IIC0 start function
5.7.5 Start condition generation function

Figure 5.9 shows the flow of the start condition generation function.

```
R_IIC00_StartCondition()

Generate a start condition

STT0 bit ← 1

Waits for the start condition to be detected (STD0 bit = 1).

Start condition detected?

No

Yes

return()
```

Figure 5.9 Start condition generation function

5.7.6 Stop condition generation function

Figure 5.10 shows the flow of the stop condition generation function.

```
R_IIC00_StopCondition()

Generate a stop condition

SPT0 bit ← 1

Waits for the stop condition to be detected (SPD0 bit = 1).

Stop condition detected?

No

Yes

return()
```

Figure 5.10 Stop condition generation function
5.7.7 EEPROM write processing

Figure 5.11 shows the flow of EEPROM write processing.

```plaintext
R_EEPROM_W()

Check EEPROM address
check_EEPROM.Addr()

Within the EEPROM address range?
Yes

Calculate the cell address

Calculate the slave address
get_slave.Addr()

Specify the access parameter

Generate a start condition
R_IIC00_StartCondition()

Specify the operation information/result flag setting

Transmit the slave address

return

Checks whether the area to be accessed is within the EEPROM and sets or clears the operation information/result flag (g_comstatus) according to the result.

Processing ends if the area to be accessed is not within the EEPROM or the number of data units to be written is 0.

Calculates the EEPROM cell address to be accessed based on the block number specified by the write parameter.

Applies the EEPROM cell address to be accessed to the slave address. (When using EEPROM of 4 Kb to 16 Kb.)

Copies the work parameter to which the slave address has been applied to the access parameter.
g_PARAA ← g_PARAC

Generates a start condition to start using the I²C bus.

Changes the value of the operation information/result flag (g_comstatus) to slave address transmission (TRANSMIT).

Transmits the start condition, and then the slave address.

Figure 5.11 EEPROM write processing
```
5.7.8 EEPROM address check processing

Figure 5.12 shows the flow of EEPROM address check processing.

Sets the initial status to "error".

\[
\text{status} \leftarrow \text{PARAERR}
\]

(status: Local variable)

Sets the EEPROM block to be accessed specified by the g_PARAI structure for which the pointer has been passed by using the parameter, to the work variable.

\[
\text{blocktmp} \leftarrow \text{PARA} \rightarrow \text{block_num}
\]

Processing results in an error and ends if the access start block number (address) is not within the EEPROM address range.

\[
\text{status} \leftarrow \text{I2C_OK: Successful}
\]

Copies the parameter of the structure for which the pointer has been passed by using the parameter to the g_PARAC work structure.

\[
g_{\text{PARAC}} \leftarrow ^{\text{PARA}}
\]

Figure 5.12 EEPROM address check processing
5.7.9 Slave address calculation

Figure 5.13 shows the flow of slave address calculation.

Remark This processing is required when using a 4 Kb EEPROM (R1EX24004A), 8 Kb EEPROM (R1EX24008A), or 16 Kb EEPROM (R1EX24016A). This processing is skipped when using a 2 Kb EEPROM (R1EX24002A) or 32 Kb EEPROM (R1EX24032A) to 512 Kb EEPROM (R1EX24512B).

Figure 5.13 Slave address calculation
5.7.10 Waiting for writing to EEPROM to finish

Figure 5.14 shows the flow of waiting for writing to EEPROM to finish.

Checks the value of the operation information/result flag (\(g\text{\_comstatus}\)) and waits for transmission to finish.

Reads the value of the operation information/result flag (\(g\text{\_comstatus}\)).
\[\text{status} \leftarrow g\text{\_comstatus}\]

Processing ends if writing results in an error.

IICMK0 \(\leftarrow 1\): Mask (disable) interrupts.

IICIF0 \(\leftarrow 0\): Clear the interrupt request flag.

Loops the processing until ACK is returned from the EEPROM.

Repeats start condition generation to slave address transmission to confirm that the write has finished.

Polls the relevant register and waits for the response from the EEPROM.

IICIF0 \(\leftarrow 0\): Clear the interrupt request.

Checks the response from the EEPROM and exits the loop if ACK is returned.

Releases the IIC bus.

IICMK0 \(\leftarrow 0\): Unmask (enable) interrupts.

Specifies a status as the returned value.

---

R01AN1551EJ0100 Rev. 1.00 Page 27 of 38
Mar. 28, 2013

Renesas
5.7.11 EEPROM read processing

Figure 5.15 shows the flow of EEPROM read processing.

R_EEPROM_R()

Check EEPROM address
check_EEPROM_Addr()

Within the EEPROM address range?

Yes

Calculate the cell address

No

Calculates whether the area to be accessed is within the EEPROM and sets or clears the operation information/result flag (g_comstatus) according to the result.

Processing ends if the area to be accessed is not within the EEPROM.

Check EEPROM address range?

Yes

Calculate the slave address
get_slave_Addr()

Within the EEPROM address range?

No

Yes

Generate a start condition
R_IIC00_StartCondition()

Specify the access parameter

Specify the operation information/result flag setting

Transmit the slave address

Copies the work parameter to which the slave address has been applied to the access parameter.

g_PARAA ← g_PARAC

Generates a start condition to start using the I^2C bus.

Changes the value of the operation information/result flag (g_comstatus) to slave address transmission (RECEIVE).

Transmits the start condition, and then the slave address.

Figure 5.15 EEPROM read processing
5.7.12 Waiting for reading from EEPROM to finish

Figure 5.16 shows the flow of waiting for reading from EEPROM to finish.

![Figure 5.16 Wait for reading to finish]

5.7.13 Finishing slave address transmission

Figure 5.17 shows the flow of finishing slave address transmission.

![Figure 5.17 Finishing slave address transmission]
5.7.14 Finishing upper address transmission

Figure 5.18 shows the flow of finishing upper address transmission.

```
R_IIC0_Tx_addr2()

Update the operation information/result flag

Transmit the lower address of EEPROM

return
```

Changes the value of the operation information/result flag (g_comstatus) to the next status (TxLoADDR = 82H).

\[ g_{\text{comstatus}} \leftarrow g_{\text{comstatus}} + 1 \]

Transmits the lower address (2nd byte) of the EEPROM.

---

5.7.15 Restart processing

Figure 5.19 shows the flow of restart processing.

```
R_IIC0_Rx_RST()

Generate a start condition
R_IIC00_StartCondition()

Update the operation information/result flag

Transmit the slave address

return
```

Generates a restart condition to switch the direction of communication.

Changes the value of the operation information/result flag (g_comstatus) to the next status (ReStart = C3H).

\[ g_{\text{comstatus}} \leftarrow \text{ReStart} \]

Transmits the slave address in read mode.
5.7.16 Starting data reception

Figure 5.20 shows the flow of data reception start processing.

```
R_IIC0_RxData_ST()

Update the operation information/result flag

Enable ACK response

Start reception

return
```

- **R_IIC0_RxData_ST()**

  - Initialize the transfer data counter
  - g_data_counter ← 0

- **Update the operation information/result flag**

  - Changes the value of the operation information/result flag (g_comstatus) to the next status (RxData = E0H).
  - g_comstatus ← RxData

- **Enable ACK response**

  - ACKE0 bit ← 1

- **Start reception**

  - IIC0 register ← FFH

Figure 5.20 Starting data reception
5.7.17 Data reception

Figure 5.21 shows the flow of data reception.

Figure 5.21 Data reception

- R_IIC0_RxData() Reads the received data from the IIC0 register and stores it in the data buffer (g_data_bufferR).
- Data counter + 1 Increments the transfer data counter.
  \[ g_{data\_counter} \leftarrow g_{data\_counter} + 1 \]
- Number of data bytes to be read = 1? 
  - No
  - Yes
    - Update the operation information/result flag (g_comstatus) to the next status (RxLast = E1H).
      \[ g_{comstatus} \leftarrow \text{RxLast} \]
    - Disable ACK response
      \[ \text{ACKE0 bit} \leftarrow 0 \]
- Start reception
- IIC0 register \leftarrow \text{FFH}
- return

Disables ACK response if the number of remaining data units to be read is 1.

Changes the value of the operation information/result flag (g_comstatus) to the next status (RxLast = E1H).

[Diagram of data reception flow]
5.7.18 Reception of final data

Figure 5.22 shows the flow of reception of final data.

```
R_IIC0_Rx_LastData()

Store the received data

Generate a stop condition
R_IIC00_StopCondition()

Update the operation information/ result flag

return
```

Reads the received data from the IIC0 register and stores it in the data buffer (g_data_bufferR).

Generates a stop condition to release the bus.

Changes the value of the operation information/result flag (g_comstatus) to the next status (TRNSEND = 00H).

```
g_comstatus ← TRNSEND
```

Figure 5.22 Reception of final data

5.7.19 Starting data transmission

Figure 5.23 shows the flow of data transmission start processing.

```
R_IIC0_TxDataST()

Initialize the transfer data counter

g_data_counter ← 0

Transmit the data to write

Update the operation information/ result flag

return
```

Reads the written data from the data buffer and writes it to the IIC0 register.

Changes the value of the operation information/result flag (g_comstatus) to the next status (TxData = A0H).

```
g_comstatus ← TxData
```

Figure 5.23 Starting data transmission
5.7.20 Data transmission

Figure 5.24 shows the flow of data transmission.

Increment the transfer data counter.
\[ g_{\text{data\_counter}} \leftarrow g_{\text{data\_counter}} + 1 \]

Starts writing to the EEPROM if all the data units have been transmitted \((g_{\text{data\_counter}} = \text{BLOCK\_SIZE})\), or transmits the next data if the number of remaining data units to write is not 0.

Generates a stop condition to instruct the EEPROM to start writing.

Changes the value of the operation information/result flag \((g_{\text{comstatus}})\) to the next status \((\text{TRNSSEND} = 00H)\).
\[ g_{\text{comstatus}} \leftarrow \text{TRNSSEND} \]

Reads the written data from the data buffer and writes it to the IIC0 register.

Figure 5.24 Data transmission
5.7.21 MD_INTIIC0 interrupt servicing

Figures 5.25 to 5.27 show the flow of INTIIC0 interrupt servicing.

![Flowchart](image)

Figure 5.25 MD_INTIIC0 interrupt servicing (1/3)
A

RxHiADDR

Finish upper address transmission
R_IIC0_Tx_addr2()

RxLoADDR

Restart processing
R_IIC0_Rx_RST()

ReStart

Start data reception
R_IIC0_RxData_ST()

B

Jumps to upper address transmission
finish processing during data reception
if g_comstatus is RxHiADDR.

Jumps to data reception restart processing
if g_comstatus is RxLoADDR.

Jumps to data reception start processing
if g_comstatus is ReStart.

D

Figure 5.26 MD_INTIIC0 interrupt servicing (2/3)
Figure 5.27 MD_INTIIC0 interrupt servicing (3/3)

Jumps to final data reception processing if ACK has not been detected after the final data was received.

Jumps to final data reception processing.

Clears bit 6 (transmission/reception bit) of the operation information/result flag ($g_{\text{comstatus}}$).

$g_{\text{comstatus}} \leftarrow g_{\text{comstatus}} \& 10111111B$

Returns $\text{NO}_{\text{ACK1}}$ if the value of the operation information/result flag ($g_{\text{comstatus}}$) indicates the response to slave address transmission.

$g_{\text{comstatus}} \leftarrow \text{NO}_{\text{ACK1}}$

Returns $\text{NO}_{\text{ACK3}}$ if the value of the operation information/result flag ($g_{\text{comstatus}}$) indicates the response to data transmission.

$g_{\text{comstatus}} \leftarrow \text{NO}_{\text{ACK3}}$

Returns $\text{NO}_{\text{ACK2}}$ if the value of the operation information/result flag ($g_{\text{comstatus}}$) indicates another status (response to cell address transmission).

$g_{\text{comstatus}} \leftarrow \text{NO}_{\text{ACK2}}$

Processing results in an error and ends, and a stop condition is generated.
6. Sample Code
Obtain the sample code from the Renesas Electronics website.

7. Reference Documents
V850ES/JF3-L User's Manual: Hardware (R01UH0017E)
V850ES/JG3-L User's Manual: Hardware (R01UH0165E)
V850ES/JG3-L On-chip USB Controller Hardware User’s Manual (R01UH0001E)
(Obtain the latest versions from the Renesas Electronics website.)

Technical updates and technical news
(Obtain the latest versions from the Renesas Electronics website.)

Website and Support
Renesas Electronics Website
   http://www.renesas.com/
Inquiries
   http://www.renesas.com/contact
## Revision Record

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Mar. 28, 2013</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>

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

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

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

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

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

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

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

1. 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.

2. 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.

3. 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.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

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

   "Standard": Computers, office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.) or may cause serious property damage (nuclear reactor control systems, military equipment etc.). 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 for which it is not intended. 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 which the product is not intended by Renesas Electronics.

6. 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.

7. 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 against them. The possibility of physical injury, 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 evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or systems manufactured by you.

8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please 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 malfunctions or damages occurring as a result of your noncompliance with applicable laws and regulations.

9. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacturer’s, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You should not use Renesas Electronics products or 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. When exporting the Renesas Electronics products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

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

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

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

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

Refer to “http://www.renesas.com/” for the latest and detailed information.