Abstract

This document describes accessing a Renesas Electronics R1EX25032ASA00A serial EEPROM using synchronous serial interface mode. The R32C/118 Group MCU has nine channels (UART0 to UART8) that can be used while in synchronous serial interface mode. This application note uses UART2. When using a channel other than UART2, refer to the User’s Manual: Hardware and rewrite the registers associated with the corresponding channel.

Products

R32C/116 Group
R32C/117 Group
R32C/118 Group

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.
## Contents

1. Specifications .............................................................................................................. 3
2. Operation Confirmation Conditions ............................................................................ 4
3. Reference Application Notes ....................................................................................... 4
4. Hardware ...................................................................................................................... 5
   4.1 Pins Used .................................................................................................................. 5
5. Software ...................................................................................................................... 6
   5.1 Operation Overview ..................................................................................................... 8
   5.1.1 Commands for Serial Transfer .................................................................................... 8
   5.1.2 Command Sequence .................................................................................................. 8
   5.1.3 Procedures for Executing EEPROM Initialization, EEPROM Write Processing, and EEPROM Read Processing 8
   5.2 Constants ................................................................................................................ 10
   5.3 Structure/Union List ................................................................................................. 11
   5.4 Variables ............................................................................................................... 12
   5.5 Functions ............................................................................................................ 13
   5.6 Function Specifications ............................................................................................ 14
   5.7 Flowcharts ................................................................................................................. 19
     5.7.1 Main Processing .......................................................................................................... 19
     5.7.2 EEPROM Initialization Sequence Start ................................................................. 20
     5.7.3 EEPROM Write Sequence Start ................................................................................ 20
     5.7.4 EEPROM Read Sequence Start .............................................................................. 21
     5.7.5 Clear Variable of Proceed to Next Command Determination .................................. 21
     5.7.6 Command Execution Start Processing ..................................................................... 22
     5.7.7 Command Execution Complete Processing ............................................................ 23
     5.7.8 Processing to Proceed to the Next Command ........................................................... 24
     5.7.9 Controlling the EEPROM S Pin .................................................................................. 26
     5.7.10 UART2 Serial Transmission (UART2 Transmit Interrupt) ......................................... 26
     5.7.11 UART2 Serial Reception (UART2 Receive Interrupt) ................................................. 27
     5.7.12 5 ms Timeout Processing (Timer A0 Interrupt) .......................................................... 27
     5.7.13 Accepting an EEPROM Write Request (INT0 Interrupt) ......................................... 28
     5.7.14 Accepting an EEPROM Read Request (INT1 Interrupt) ......................................... 28
     5.7.15 Timer A0 Initialization ............................................................................................ 29
     5.7.16 UART2 Initialization ............................................................................................... 30
6. Sample Code .............................................................................................................. 31
7. Reference Documents ................................................................................................. 31
1. Specifications

This document describes how to access an EEPROM using synchronous serial interface mode through UART2. After initializing the EEPROM status register (EEPROM initialization), data is written to the EEPROM (EEPROM write processing) and then data is read from the EEPROM (EEPROM read processing).

This application note uses a Renesas Electronics R1EX25xxx Series EEPROM. For details on the EEPROM, refer to the product datasheet.

Conditions for accessing the EEPROM:
- Bit rate: 1 Mbps
- Transfer data length: 1 to 32 bytes (not including the instruction codes or memory address)

Table 1.1 lists the Peripheral Functions and Their Applications. Figure 1.1 shows the Connection Diagram.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial interface (UART2)</td>
<td>Communication with the EEPROM</td>
</tr>
<tr>
<td>INT0 interrupt</td>
<td>Executes EEPROM write processing</td>
</tr>
<tr>
<td>INT1 interrupt</td>
<td>Executes EEPROM read processing</td>
</tr>
<tr>
<td>Timer A0</td>
<td>5 ms timer for timeout detection (5 ms is the maximum amount of time needed to write to the EEPROM or write to the EEPROM status register)</td>
</tr>
</tbody>
</table>

Figure 1.1 Connection Diagram

Notes:
1. In this application note, EEPROM W pin control is not performed.
2. In this application note, EEPROM HOLD pin control is not performed.
3. The LED is ON when key input can be accepted. The LED is OFF when the EEPROM is being accessed or when there is an EEPROM hardware error.
2. Operation Confirmation Conditions

The sample code accompanying this application note has been run and confirmed under the conditions below.

### Table 2.1 Operation Confirmation Conditions

<table>
<thead>
<tr>
<th>Item</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCU used</td>
<td>R5F64189DFD (R32C/118 Group)</td>
</tr>
<tr>
<td>Device used</td>
<td>R1EX25032ASA00A</td>
</tr>
<tr>
<td>Operating frequencies</td>
<td>• XIN clock: 16 MHz&lt;br&gt; • PLL clock: 100 MHz&lt;br&gt; • Base clock: 50 MHz&lt;br&gt; • CPU clock: 50 MHz&lt;br&gt; • Peripheral bus clock: 25 MHz&lt;br&gt; • Peripheral clock: 25 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>5 V</td>
</tr>
<tr>
<td>Integrated development environment</td>
<td>Renesas Electronics&lt;br&gt; High-performance Embedded Workshop Version 4.08</td>
</tr>
<tr>
<td>C compiler</td>
<td>Renesas Electronics&lt;br&gt; R32C/100 Series C Compiler V.1.02 Release 01</td>
</tr>
</tbody>
</table>
|                       | Compile options<br> -D__STACKSIZE__=0X300<br> -D__ISTACKSIZE__=0X300<br> -DVECTOR_ADR=0xFFFFFBDC<br> -c -finfo -dir "$(CONFIGDIR)"
                       | Default setting is used in the integrated development environment.       |
| Operating mode        | Single-chip mode                                                         |
| Sample code version   | 1.00                                                                     |
| Board used            | Renesas Starter Kit for R32C/118 (product name: R0K64189S000BE)          |

3. Reference Application Notes

Application notes associated with this application note are listed below. Refer to these application notes for additional information.

- R32C/100 Series Configuring PLL Mode (REJ05B1221)
- R32C/100 Series Serial Interface Operation in Special Mode 2 Using Master Transmission/Reception (R01AN0493EJ)
- R32C/100 Series Serial Interface Operation (Transmission in Clock Synchronized Serial Interface Mode) (REJ05B1233)
- R32C/100 Series Serial Interface Operation When Receiving Data in Synchronous Serial Interface Mode (R01AN0178EJ)
4. Hardware

4.1 Pins Used

Table 4.1 lists the Pins Used and Their Functions.

<table>
<thead>
<tr>
<th>Pin Name</th>
<th>I/O</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0_0</td>
<td>Output</td>
<td>Outputs to control the EEPROM S pin (chip-select pin)</td>
</tr>
<tr>
<td>P7_0/TXD2</td>
<td>Output</td>
<td>Outputs serial data to the EEPROM</td>
</tr>
<tr>
<td>P7_1/RXD2</td>
<td>Input</td>
<td>Inputs serial data from the EEPROM</td>
</tr>
<tr>
<td>P7_2/CLK2</td>
<td>Output</td>
<td>Outputs a clock to set the timing for serial data I/O with the EEPROM</td>
</tr>
<tr>
<td>P4_0</td>
<td>Output</td>
<td>Outputs for the LED that indicates the EEPROM is being accessed</td>
</tr>
<tr>
<td>P8_2/INT0</td>
<td>Input</td>
<td>Executes EEPROM write processing</td>
</tr>
<tr>
<td>P8_3/INT1</td>
<td>Input</td>
<td>Executes EEPROM read processing</td>
</tr>
</tbody>
</table>
5. Software

In the sample code for this application note, EEPROM initialization is performed when the MCU is reset, 32 bytes of data are written to the EEPROM (EEPROM write processing) \(^1\) when a falling edge is detected on the INT0 signal, and 32 bytes of data are read from the EEPROM (EEPROM read processing) \(^2\) when a falling edge is detected on the INT1 signal.

Notes:
1. When writing to the EEPROM with user data based on the sample code provided with this application note, designate the EEPROM address and size of the write data so the EEPROM address written does not straddle the page boundaries.
2. When reading the EEPROM with user data based on the sample code provided with this application note, designate the EEPROM address and size of the read data so the EEPROM address read is not larger than the last address.

Table 5.1 lists the instructions for accessing the EEPROM. For details on the instructions, refer to the R1EX25xxx Series datasheet.

Table 5.1 Instructions for Accessing the EEPROM

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>WREN: Write enable</td>
<td>0000 0110</td>
</tr>
<tr>
<td>WRDI: Write disable</td>
<td>0000 0100</td>
</tr>
<tr>
<td>RDSR: Read the EEPROM status register</td>
<td>0000 0101</td>
</tr>
<tr>
<td>WRSR: Write to the EEPROM status register</td>
<td>0000 0001</td>
</tr>
<tr>
<td>READ: Read the data</td>
<td>0000 0011</td>
</tr>
<tr>
<td>WRITE: Write the data</td>
<td>0000 0010</td>
</tr>
</tbody>
</table>

Figure 5.1 shows the format of the EEPROM status register. Use the RDSR instruction to read the register and use the WRSR instruction to write to the register. For details on the instructions, refer to the R1EX25xxx Series datasheet.

![Figure 5.1 EEPROM Status Register Format](image-url)
Table 5.2 lists the UART2 channel settings used when accessing the EEPROM.

**Table 5.2 UART2 Channel Settings Used When Accessing the EEPROM**

<table>
<thead>
<tr>
<th>Mode</th>
<th>Synchronous serial interface mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transmit/receive clock</td>
<td>Internal clock</td>
</tr>
<tr>
<td>U2BRG count source</td>
<td>f1</td>
</tr>
<tr>
<td>TXD2 pin and CLK2 pin output method</td>
<td>Push-pull output</td>
</tr>
<tr>
<td>Bit order</td>
<td>MSB first</td>
</tr>
<tr>
<td>CTS function</td>
<td>Disabled</td>
</tr>
<tr>
<td>CLK polarity</td>
<td>Output transmit data on the falling edge of the transmit/receive clock and input receive data on the rising edge.</td>
</tr>
<tr>
<td>Data logic</td>
<td>Not inverted</td>
</tr>
<tr>
<td>UART2 transmit interrupt</td>
<td>Used (interrupt source: U2TB register empty)</td>
</tr>
<tr>
<td>UART2 receive interrupt</td>
<td>Used</td>
</tr>
<tr>
<td>Bit rate</td>
<td>1 Mbps</td>
</tr>
</tbody>
</table>

The formula for calculating the transfer rate is as follows:

\[
\text{Transfer rate} = \frac{\text{U2BRG count source}}{2 \times (\text{U2BRG register setting value} + 1)} = \frac{25 \text{ MHz (f1)}}{2 \times (11 + 1)} \approx 1 \text{ Mbps}
\]
5.1 Operation Overview

5.1.1 Commands for Serial Transfer
In this application note, EEPROM instructions (WREN, WRSR, RDSR, WRITE, and READ) for port operation and serial transfer are performed in minimum units called “commands”. A command performs the following:

1. Drive port P0_0, which is connected to the S pin, low.
2. Transmit EEPROM instruction.
3. When necessary, transmit and receive EEPROM instruction parameters.
4. Set port P0_0 to high.

5.1.2 Command Sequence
In EEPROM initialization, EEPROM write processing, and EEPROM read processing, commands are executed in a predetermined order. In this application note, the order for executing these commands is called the “command sequence”. The command sequence for each process is shown below.

- EEPROM initialization sequence:
  WREN command → RDSR command → WRSR command → RDSR command
- EEPROM write sequence:
  WREN command → RDSR command → WRITE command → RDSR command
- EEPROM read sequence:
  READ command

5.1.3 Procedures for Executing EEPROM Initialization, EEPROM Write Processing, and EEPROM Read Processing
In this application note, synchronous serial interface mode of the serial interface is used to perform half-duplex communication with the EEPROM.

1. Command sequence start
   An MCU reset calls the EEPROM initialization sequence start, an INT0 interrupt calls the function to start the EEPROM write sequence, and an INT1 interrupt calls the function to start the EEPROM read sequence.
   The following processes are executed by functions:
   • The variable that determines whether or not to proceed to the next command is cleared.
   • The command sequence is set.
   • The command execution status flag (proc_state) is set to command execution started (STATE_BEGIN).
   • The transmit data is stored in the transmit data array (txdata_rw[]) for EEPROM write processing and EEPROM read processing.

2. Command execution start processing
   When the command execution status flag becomes command execution started (STATE_BEGIN), the following processes are executed:
   • The variables used in the transmit interrupt and receive interrupt are set.
   • Low is output from port P0_0.
   • The timer used for timeout detection starts when the RDSR (WIP bit confirmation) command is executed.
   • The command execution status flag is set to command executed (STATE_EXECUTE).
   • UART2 transmit/receive is enabled.
   • EEPROM instructions are transmitted.
(3) Transmitting transmit data
When the data in the U2TB register is transferred to the transmit shift register, a transmit interrupt is generated. In the transmit interrupt handler, 1 byte of transmit data stored in the transmit data array (depending on the command, stored in txdata_rw[], txdata_wren[], txdata_rdsr[], or txdata_wrsr[]) is written to the U2TB register. This process is repeated each time a transmit interrupt is generated until the transmit data has been completely transmitted.

(4) Storing received data
A receive interrupt is generated when the MCU receives data. In the receive interrupt handler, the U2RB register is read and the data is stored in the receive data storage buffer (rxdata_buf[]). After all data has been received, transmission and reception stop, and the command execution status flag is set to command execution complete (STATE_END).

(5) Command execution complete processing
When the command execution status flag becomes command execution complete (STATE_END), the following processes are executed:
• High is output from port P0_0
• When the RDSR command is executed
  The value in the EEPROM status register is stored in the EEPROM status variable (rdsr_val)
• When the READ command is executed
  Data read from the EEPROM is stored in the user designated area.

(6) Processing to proceed to next command
After step (5) is performed, the following processing is executed. Whether or not to proceed to the next command is determined, p_cmd_type is incremented, and the command sequence continues. Also, the command execution status flag is set to command execution started (STATE_BEGIN) in order to start the next command.

Subsequently, steps (2) to (6) are repeated. When all commands are completed, the command execution status flag is set to command not executed (STATE_NON), and the command sequence is completed.

Figure 5.2 shows the Timing Diagram.
### 5.2 Constants

Table 5.3 lists the Constants Used in the Sample Code.

Table 5.3 Constants Used in the Sample Code

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEP_T_S_HOLD</td>
<td>0x000A</td>
<td>S pin set-up and hold wait time</td>
</tr>
<tr>
<td>EEP_REG_INI</td>
<td>0x00</td>
<td>Initial setting value for the EEPROM status register</td>
</tr>
<tr>
<td>EEP_REG_WEL</td>
<td>0x02</td>
<td>WEL bit in the EEPROM status register</td>
</tr>
<tr>
<td>EEP_REG_WIP</td>
<td>0x01</td>
<td>WIP bit in the EEPROM status register</td>
</tr>
<tr>
<td>EEPROM_MEM_ADDR</td>
<td>0x0000</td>
<td>Address on the EEPROM when executing the READ command or WRITE command</td>
</tr>
<tr>
<td>BUFSIZE</td>
<td>32</td>
<td>Size of the body of the transmit/receive data</td>
</tr>
<tr>
<td>TXSIZE (BUFSIZE+3)</td>
<td></td>
<td>Maximum transmit data size (+3 indicates the EEPROM instruction plus the EEPROM address)</td>
</tr>
<tr>
<td>DUMMY</td>
<td>0xff</td>
<td>Dummy data transmitted during data receive processing</td>
</tr>
<tr>
<td>UART_BRG</td>
<td>(12 - 1)</td>
<td>Setting value for the UART baud rate</td>
</tr>
</tbody>
</table>
5.3 Structure/Union List

Figure 5.3 shows the Structure/Union Used in the Sample Code.

```c
/* **** Port definition **** */
#define EEP_P_S p0_0 /**< Port register for EEPROM S pin control */
#define EEP_D_S pd0_0 /**< Port direction register for EEPROM S pin control */
#define EEP_P_LED p4_0 /**< Port register for the LED */
#define EEP_D_LED pd4_0 /**< Port direction register for the LED */

/* **** Command definition **** */
typedef enum
{
    CMD_WREN, /**< WREN */
    CMD_RDSR_WEL, /**< RDSR[Read the WEL bit] */
    CMD_RDSR_WIP, /**< RDSR[Read the WIP bit] */
    CMD_WRSR, /**< WRSR */
    CMD_READ, /**< READ */
    CMD_WRITE, /**< WRITE */
    CMD_TERMINATE, /**< All commands complete */
} eeprom_command_t;

/* **** Command execution status definition **** */
typedef enum
{
    STATE_NON, /**< Command not executed */
    STATE_BEGIN, /**< Command execution started */
    STATE_EXECUTE, /**< Command mid-execution */
    STATE_END, /**< Command execution completed */
    STATE_ERROR, /**< Error occurred during command execution */
} command_process_state_t;

/* **** Interrupt request level definition **** */
typedef enum
{
    INT_LEVEL_DISABLE = 0, /**< Interrupts disabled */
    INT_LEVEL_1 = 1, /**< Level 1 */
    INT_LEVEL_2 = 2, /**< Level 2 */
    INT_LEVEL_3 = 3, /**< Level 3 */
    INT_LEVEL_4 = 4, /**< Level 4 */
    INT_LEVEL_5 = 5, /**< Level 5 */
    INT_LEVEL_6 = 6, /**< Level 6 */
    INT_LEVEL_7 = 7, /**< Level 7 */
} interrupt_level_t;

/* **** Port status definition **** */
typedef enum
{
    LEVEL_LOW, /**< Low level */
    LEVEL_HIGH /**< High level */
} logic_level_t;
```

Figure 5.3 Structure/Union Used in the Sample Code
## 5.4 Variables

Table 5.4 lists the Global Variables and Table 5.5 lists the const Variables.

### Table 5.4 Global Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8_t</td>
<td>rdsr_val</td>
<td>EEPROM status variable</td>
<td>variable_clear, cmd_proc_end, cmd_proc_change</td>
</tr>
<tr>
<td>uint8_t</td>
<td>txdata_rw[]</td>
<td>Transmit data array when executing a READ or WRITE command</td>
<td>R_EEP_StartWriteSeq, R_EEP_StartReadSeq (1)</td>
</tr>
<tr>
<td>uint8_t</td>
<td>rxdata_buf[]</td>
<td>Receive data storage buffer</td>
<td>cmd_proc_begin, cmd_proc_end</td>
</tr>
<tr>
<td>uint8_t</td>
<td>comm_data_size_array[]</td>
<td>Transmit data size for each command</td>
<td>R_EEP_StartWriteSeq, R_EEP_StartReadSeq, cmd_proc_begin, cmd_proc_end</td>
</tr>
<tr>
<td>uint8_t const *</td>
<td>p_cmd_type</td>
<td>Command pointer</td>
<td>R_EEP_StartInitSeq, R_EEP_StartWriteSeq, R_EEP_StartReadSeq, cmd_proc_begin, cmd_proc_end</td>
</tr>
<tr>
<td>uint8_t</td>
<td>proc_state</td>
<td>Command execution status flag</td>
<td>main, R_EEP_StartInitSeq, R_EEP_StartWriteSeq, R_EEP_StartReadSeq, cmd_proc_begin, cmd_proc_change</td>
</tr>
<tr>
<td>bool</td>
<td>f_timeout</td>
<td>Timeout flag</td>
<td>variable_clear, cmd_proc_begin, cmd_proc_change, timer_a0_interrupt, int0_interrupt, int1_interrupt</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>p_txdata</td>
<td>Transmit data pointer</td>
<td>cmd_proc_begin, uart2_tx_interrupt</td>
</tr>
<tr>
<td>uint8_t</td>
<td>remaining_tx_size</td>
<td>Untransmitted data size</td>
<td>cmd_proc_begin, uart2_tx_interrupt</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>p_rxdata_buf</td>
<td>Pointer for the receive data storage buffer</td>
<td>cmd_proc_begin, uart2_rx_interrupt</td>
</tr>
<tr>
<td>uint8_t</td>
<td>remaining_rx_size</td>
<td>Unreceived data size</td>
<td>cmd_proc_begin, uart2_rx_interrupt</td>
</tr>
<tr>
<td>uint8_t *</td>
<td>p_eep_read_data</td>
<td>Pointer for the area where the data read from the EEPROM is stored</td>
<td>R_EEP_StartReadSeq, cmd_proc_end</td>
</tr>
<tr>
<td>uint8_t</td>
<td>write_buffer[BUFSIZE]</td>
<td>Area for EEPROM write data</td>
<td>int0_interrupt, main</td>
</tr>
<tr>
<td>uint8_t</td>
<td>read_buffer[BUFSIZE]</td>
<td>Area for storing data read from the EEPROM</td>
<td>int1_interrupt</td>
</tr>
</tbody>
</table>

Note:
1. This function is used as an element in the composition of `p_tx_data_array[]`. 

(1) `R_EEP_StartReadSeq` is defined in Table 5.5.
### Table 5.5  const Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>const uint8_t</td>
<td>cmd_seq_init[]</td>
<td>Command sequence storage array for EEPROM initialization</td>
<td>R_EEP_StartInitSeq</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>cmd_seq_write[]</td>
<td>Command sequence storage array for EEPROM write processing</td>
<td>R_EEP_StartWriteSeq</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>cmd_seq_read[]</td>
<td>Command sequence storage array for EEPROM read processing</td>
<td>R_EEP_StartReadSeq</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>txdata_wren[]</td>
<td>Transmit data array when using WREN</td>
<td>See Note 1.</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>txdata_rdsr[]</td>
<td>Transmit data array when using RDSR</td>
<td>See Note 1.</td>
</tr>
<tr>
<td>const uint8_t</td>
<td>txdata_wrsr[]</td>
<td>Transmit data array when using WRSR</td>
<td>See Note 1.</td>
</tr>
<tr>
<td>const uint8_t * const</td>
<td>p_tx_data_array[]</td>
<td>Pointer for the transmit data arrays corresponding to each command</td>
<td>cmd_proc_begin</td>
</tr>
</tbody>
</table>

Note:
1. This function is used as an element in the composition of p_tx_data_array[].

### 5.5 Functions

Table 5.6 lists the Functions.

### Table 5.6  Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
</tr>
<tr>
<td>R_EEP_StartInitSeq</td>
<td>EEPROM initialization sequence start</td>
</tr>
<tr>
<td>R_EEP_StartWriteSeq</td>
<td>EEPROM write sequence start</td>
</tr>
<tr>
<td>R_EEP_StartReadSeq</td>
<td>EEPROM read sequence start</td>
</tr>
<tr>
<td>variable_clear</td>
<td>Clear variable of proceed to next command determination</td>
</tr>
<tr>
<td>cmd_proc_begin</td>
<td>Command execution start processing</td>
</tr>
<tr>
<td>cmd_proc_end</td>
<td>Command execution complete processing</td>
</tr>
<tr>
<td>cmd_proc_change</td>
<td>Processing to proceed to next command</td>
</tr>
<tr>
<td>eep_s_control</td>
<td>Controlling the EEPROM S pin</td>
</tr>
<tr>
<td>uart2_tx_interrupt</td>
<td>UART2 serial transmission (UART2 transmit interrupt)</td>
</tr>
<tr>
<td>uart2_rx_interrupt</td>
<td>UART2 serial reception (UART2 receive interrupt)</td>
</tr>
<tr>
<td>timer_a0_interrupt</td>
<td>5 ms timeout processing (timer A0 interrupt)</td>
</tr>
<tr>
<td>int0_interrupt</td>
<td>Accepting an EEPROM write request (INT0 interrupt)</td>
</tr>
<tr>
<td>int1_interrupt</td>
<td>Accepting an EEPROM read request (INT1 interrupt)</td>
</tr>
<tr>
<td>timer_a0_init</td>
<td>Timer A0 initialization</td>
</tr>
<tr>
<td>uart2_init</td>
<td>UART2 initialization</td>
</tr>
</tbody>
</table>
## 5.6 Function Specifications

The following tables list the sample code function specifications.

<table>
<thead>
<tr>
<th>Function</th>
<th>Outline</th>
<th>Header</th>
<th>Declaration</th>
<th>Description</th>
<th>Argument</th>
<th>Returned value</th>
</tr>
</thead>
<tbody>
<tr>
<td>main</td>
<td>Main processing</td>
<td>None</td>
<td>void main(void)</td>
<td>This function performs the initial settings for the system clock, UART2, and timer A0, and initializes the variables. Then, EEPROM initialization sequence is started and the MCU enters the main loop. In the main loop, the command execution start processing, command execution complete processing, and processing to proceed to the next command are performed according to the command execution status flag.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>R_EEP_StartInitSeq</td>
<td>EEPROM initialization sequence start</td>
<td>None</td>
<td>void R_EEP_StartInitSeq(void)</td>
<td>The command sequence is performed in the EEPROM initialization sequence, and the command execution status flag is set to command execution started (STATE_BEGIN).</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>
| R_EEP_StartWriteSeq | EEPROM write sequence start | None    | void R_EEP_StartWriteSeq(uint8_t *p_write_data, uint8_t size, uint16_t eeprom_adrs) | This function sets the WRITE command transmit data size and generates serial data. Then, the command sequence is set in the EEPROM write processing, and the command execution status flag is set to command execution start (STATE_BEGIN). | uint8_t *p_write_data: Pointer for the area where the write data is stored  
uint8_t size: Size of the write data (in bytes)  
uint16_t eeprom_adrs: EEPROM address         | None           |
### R_EEP_StartReadSeq

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>EEPROM read sequence start</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Declaration</strong></td>
<td>void R_EEP_StartReadSeq(uint8_t *p_read_data, uint8_t size, uint16_t eeprom_adrs)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This function sets the READ command transmit data size and generates serial data. It also sets the pointer (p_eep_read_data) for the area where the data read from the EEPROM is stored. Then, the command sequence is set in the EEPROM read processing, and the command execution status flag is set to command execution start (STATE_BEGIN).</td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>uint8_t *p_read_data: Pointer for the area where the read data is stored</td>
</tr>
<tr>
<td></td>
<td>uint8_t size: Size of the read data (in bytes)</td>
</tr>
<tr>
<td></td>
<td>uint16_t eeprom_adrs: EEPROM address</td>
</tr>
<tr>
<td><strong>Returned value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### variable_clear

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Clear variable of proceed to next command determination</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void variable_clear(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This function clears the variable used in determining command transition.</td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Returned value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### cmd_proc_begin

<table>
<thead>
<tr>
<th><strong>Outline</strong></th>
<th>Command execution start processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void cmd_proc_begin(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This function sets the variable for the serial transmit/receive processing, and after the S pin, which is connected to the P0_0 port, is driven low. Also, when executing the RDSR command (read the WIP bit), the timer for the timeout detection starts. Then, the command execution status flag is set to command execution in progress (STATE_EXECUTE), UART2 is transmit/receive enabled, and the first byte of data is transmitted.</td>
</tr>
<tr>
<td><strong>Argument</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Returned value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
### cmd_proc_end

<table>
<thead>
<tr>
<th>Outline</th>
<th>Command execution complete processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void cmd_proc_end(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function drives S pin, which is connected to the P0_0 port, high. When executing the RDSR command, the value received from the EEPROM status register is set to the EEPROM status variable. When executing the READ command, the EEPROM data received is stored in the user designated area.</td>
</tr>
<tr>
<td>Argument</td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
</tbody>
</table>

### cmd_proc_change

<table>
<thead>
<tr>
<th>Outline</th>
<th>Processing to proceed to next command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void cmd_proc_change(void)</td>
</tr>
</tbody>
</table>
| Description   | This function performs proceed to next command determination and proceed to next command.  
• Proceed to next command determination:  
This function reads the EEPROM status register and confirms that a timeout has occurred. This function also determines whether or not to proceed to the next command, and determines if an error occurred during command execution.  
• Proceed to next command:  
No error occurred during command execution and the command sequence proceeds to the next command  
Increment p_cmd_type and continue the command sequence. When the command becomes all commands complete (CMD_TERMINATE), the command execution status flag becomes command not executed (STATE_NON). In all other cases, the command execution status flag becomes command execution started (STATE_BEGIN).  
No error occurred during command execution and the command sequence does not proceed to the next command  
The command execution status flag is set to command execution started (STATE_BEGIN).  
An error occurred during command execution  
The command execution status flag is set to error occurred during command execution (STATE_ERROR). Also, in the sample code accompanying this application note, there is no processing to transition out of the STATE_ERROR state. |
| Argument      | None                                   |
| Returned value| None                                   |
### eep_s_control

**Outline**
Controlling the EEPROM S pin

**Header**
None

**Declaration**
void eep_s_control(logic_level_t level)

**Description**
High or low is output from port P0_0 to the EEPROM S pin.

**Argument**
logic_level_t level: Output level is set
- LEVEL_LOW: Low level
- LEVEL_HIGH: High level

**Returned value**
None

### uart2_tx_interrupt

**Outline**
UART2 serial transmission (UART2 transmit interrupt)

**Header**
None

**Declaration**
void uart2_tx_interrupt(void)

**Description**
This function is called by the UART2 transmit interrupt that occurs when the U2TB register becomes empty. If there is untransmitted data, 1 byte of the untransmitted data is transmitted.

**Argument**
None

**Returned value**
None

### uart2_rx_interrupt

**Outline**
UART2 serial reception (UART2 receive interrupt)

**Header**
None

**Declaration**
void uart2_rx_interrupt(void)

**Description**
This function is called by the UART2 receive interrupt that occurs when reception is completed. After reading the UART2 receive buffer (U2RB), this function stores the data in the receive data storage buffer (rxdata_buf[]). When all data is received, UART2 transmission/reception is disabled, and the command execution status flag is set to command execution completed (STATE_END).

**Argument**
None

**Returned value**
None

### timer_a0_interrupt

**Outline**
5 ms timeout processing (timer A0 interrupt)

**Header**
None

**Declaration**
void timer_a0_interrupt(void)

**Description**
After starting timer A0 and 5 ms elapse, this function is called by the timer A0 interrupt that occurs. This function sets the timeout flag (f_timeout) and stops the timer for timeout detection.

**Argument**
None

**Returned value**
None
### int0_interrupt

<table>
<thead>
<tr>
<th>Outline</th>
<th>Accepting an EEPROM write request (INT0 interrupt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void int0_interrupt(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called by the INT0 interrupt that occurs when a low is input to port P8_2. This function executes the EEPROM write sequence start (R_EEP_StartWriteSeq).</td>
</tr>
<tr>
<td>Argument</td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
</tbody>
</table>

### int1_interrupt

<table>
<thead>
<tr>
<th>Outline</th>
<th>Accepting an EEPROM read request (INT1 interrupt)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void int1_interrupt(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function is called by the INT1 interrupt that occurs when a low is input to port P8_3. This function executes the EEPROM read sequence start (R_EEP_StartReadSeq).</td>
</tr>
<tr>
<td>Argument</td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
</tbody>
</table>

### timer_a0_init

<table>
<thead>
<tr>
<th>Outline</th>
<th>Timer A0 initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void timer_a0_init(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function sets timer A0 to timer mode.</td>
</tr>
<tr>
<td>Argument</td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
</tbody>
</table>

### uart2_init

<table>
<thead>
<tr>
<th>Outline</th>
<th>UART2 initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void uart2_init(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function sets the UART2 channel to synchronous serial interface mode.</td>
</tr>
<tr>
<td>Argument</td>
<td>None</td>
</tr>
<tr>
<td>Returned value</td>
<td>None</td>
</tr>
</tbody>
</table>
5.7 Flowcharts

5.7.1 Main Processing

Figure 5.4 shows the Main Processing.

- **main**
  - Disable maskable interrupts
    - I flag ← 0
  - PLL clock setting
    - SetPLLclock()
    - Clock frequencies are set while in PLL mode
  - UART2 initialization
    -_uart2_init()
    - UART2 is set to synchronous serial interface mode
  - Timer A0 initialization
    - timer_a0_init()
    - Set timer A0 to timer mode
  - Enable maskable interrupts
    - I flag ← 1
  - Initialize the command execution status flag
    - proc_state ← STATE_NON
  - Set write data
    - write_buffer[] ← 0 to 31
  - Turn OFF the LED
    - EEP_P_LED ← 1
  - EEPROM initialization sequence start
    - R_EEP_StartInitSeq()

- Command execution status flag
  - STATE_BEGIN
  - default
    - Command execution complete processing
      - cmd_proc_end()
    - Processing to proceed to next command
      - cmd_proc_change()
  - Command execution start processing
    - cmd_proc_begin()

- Is the command execution status flag set to "command not executed (STATE_NON)?"
  - No
  - Enable interrupts INT0 and INT1
  - Turn ON the LED
    - EEP_P_LED ← 0
  - Yes

Figure 5.4 Main Processing
5.7.2 EEPROM Initialization Sequence Start

Figure 5.5 shows the start of the EEPROM initialization sequence.

![Diagram of EEPROM Initialization Sequence Start]

5.7.3 EEPROM Write Sequence Start

Figure 5.6 shows the start of the EEPROM write sequence.

![Diagram of EEPROM Write Sequence Start]
### 5.7.4 EEPROM Read Sequence Start

Figure 5.7 shows the start of the EEPROM read sequence.

![Diagram](https://via.placeholder.com/150)

**Arguments**
- `uint8_t *p_read_data`: Pointer for the area where read data is stored
- `uint8_t size`: Size of read data (in bytes)
- `uint16_t eeprom_adrs`: EEPROM address

**Is the command execution status flag set to "command not executed (STATE_NON)"?**
- Yes
- No

**Is the read data size 0?**
- Yes
- No

**Clear variable of proceed to next command determination**
- `variable_clear`

**Set the size of the data to be transmitted using the READ command**
- `comm_data_size_array[CMD_READ] ← size (max. 32 bytes) + 3 (+3 is the amount of the EEPROM instruction plus the EEPROM address)`

**Generate serial data for transmission, and store that data in the transmit data array until a READ or WRITE command is executed**
- `txdata_rw[0] ← 03h: EEPROM instruction`
- `txdata_rw[1] ← (eeprom_adrs >> 8): EEPROM address`
- `txdata_rw[2] ← (eeprom_adrs)`
- `txdata_rw[i+3] ← DUMMY: Dummy data`

**Set the pointer for the area where the data read from the EEPROM is stored**
- `p_eep_read_data ← p_read_data`

**Set the command sequence with the EEPROM read processing**
- `p_cmd_type ← cmd_seq_read`

**Set the command execution status flag to command execution started**
- `proc_state ← STATE_BEGIN`

**Return**

![Figure 5.7 EEPROM Read Sequence Start](https://via.placeholder.com/150)

### 5.7.5 Clear Variable of Proceed to Next Command Determination

Figure 5.8 shows how to clear the variable of the proceed to the next command determination.

**Clear variable of proceed to next command determination**
- `variable_clear`

**rdsr_val ← 0: EEPROM status variable**
- `f_timeout ← false: Timeout flag`

**Return**

![Figure 5.8 Clear Variable of Proceed to Next Command Determination](https://via.placeholder.com/150)
5.7.6 Command Execution Start Processing

Figure 5.9 shows the command execution start processing.

![Diagram of command execution start processing]

1. If the first byte of data is written directly to the UART transmit buffer, a transmit interrupt is generated before the pointer indicated by the transmit buffer is updated, causing the pointer to become misaligned. To prevent this from happening, the first byte of transmit data should be saved to the automatic variable first.
5.7.7 Command Execution Complete Processing

Figure 5.10 shows the command execution complete processing.

```
Figure 5.10 Command Execution Complete Processing

```
5.7.8 Processing to Proceed to the Next Command

Figure 5.11 and Figure 5.12 show the processing to proceed to the next command.

![Flowchart for Processing to Proceed to the Next Command]

Figure 5.11 Processing to Proceed to the Next Command (1/2)
Figure 5.12  Processing to Proceed to the Next Command (2/2)
5.7.9 Controlling the EEPROM S Pin

Figure 5.13 shows controlling the EEPROM S pin.

```
deep_s_control

Wait 90 ns

Is the output level set to high?

No

Yes

Set the output of the EEPROM S pin control port (P0_0) to high

Set the output of the EEPROM S pin control port (P0_0) to low

Wait 90 ns

return
```

Figure 5.13 Controlling the EEPROM S Pin

5.7.10 UART2 Serial Transmission (UART2 Transmit Interrupt)

Figure 5.14 shows UART2 serial transmission (UART2 transmit interrupt).

```
uart2_tx_interrupt

Has all data been transmitted?

Yes

No

Decrement the untransmitted data size

Transmit 1 byte of data U2TB ← *p_txdata

Increment the transmit data pointer

return
```

Figure 5.14 UART2 Serial Transmission (UART2 Transmit Interrupt)
5.7.11 UART2 Serial Reception (UART2 Receive Interrupt)

Figure 5.15 shows UART2 serial reception (UART2 receive interrupt).

![ UART2 Serial Reception Diagram ]

5.7.12 5 ms Timeout Processing (Timer A0 Interrupt)

Figure 5.16 shows 5 ms timeout processing (timer A0 interrupt).

![ 5 ms Timeout Processing Diagram ]
5.7.13 Accepting an EEPROM Write Request (INT0 Interrupt)

Figure 5.17 shows accepting an EEPROM write request (INT0 interrupt).

![Figure 5.17 Accepting an EEPROM Write Request (INT0 Interrupt)](image)

5.7.14 Accepting an EEPROM Read Request (INT1 Interrupt)

Figure 5.18 shows accepting an EEPROM read request (INT1 interrupt).

![Figure 5.18 Accepting an EEPROM Read Request (INT1 Interrupt)](image)
### 5.7.15 Timer A0 Initialization

Figure 5.19 shows timer A0 initialization.

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>timer_a0_init</td>
</tr>
<tr>
<td>2.</td>
<td>Disable the timer A0 interrupt</td>
</tr>
<tr>
<td>3.</td>
<td>TA0IC register ← 00h</td>
</tr>
<tr>
<td>4.</td>
<td>Bits ILVL2 to ILVL0 = 000b: Level 0 (interrupt disabled)</td>
</tr>
<tr>
<td>5.</td>
<td>Stop the timer A0 count</td>
</tr>
<tr>
<td>6.</td>
<td>TABSR register</td>
</tr>
<tr>
<td>7.</td>
<td>TA0S bit ← 0</td>
</tr>
<tr>
<td>8.</td>
<td>Set the timer A0 mode register</td>
</tr>
<tr>
<td>9.</td>
<td>TA0MR register ← 40h</td>
</tr>
<tr>
<td>10.</td>
<td>Bits TMOD1 and TMOD0 = 00b: Timer mode</td>
</tr>
<tr>
<td>11.</td>
<td>Bits MR2 and MR1 = 00b: No gate function</td>
</tr>
<tr>
<td>12.</td>
<td>Bits TCK1 and TCK0 = 01b: /8</td>
</tr>
<tr>
<td>13.</td>
<td>TA0IC register ← 02h</td>
</tr>
<tr>
<td>14.</td>
<td>Bits ILVL2 to ILVL0 = 010b: Level 2</td>
</tr>
<tr>
<td>15.</td>
<td>Set the timer A0 interrupt control register</td>
</tr>
<tr>
<td>16.</td>
<td>return</td>
</tr>
</tbody>
</table>

**Figure 5.19 Timer A0 Initialization**
5.7.16 UART2 Initialization

Figure 5.20 shows UART2 initialization.

```plaintext
uart2_init

S2TIC register ← 00h
Bits ILVL2 to ILVL0 = 000b: Level 0 (interrupt disabled)
S2RIC register ← 00h
Bits ILVL2 to ILVL0 = 000b: Level 0 (interrupt disabled)

U2C1 register ← 00h
TE bit = 0: Transmission disabled
RE bit = 0: Reception disabled
U2MR register ← 01h
Bits SMD2 to SMD0 = 001b: Synchronous serial interface mode
CKDIR bit = 0: Internal clock

U2C0 register ← 09h
Bits CLK1 and CLK0 = 00b: f1
CRD bit = 1: CTS disabled
CKPOL bit = 0: Output transmit data on the falling edge of the transmit/receive
clock and input receive data on the rising edge
UFORM bit = 1: MSB first

U2BRG register ← UART_BRG: Approximately 1 Mbps
U2C1 register ← 00h
U2RRM bit = 0: Continuous receive mode disabled
U2LCH bit = 0: Data is not logic inverted

U2SMR register ← 00h
U2SMR2 register ← 00h
U2SMR3 register ← 00h
U2SMR4 register ← 00h

S2TIC register ← 03h
Bits ILVL2 to ILVL0 = 011b: Level 3
S2RIC register ← 03h
Bits ILVL2 to ILVL0 = 011b: Level 3

P7_0S register ← 03h
Bits PSEL2 to PSEL0 = 011b: UART2 output (TXD2)
P7_1S register ← 00h
Bits PSEL2 to PSEL0 = 000b: I/O port P7_1
P7_2S register ← 03h
Bits PSEL2 to PSEL0 = 011b: UART2 output (CLK2 output)
P7D register ← 05h
PD7_0 bit = 1: Output port
PD7_1 bit = 0: Input port
PD7_2 bit = 1: Output port

P0 register
P0_0 bit ← 1
P0D register
PD0_0 bit ← 1

P4 register
P4_0 bit ← 0
PD4 register
PD4_0 bit ← 1
```

Figure 5.20 UART2 Initialization
6. **Sample Code**
Sample code can be downloaded from the Renesas Electronics website.

7. **Reference Documents**
R32C/117 Group User’s Manual: Hardware Rev.1.10
R32C/118 Group User's Manual: Hardware Rev.1.10
The latest versions can be downloaded from the Renesas Electronics website.

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

C Compiler Manual
R32C/100 Series C Compiler Package V.1.02
C Compiler User’s Manual Rev.2.00
The latest version can be downloaded from the Renesas Electronics website.

R1EX25xxx Series EEPROM Datasheet Rev. 0.01
The latest version can be downloaded from the Renesas Electronics website.

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

Inquiries
http://www.renesas.com/contact/
# Revision History

## R32C/100 Series
### Accessing an EEPROM Using Synchronous Serial Interface Mode

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Dec. 14, 2012</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.

<table>
<thead>
<tr>
<th>1. Handling of Unused Pins</th>
<th>Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>2. Processing at Power-on</td>
<td>The state of the product is undefined at the moment when power is supplied.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>3. Prohibition of Access to Reserved Addresses</td>
<td>Access to reserved addresses is prohibited.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>4. Clock Signals</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>5. Differences between Products</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>
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-crim systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized to be used in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implementations 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 the possibility of physical injury, injury or damage caused by fire or 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 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 damages or losses 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 manufacture, 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. 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.