RX62N, RX621 Group
Example of Memory-Protection Unit Settings

Introduction

This application note describes, as example settings for the memory-protection unit (MPU) of the RX62N Group and RX621 Group, the method for specifying memory-protection areas, the area search function, the conditions under which access exceptions are generated, and exception handling when an access exception occurs.

Target Device
RX62N Group, RX621 Group

Contents

1. Specifications .................................................................................................................................... 2
2. Operation Confirmation Conditions ................................................................................................... 3
3. Peripheral Functions ......................................................................................................................... 4
4. Hardware ........................................................................................................................................... 7
5. Software ............................................................................................................................................ 8
6. Sample Code ................................................................................................................................... 51
7. Reference Documents ..................................................................................................................... 51
1. Specifications

The example program presented in this application note reads, writes, and executes code in a “read-only area,” a “write-only area,” and an “execute-only area,” respectively, set in the MPU.

A memory-protection error occurs when an unauthorized access to one of these areas takes place.

It is also possible to use the area search function to detect the access enabled/disabled status of an area.

The example program presented in this application note uses three switches and four LEDs.

- Pressing switch 1 (SW1) selects the area to be accessed from among the available areas.
- Pressing switch 2 (SW2) toggles the area search function on and off (enabled/disabled).
- Pressing switch 3 (SW3) performs an access, using the access method specified by SW1 and SW2.

The LEDs indicate the operating status of the program.

- LED0 and LED1 indicate the area being accessed.
- LED2 indicates whether the area search function is enabled or disabled.
- LED3 indicates the execution result.

Note that the example program presented in this application note uses conditional compilation to switch between read, write, and execute access. For details, see section 5, Software.

Table 1.1 lists the peripheral functions used by the sample program. Figure 1.1 shows an operation overview.

<table>
<thead>
<tr>
<th>Peripheral Function</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPU</td>
<td>Memory protection</td>
</tr>
<tr>
<td>Compare match timer channel 0 (CMT0)</td>
<td>System timer (switch chattering elimination, LED blink timer)</td>
</tr>
</tbody>
</table>

![Figure 1.1 Operation Overview](image)
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>RX62N Group R5F562N8BDBG</td>
</tr>
<tr>
<td>Operating frequency</td>
<td>• Main clock: 12.0 MHz</td>
</tr>
<tr>
<td></td>
<td>• ICLK: 96 MHz</td>
</tr>
<tr>
<td></td>
<td>• PCLK: 48 MHz</td>
</tr>
<tr>
<td></td>
<td>• BCLK: 24 MHz</td>
</tr>
<tr>
<td>Operating voltage</td>
<td>5.0 V (CPU operating voltage: 3.3 V)</td>
</tr>
<tr>
<td>Integrated development</td>
<td>High-performance Embedded Workshop Version 4.09.00.007</td>
</tr>
<tr>
<td>environment</td>
<td></td>
</tr>
<tr>
<td>Toolchain</td>
<td>RX Standard Toolchain (V1.2.0.0)</td>
</tr>
<tr>
<td></td>
<td>RX Family C/C++ Compiler Driver V.1.02.00.000</td>
</tr>
<tr>
<td></td>
<td>RX Family C/C++ Compiler V.1.02.00.000</td>
</tr>
<tr>
<td></td>
<td>RX Family Assembler V.1.02.00.000</td>
</tr>
<tr>
<td></td>
<td>Optimizing Linkage Editor V.10.02.00.000</td>
</tr>
<tr>
<td></td>
<td>RX Family C/C++ Standard Library Generator V1.02.00.000</td>
</tr>
<tr>
<td>iodefine.h version</td>
<td>V 2.0</td>
</tr>
<tr>
<td>Endian mode</td>
<td>Little-endian</td>
</tr>
<tr>
<td>Operating mode</td>
<td>Single-chip mode</td>
</tr>
<tr>
<td>Sample code version</td>
<td>Version 1.00</td>
</tr>
<tr>
<td>Board used</td>
<td>Renesas Starter Kit+ for RX62N (R0K5562N0S000BE #WS)</td>
</tr>
</tbody>
</table>
3. Peripheral Functions

Supplementary information about the MPU is presented below. A basic discussion is provided in RX62N Group, RX621 Group—User’s Manual: Hardware.

The RX62N Group and RX621 Group are equipped with an on-chip MPU that performs address checking of CPU accesses within the entire address space (0000 0000h to FFFF FFFFh). Area-specific access-control information can be specified for up to eight areas and the background area. The supported access-control information for the individual areas consists of permission to read, permission to write, and permission to execute.

The access-control information is enabled when the processor mode of the CPU is user mode. Memory protection is not applied when the CPU is in supervisor mode. When the access-control information is enabled and an attempt by the CPU to access an area causes an access-control information violation (a memory-protection error) to be detected, access exception handling starts. Note that it is possible to limit accesses to permitted areas only, and prevent the CPU from generating access-control information violations, by using the area search function to determine the access-control information of areas to be accessed.

Notes:
1. The MPU only monitors memory accesses by the CPU. It has no effect on memory accesses by the DMAC or DTC.
2. Interrupt handling is performed in supervisor mode, so the MPU access-control information has no effect.
3. To enable the MPU access-control information, it is necessary to put the CPU into user mode.
4. The background area refers to the entire address space, except for the areas set as memory-protection areas 0 to 7.
5. Registers related to the memory-protection unit can be accessed only in supervisor mode.

For details on transitioning from user mode to supervisor mode, see 5.7.17, Change from User Mode to Supervisor Mode.

For details on transitioning from supervisor mode to user mode, see 5.7.16, Change from Supervisor Mode to User Mode.
3.1 Memory Protection by the MPU

The example program presented in this application note uses the MPU to specify the memory protection applied to different areas of memory. Figure 3.1 shows the section allocations and memory-protection area settings, and Table 3.1 lists the access-control information for each memory area.

Note that in order to implement execute access to monitor area A and monitor area B in the on-chip RAM area, the program PSAMPLE1 is transferred to PRead_Area and the program PSAMPLE2 is transferred to PWrite_Area.

```
<table>
<thead>
<tr>
<th>Section Name</th>
<th>Section Allocation</th>
<th>Memory-Protection Area Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>SU, SI</td>
<td>Stack area</td>
<td></td>
</tr>
<tr>
<td>PUser_Area</td>
<td>User program area</td>
<td></td>
</tr>
<tr>
<td>BUser_Area</td>
<td>User RAM area</td>
<td></td>
</tr>
<tr>
<td>BUser_Area_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BRead_Area</td>
<td>Monitor area A</td>
<td>Memory-protection area 3</td>
</tr>
<tr>
<td>BWrite_Area</td>
<td>Write-only area</td>
<td></td>
</tr>
<tr>
<td>PRead_Area</td>
<td>Monitor area A</td>
<td>Memory-protection area 4</td>
</tr>
<tr>
<td>PWrite_Area</td>
<td>Write-only area</td>
<td>Memory-protection area 5</td>
</tr>
<tr>
<td>User program area</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor area A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor area B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor area C</td>
<td>Execute-only area</td>
<td></td>
</tr>
<tr>
<td>FIXEDVECT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Portions without indications are background area.
```

Figure 3.1 Section Allocations and Memory-Protection Area Settings
### Table 3.1 Access-Control Information by Memory Area

<table>
<thead>
<tr>
<th>Area Name and Description</th>
<th>Corresponding Register</th>
<th>Setting Address</th>
<th>Access Control Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack area specified in sample code</td>
<td>RSPAGE0 (area 0 start)</td>
<td>SU start address</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>REPAGE0 (area 0 end)</td>
<td>SI end address</td>
<td>○</td>
</tr>
<tr>
<td>I/O register area specified in sample code</td>
<td>RSPAGE1 (area 1 start)</td>
<td>0008 C00h</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>REPAGE1 (area 1 end)</td>
<td>0008 C100h</td>
<td>○</td>
</tr>
<tr>
<td>Program area specified in sample code</td>
<td>RSPAGE2 (area 2 start)</td>
<td>PUser_Area start address</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>REPAGE2 (area 2 end)</td>
<td>PUser_Area end address</td>
<td>×</td>
</tr>
<tr>
<td>RAM use area specified in sample code</td>
<td>RSPAGE3 (area 3 start)</td>
<td>BUser_Area start address</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>REPAGE3 (area 3 end)</td>
<td>BUser_Area_1 end address</td>
<td>○</td>
</tr>
<tr>
<td>Monitor area A Read-only area</td>
<td>RSPAGE4 (area 4 start)</td>
<td>PRead_Area start address</td>
<td>○</td>
</tr>
<tr>
<td></td>
<td>REPAGE4 (area 4 end)</td>
<td>BRead_Area end address</td>
<td>○</td>
</tr>
<tr>
<td>Monitor area B Write-only area</td>
<td>RSPAGE5 (area 5 start)</td>
<td>PWrite_Area start address</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>REPAGE5 (area 5 end)</td>
<td>BWrite_Area end address</td>
<td>×</td>
</tr>
<tr>
<td>Monitor area C Execute-only area</td>
<td>RSPAGE6 (area 6 start)</td>
<td>PExecute_Area start address</td>
<td>×</td>
</tr>
<tr>
<td></td>
<td>REPAGE6 (area 6 end)</td>
<td>CExecute_Area end address</td>
<td>×</td>
</tr>
<tr>
<td>Not used</td>
<td>RSPAGE7 (area 7 start)</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>REPAGE7 (area 7 end)</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Background area</td>
<td>MPBAC</td>
<td>Entire memory area other than the above areas</td>
<td>×</td>
</tr>
</tbody>
</table>

[Legend] ○ indicates enabled; × indicates disabled.

Note: 1. Access information has no effect for areas accessed in supervisor mode, in which the MPU memory-protection function is disabled.
4. Hardware

4.1 Hardware Configuration

In the example presented in this application note, the access method and access execution are manipulated by using switches SW1 to SW3. LEDs are used to show the operating status of the program.

Figure 4.1 shows a connection example for this application note.

![Figure 4.1 Connection Example](image)

4.2 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>P02</td>
<td>Output</td>
<td>LED0 control</td>
</tr>
<tr>
<td>P03</td>
<td>Output</td>
<td>LED1 control</td>
</tr>
<tr>
<td>P05</td>
<td>Output</td>
<td>LED2 control</td>
</tr>
<tr>
<td>P34</td>
<td>Output</td>
<td>LED3 control</td>
</tr>
<tr>
<td>P00</td>
<td>Input</td>
<td>SW1</td>
</tr>
<tr>
<td>P01</td>
<td>Input</td>
<td>SW2</td>
</tr>
<tr>
<td>P07</td>
<td>Input</td>
<td>SW3</td>
</tr>
</tbody>
</table>
5. Software

In the sample code the type of access by the CPU to the specified area is fixed at read access. To change the type of access by the CPU, it is necessary to change the definition of PRG_TYPE in the header file main.h. Figure 5.1 shows the portion of the header file main.h in which PRG_TYPE is defined. The type of access performed by the CPU can be selected among “read,” “write,” and “execute” by changing the PRG_TYPE definition shown in figure 5.1 and then recompiling sample code. Table 5.1 lists the correspondences between the PRG_TYPE definition and the operation of the sample code.

Figure 5.1 Location for Changing Access Method in Sample Code (Example with Read Access Selected)

```c
// Excerpt from file main.h
#define READ_ACCESS (0x00000008)
#define WRITE_ACCESS (0x00000004)
#define EXECUTE_ACCESS (0x00000002)
#define PRG_TYPE READ_ACCESS
```

Table 5.1 Correspondences between PRG_TYPE Definition and Operation of Sample Code

<table>
<thead>
<tr>
<th>Definition</th>
<th>Sample Code Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_ACCESS</td>
<td>Fixed at read access to specified area</td>
</tr>
<tr>
<td>WRITE_ACCESS</td>
<td>Fixed at write access to specified area</td>
</tr>
<tr>
<td>EXECUTE_ACCESS</td>
<td>Fixed at execute access to specified area</td>
</tr>
</tbody>
</table>

A description of the software is provided below. The only portion of the program that changes when conditional compilation is used to switch between read, write, and execute access is described in 5.7.9, Function for Accessing Specified Access Area. Flowcharts of operation when different conditional compilation options are used are shown in 5.7.9, Function for Accessing Specified Access Area.
5.1 **Operation Overview**

The sample code accesses the individual memory areas listed in Table 3.1 and makes section allocation and memory-protection area settings as shown in Figure 3.1. Figure 5.2 is a sequence diagram. When the program is run it enters the access method setting mode. The initial settings specify monitor area A as the access area and enable the area search function.

![Sequence Diagram](image)

**Figure 5.2 Sequence Diagram**
5.1.1 Access Method Setting Mode

(1) SW1 Pressed
Each press of SW1 cycles the area to be accessed one step forward in the following sequence: monitor area A (memory-protection area 4) → monitor area B (memory-protection area 5) → monitor area C (memory-protection area 6) → monitor area A (memory-protection area 4). Figure 5.3 illustrates the operation when SW1 is pressed.

![Figure 5.3 SW1 Pressed](image)

(2) SW2 Pressed
Each press of SW2 toggles between the area search function enabled and disabled states. When the area search function is enabled, the area search function is used on the addresses specified beforehand to be accessed. Access control violations can be avoided, and accesses limited to permitted areas only, by checking the access-control information. Figure 5.4 illustrates the operation when SW2 is pressed.

![Figure 5.4 SW2 Pressed](image)
(3) Program Status Display
Figure 5.5 shows the program status display indications in the access method setting mode.

<table>
<thead>
<tr>
<th>Access Area</th>
<th>Area Search Function</th>
<th>Access Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED0</td>
<td>LED1</td>
<td>LED2</td>
</tr>
<tr>
<td>Monitor area A</td>
<td>Enabled</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Monitor area B</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Monitor area C</td>
<td>Enabled</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>—</td>
</tr>
</tbody>
</table>

Figure 5.5 Program Status Display Indications in Access Method Setting Mode
5.1.2 Access Result Display Mode

(1) SW3 Pressed

When SW3 is pressed in access method setting mode, the program enters access result display mode, performs an access of the specified type, and displays the access result. Figure 5.6 illustrates the operation when SW3 is pressed.

If no memory-protection error occurs, pressing SW1 or SW2 in access result display mode causes the program to return to access method setting mode. Pressing SW1 switches the area to be accessed, and pressing SW2 toggles the area search function enabled/disabled state, after which the program enters access method setting mode.

If a memory-protection error occurs, pressing SW3 in access result display mode causes the program to retry the same access method, after which the access result is displayed.

![Figure 5.6 SW3 Pressed](image-url)
(2) Program Status Display

Figure 5.7 shows the program status display indications in access result display mode.

<table>
<thead>
<tr>
<th>Access Area</th>
<th>Area Search Function</th>
<th>Access Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED0</td>
<td>LED1</td>
<td>LED2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monitor area A</td>
<td>Enabled</td>
<td>Prohibited access type detected, avoided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted access type performed.</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>Error occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No error.</td>
</tr>
<tr>
<td>Monitor area B</td>
<td>Enabled</td>
<td>Prohibited access type detected, avoided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted access type performed.</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>Error occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No error.</td>
</tr>
<tr>
<td>Monitor area C</td>
<td>Enabled</td>
<td>Prohibited access type detected, avoided.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Permitted access type performed.</td>
</tr>
<tr>
<td></td>
<td>Disabled</td>
<td>Error occurred.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No error.</td>
</tr>
</tbody>
</table>

Figure 5.7 Program Status Display Indications in Access Result Display Mode
5.2 File Composition

Table 5.2 lists the files containing the sample code. Note that files that are generated automatically by the integrated development environment and are used without modification are not listed. (Changes resulting from alterations to typedefine.h are also omitted.

Table 5.2 Files Used in the Sample Code

<table>
<thead>
<tr>
<th>File Name</th>
<th>Outline</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>main.h</td>
<td>Definition file</td>
<td></td>
</tr>
<tr>
<td>resetprg.c</td>
<td>Initial setting processing after reset</td>
<td></td>
</tr>
<tr>
<td>main.c</td>
<td>Main processing routine</td>
<td></td>
</tr>
<tr>
<td>initialize.c</td>
<td>Initial settings</td>
<td></td>
</tr>
<tr>
<td>vect.h</td>
<td>Interrupt handler declarations</td>
<td>Excep_Memory_ProtectionError function declaration added.</td>
</tr>
<tr>
<td>vecttbl.c</td>
<td>Fixed vector interrupt vector table</td>
<td>Excep_Memory_ProtectionError function added to access exception handler.</td>
</tr>
<tr>
<td>intprg.c</td>
<td>Vector function definitions</td>
<td>This file is generated automatically by the integrated development environment and contains the code of various interrupt handlers. In the example program presented in this application note, the Excep_Memory_ProtectionError function is deleted from this file and the functions Excep_BRK and Excep_CMTU0_CMT0 are modified.</td>
</tr>
<tr>
<td>Excep_Memory_ProtectionError.src</td>
<td>Access exception handler</td>
<td>Reads the instruction address when a memory-protection error occurs.</td>
</tr>
<tr>
<td>MPU_Error.c</td>
<td>Error information storage and error display when memory-protection error occurs</td>
<td></td>
</tr>
<tr>
<td>stacksct.h</td>
<td>Stack size setting</td>
<td></td>
</tr>
<tr>
<td>iodefine_mpu.h</td>
<td>Definition file of registers related to memory-protection function</td>
<td>Added because MPU-related registers are not included in iodefine.h version 2.0.</td>
</tr>
</tbody>
</table>
## 5.3 Constants

Table 5.3 and Table 5.4 list the constants used in the sample code.

### Table 5.3 Constants Used in Sample Code (1)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ_ACCESS</td>
<td>000000008h</td>
<td>Sample code for read access</td>
</tr>
<tr>
<td>WRITE_ACCESS</td>
<td>000000004h</td>
<td>Sample code for write access</td>
</tr>
<tr>
<td>EXECUTE_ACCESS</td>
<td>000000002h</td>
<td>Sample code for execute access</td>
</tr>
<tr>
<td>PRG_TYPE</td>
<td>READ_ACCESS</td>
<td>Selects sample code for read access. (See table 5.1.)</td>
</tr>
<tr>
<td>SETUP_STATUS</td>
<td>0</td>
<td>Sample code execution is in access method setting mode.</td>
</tr>
<tr>
<td>RESULT_STATUS</td>
<td>1</td>
<td>Sample code execution is in access result display mode.</td>
</tr>
<tr>
<td>SET</td>
<td>1</td>
<td>Set to 1.</td>
</tr>
<tr>
<td>CLEAR</td>
<td>0</td>
<td>Clear to 0.</td>
</tr>
<tr>
<td>NO_INPUT</td>
<td>0</td>
<td>No switch input</td>
</tr>
<tr>
<td>PUSH_DOWN_SW1</td>
<td>01h</td>
<td>SW1 pressed</td>
</tr>
<tr>
<td>PUSH_DOWN_SW2</td>
<td>02h</td>
<td>SW2 pressed</td>
</tr>
<tr>
<td>PUSH_DOWN_SW3</td>
<td>80h</td>
<td>SW3 pressed</td>
</tr>
<tr>
<td>SW_VALID</td>
<td>1</td>
<td>Indicates valid switch press.</td>
</tr>
<tr>
<td>SW_CHAT_TIMES</td>
<td>3</td>
<td>Comparison count to determine valid switch input signal</td>
</tr>
<tr>
<td>SW_FIX</td>
<td>0</td>
<td>Switch input final determination</td>
</tr>
<tr>
<td>ERROR_STATUS_CLEAR</td>
<td>01h</td>
<td>Clears memory-protection error status register.</td>
</tr>
<tr>
<td>NO_ERROR</td>
<td>0</td>
<td>No memory-protection error generated.</td>
</tr>
<tr>
<td>MPU_ERROR_MASK</td>
<td>000000007h</td>
<td>Mask for memory-protection error status</td>
</tr>
<tr>
<td>READ_ERROR</td>
<td>000000002h</td>
<td>Operand access memory-protection error (read) generated.</td>
</tr>
<tr>
<td>WRITE_ERROR</td>
<td>000000006h</td>
<td>Operand access memory-protection error (write) generated.</td>
</tr>
<tr>
<td>EXECUTE_ERROR</td>
<td>00000001h</td>
<td>Instruction memory-protection error generated.</td>
</tr>
<tr>
<td>RWE_MASK</td>
<td>00000000Eh</td>
<td>Mask for read, write, execute enable/disable</td>
</tr>
<tr>
<td>READ_MASK</td>
<td>000000008h</td>
<td>Mask for read enable/disable</td>
</tr>
<tr>
<td>WRITE_MASK</td>
<td>000000004h</td>
<td>Mask for write enable/disable</td>
</tr>
<tr>
<td>EXECUTE_MASK</td>
<td>000000002h</td>
<td>Mask for execute enable/disable</td>
</tr>
<tr>
<td>ALL_ENABLE</td>
<td>00000000Eh</td>
<td>Read, write, execute enabled.</td>
</tr>
<tr>
<td>READ_ENABLE</td>
<td>000000008h</td>
<td>Read enabled.</td>
</tr>
<tr>
<td>WRITE_ENABLE</td>
<td>000000004h</td>
<td>Write enabled.</td>
</tr>
<tr>
<td>EXECUTE_ENABLE</td>
<td>000000002h</td>
<td>Execute enabled.</td>
</tr>
<tr>
<td>DISABLE</td>
<td>0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>AREA_SEARCH_ENABLE</td>
<td>001h</td>
<td>Area search function enabled.</td>
</tr>
<tr>
<td>AREA_A</td>
<td>0</td>
<td>Monitor area A</td>
</tr>
<tr>
<td>AREA_B</td>
<td>AREA_A + 1</td>
<td>Monitor area B</td>
</tr>
<tr>
<td>AREA_C</td>
<td>AREA_B + 1</td>
<td>Monitor area C</td>
</tr>
</tbody>
</table>
### Table 5.4 Constants Used in Sample Code (2)

<table>
<thead>
<tr>
<th>Constant Name</th>
<th>Setting Value</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED0_MASK</td>
<td>04h</td>
<td>Mask for LED0 control</td>
</tr>
<tr>
<td>LED1_MASK</td>
<td>08h</td>
<td>Mask for LED1 control</td>
</tr>
<tr>
<td>LED2_MASK</td>
<td>20h</td>
<td>Mask for LED2 control</td>
</tr>
<tr>
<td>LED3_MASK</td>
<td>10h</td>
<td>Mask for LED3 control</td>
</tr>
<tr>
<td>SET_BLINK_CNT</td>
<td>20</td>
<td>LED blink period setting</td>
</tr>
<tr>
<td>BLINK_TIMING</td>
<td>0</td>
<td>LED blink timing determination</td>
</tr>
<tr>
<td>MPU_REG_MASK</td>
<td>FFFFFFF0h</td>
<td>Mask for MPU register write</td>
</tr>
<tr>
<td>SCOPE_ICLK</td>
<td>00000000h</td>
<td>Sets I clock multiplication ratio to ×8.</td>
</tr>
<tr>
<td>SCOPE_PCLK</td>
<td>00000100h</td>
<td>Sets P clock multiplication ratio to ×4.</td>
</tr>
<tr>
<td>SCOPE_BCLK</td>
<td>00020000h</td>
<td>Sets B clock multiplication ratio to ×2.</td>
</tr>
<tr>
<td>LED3_ACTIVE</td>
<td>10h</td>
<td>Sets to output bit corresponding to LED3.</td>
</tr>
<tr>
<td>SET_PORT0_DDR</td>
<td>2Ch</td>
<td>Sets to output ports for LED0, LED1, and LED2; sets to input ports for SW1, SW2, and SW3.</td>
</tr>
<tr>
<td>TURN_OFF_LED_0_1_2</td>
<td>2Ch</td>
<td>Turns off LED0, LED1, and LED2.</td>
</tr>
<tr>
<td>TURN_OFF_LED_3</td>
<td>10h</td>
<td>Turns off LED3.</td>
</tr>
<tr>
<td>ENABLE_INPUT_BUF_SW1</td>
<td>01h</td>
<td>Enables input control buffer for SW1.</td>
</tr>
<tr>
<td>ENABLE_INPUT_BUF_SW2</td>
<td>02h</td>
<td>Enables input control buffer for SW2.</td>
</tr>
<tr>
<td>ENABLE_INPUT_BUF_SW3</td>
<td>08h</td>
<td>Enables input control buffer for SW3.</td>
</tr>
<tr>
<td>ACCESS_CONTROL_INF0</td>
<td>0Dh</td>
<td>Memory-protection area 0 access-control information</td>
</tr>
<tr>
<td>ACCESS_CONTROL_INF1</td>
<td>0Dh</td>
<td>Memory-protection area 1 access-control information</td>
</tr>
<tr>
<td>ACCESS_CONTROL_INF2</td>
<td>03h</td>
<td>Memory-protection area 2 access-control information</td>
</tr>
<tr>
<td>ACCESS_CONTROL_INF3</td>
<td>03h</td>
<td>Memory-protection area 3 access-control information</td>
</tr>
<tr>
<td>ACCESS_CONTROL_INF4</td>
<td>09h</td>
<td>Memory-protection area 4 access-control information</td>
</tr>
<tr>
<td>ACCESS_CONTROL_INF5</td>
<td>05h</td>
<td>Memory-protection area 5 access-control information</td>
</tr>
<tr>
<td>ACCESS_CONTROL_INF6</td>
<td>03h</td>
<td>Memory-protection area 6 access-control information</td>
</tr>
<tr>
<td>IO_DEFINE_STARTADDRESS</td>
<td>0008C000h</td>
<td>I/O register start address in user mode</td>
</tr>
<tr>
<td>IO_DEFINE_ENDADDRESS</td>
<td>0008C100h</td>
<td>I/O register end address in user mode</td>
</tr>
<tr>
<td>BACKGROUND_ACCESS_CONTROL</td>
<td>0000h</td>
<td>Background area access-control information</td>
</tr>
<tr>
<td>MPU_ENABLE</td>
<td>0001h</td>
<td>MPU enabled.</td>
</tr>
<tr>
<td>E_OK</td>
<td>0</td>
<td>MPU register successful write</td>
</tr>
<tr>
<td>WRITE_FAULT</td>
<td>1</td>
<td>MPU register write failure</td>
</tr>
<tr>
<td>SYSTEM_TIME</td>
<td>3A97h</td>
<td>System timer period setting (10 ms)</td>
</tr>
</tbody>
</table>
### 5.4 Variables
Table 5.5 lists the global variables, Table 5.6 the static variables, and Table 5.7 the const variables.

#### Table 5.5 Global Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>g_rd_isp</td>
<td>Stores the PSW save data when changing from user mode to supervisor mode.</td>
<td>Change_PSW_PM_to_SupervisorMode, Excep_BRK</td>
</tr>
<tr>
<td></td>
<td>g_mpu_err_status</td>
<td>Stores the memory-protection error status.</td>
<td>chip_init, output_resultLED, MPU_Error</td>
</tr>
<tr>
<td></td>
<td>g_mpu_err_instruction_adrs</td>
<td>Stores the address of an instruction that generated a memory-protection error.</td>
<td>chip_init, Excep_Memory_ProtectionError</td>
</tr>
<tr>
<td></td>
<td>g_mpu_err_operand_adrs</td>
<td>Stores the address of an operand access that generated a memory-protection error.</td>
<td>chip_init, MPU_Error</td>
</tr>
<tr>
<td></td>
<td>g_mpu_err_area</td>
<td>Stores the area that generated a memory-protection error and the access-control information of the area.</td>
<td>chip_init, MPU_Error</td>
</tr>
<tr>
<td></td>
<td>g_mpu_hit_area</td>
<td>Stores the area where a search hit occurred for an address with the area search function and the access-control information of the area.</td>
<td>chip_init, Memory_Protection, output_resultLED, Area_Access</td>
</tr>
<tr>
<td>uint8_t</td>
<td>g_access_area</td>
<td>Stores the area data for an access. (AREA_A: monitor area A, AREA_B: monitor area B, AREA_C: monitor area C)</td>
<td>chip_init, Memory_Protection, output_setupLED, Area_Access_Select</td>
</tr>
<tr>
<td></td>
<td>g_area_search_flg</td>
<td>Stores the area search function enabled/disabled status. (0: Disabled, 1: Enabled)</td>
<td>chip_init, Memory_Protection, output_setupLED, output_resultLED, Area_Access_Select</td>
</tr>
<tr>
<td></td>
<td>g_system_tmr_flg</td>
<td>Stores the system timer period flag. (0: Waiting for period to elapse, 1: Period elapsed)</td>
<td>chip_init, main, Excep_CMTU0_CMT0</td>
</tr>
</tbody>
</table>
### Table 5.6  static Variables

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>static uint8_t</td>
<td>g_sw_status</td>
<td>Stores finalized switch input data.</td>
<td>main, get_sw, Memory_Protection</td>
</tr>
<tr>
<td>static uint8_t</td>
<td>g_last_sw</td>
<td>Stores previous switch input data.</td>
<td>main, get_sw</td>
</tr>
<tr>
<td>static uint8_t</td>
<td>g_sw_chat_cnt</td>
<td>Switch chattering elimination counter</td>
<td>main, get_sw</td>
</tr>
<tr>
<td>static uint8_t</td>
<td>g_sw_update</td>
<td>Stores switch input data finalization update status. (0: Not updated, 1: Updated)</td>
<td>main, get_sw, Memory_Protection, control_LED</td>
</tr>
<tr>
<td>static uint8_t</td>
<td>g_led_cycle_cnt</td>
<td>LED blink counter</td>
<td>main, control_LED</td>
</tr>
<tr>
<td>static uint8_t</td>
<td>g_prg_status</td>
<td>Stores program status.</td>
<td>main, Memory_Protection, control_LED</td>
</tr>
<tr>
<td>uint32_t</td>
<td>g_Area_A Operand</td>
<td>Monitor area A operand</td>
<td>areaAccess_Select</td>
</tr>
<tr>
<td>uint32_t</td>
<td>g_Area_B Operand</td>
<td>Monitor area B operand</td>
<td>areaAccess_Select</td>
</tr>
</tbody>
</table>

### Table 5.7  const Variable

<table>
<thead>
<tr>
<th>Type</th>
<th>Variable Name</th>
<th>Contents</th>
<th>Function Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint32_t</td>
<td>g_Area_C_operand</td>
<td>Monitor area C operand</td>
<td>areaAccess_Select</td>
</tr>
</tbody>
</table>
## 5.5 Functions

Table 5.8 lists the functions.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerON_Reset_PC</td>
<td>Startup program</td>
</tr>
<tr>
<td>Main</td>
<td>Main processing routine</td>
</tr>
<tr>
<td>get_sw</td>
<td>Switch input detect</td>
</tr>
<tr>
<td>Memory_Protection</td>
<td>Access area and access method setting and access control</td>
</tr>
<tr>
<td>control_LED</td>
<td>Valid switch input LED display update and LED blink update</td>
</tr>
<tr>
<td>output_setupLED</td>
<td>Access method setting LED control</td>
</tr>
<tr>
<td>output_resultLED</td>
<td>Access result display LED control</td>
</tr>
<tr>
<td>areaAccess_Select</td>
<td>Specified access area determination</td>
</tr>
<tr>
<td>Area_Access</td>
<td>Function for accessing specified access area</td>
</tr>
<tr>
<td>areaSearch</td>
<td>Acquisition of area corresponding to address specified by area search function and access-control information of area</td>
</tr>
<tr>
<td>operand_Access_Read</td>
<td>Function for reading specified operand</td>
</tr>
<tr>
<td>operand_Access_Write</td>
<td>Function for writing to specified operand</td>
</tr>
<tr>
<td>Area_A_prg</td>
<td>Function for executing program in monitor area A</td>
</tr>
<tr>
<td>Area_B_prg</td>
<td>Function for executing program in monitor area B</td>
</tr>
<tr>
<td>Area_C_prg</td>
<td>Function for executing program in monitor area C</td>
</tr>
<tr>
<td>Change_PSW_PM_to_UserMode</td>
<td>Change from supervisor mode to user mode</td>
</tr>
<tr>
<td>Change_PSW_PM_to_SuperVisorMode</td>
<td>Change from user mode to supervisor mode</td>
</tr>
<tr>
<td>init</td>
<td>Call initial settings functions</td>
</tr>
<tr>
<td>chip_init</td>
<td>MCU initial settings, operating clock settings, RAM initialization</td>
</tr>
<tr>
<td>mpu_init</td>
<td>MPU initialization</td>
</tr>
<tr>
<td>verify_mpu_reg</td>
<td>Verify MPU-related registers</td>
</tr>
<tr>
<td>cmt0_init</td>
<td>CMT0 settings</td>
</tr>
<tr>
<td>Excep_BRK</td>
<td>Unconditional trap exception handler</td>
</tr>
<tr>
<td>Excep_Memory_ProtectionError</td>
<td>Memory-protection error access exception handler</td>
</tr>
<tr>
<td>MPU_Error</td>
<td>Memory-protection error information storage and error display</td>
</tr>
<tr>
<td>Excep_CMTU0_CMT0</td>
<td>CMT0 compare match exception handler</td>
</tr>
</tbody>
</table>
5.6 Function Specifications

The specifications of the sample code functions are listed below.

**PowerON_Reset_PC**

<table>
<thead>
<tr>
<th>Outline</th>
<th>Startup program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>None</td>
</tr>
<tr>
<td>Declaration</td>
<td>void PowerON_Reset_PC(void)</td>
</tr>
<tr>
<td>Description</td>
<td>The PowerON_Reset_PC function is called after a reset. It uses embedded functions and standard library functions to set the CPU registers. Then it calls the init function and main function.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

**main**

<table>
<thead>
<tr>
<th>Outline</th>
<th>Main processing routine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>main.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>void main(void)</td>
</tr>
<tr>
<td>Description</td>
<td>This function calls the Change_PSW_PM_to_UserMode function, get_sw function, Memory_Protection function, and control_LED function.</td>
</tr>
<tr>
<td>Arguments</td>
<td>None</td>
</tr>
<tr>
<td>Return Value</td>
<td>None</td>
</tr>
</tbody>
</table>

**get_sw**

<table>
<thead>
<tr>
<th>Outline</th>
<th>Switch input detect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>main.h</td>
</tr>
<tr>
<td>Declaration</td>
<td>static uint8_t get_sw(uint8_t *p_last_sw, uint8_t *p_sw_chat_cnt, uint8_t *p_sw_status)</td>
</tr>
<tr>
<td>Description</td>
<td>This function detects input from the switches.</td>
</tr>
<tr>
<td>Arguments</td>
<td>uint8_t *p_last_sw   : Previous switch input data</td>
</tr>
<tr>
<td></td>
<td>uint8_t *p_sw_chat_cnt : Switch input signal match counter</td>
</tr>
<tr>
<td></td>
<td>uint8_t *p_sw_status : Finalized switch input data</td>
</tr>
<tr>
<td>Return Value</td>
<td>The switch input valid/invalid state is returned. CLEAR: Switch input invalid SW_VALID: Switch input valid</td>
</tr>
</tbody>
</table>
### Memory_Protection

<table>
<thead>
<tr>
<th>Outline</th>
<th>Access area and access method setting and access control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>main.h</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>static void Memory_Protection( uint8_t sw_status, uint8_t sw_update, uint8_t *p_prg_status )</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Based on the switch input state indicated by the arguments, this function calls functions to switch the access area, switch the area search function enable/disable state, and access the specified area.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>uint8_t sw_status : Finalized switch input data</td>
</tr>
<tr>
<td></td>
<td>uint8_t sw_update : Switch input valid/invalid</td>
</tr>
<tr>
<td></td>
<td>uint8_t *p_prg_status : Program status</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### control_LED

<table>
<thead>
<tr>
<th>Outline</th>
<th>Valid switch input LED display update and LED blink update</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>main.h</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>static void control_LED(uint8_t sw_update, uint8_t prg_status, uint8_t *pLed_cycle_cnt)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This function controls the LEDs to display the program status, as indicated by the arguments. It also controls LED blinking.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>uint8_t sw_update : Switch input valid/invalid</td>
</tr>
<tr>
<td></td>
<td>uint8_t prg_status : Program status</td>
</tr>
<tr>
<td></td>
<td>uint8_t *pLed_cycle_cnt : LED blink control timer</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### output_setupLED

<table>
<thead>
<tr>
<th>Outline</th>
<th>Access method setting LED control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>main.h</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void output_setupLED(void)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This function performs access method setting LED control.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### output_resultLED

<table>
<thead>
<tr>
<th>Outline</th>
<th>Access result display LED control</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header</strong></td>
<td>main.h</td>
</tr>
<tr>
<td><strong>Declaration</strong></td>
<td>void output_resultLED (uint8_t prg_status, uint8_t *pLed_cycle_cnt)</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>This function performs access result display LED control.</td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>uint8_t prg_status : Program status</td>
</tr>
<tr>
<td></td>
<td>uint8_t *pLed_cycle_cnt : LED blink control timer</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
### areaAccess_Select

**Outline**
Specified access area determination

**Header**
main.h

**Declaration**
static void areaAccess_Select (void)

**Description**
This function determines the access area and calls the Area_Access function.

**Arguments**
None

**Return Value**
None

---

### areaAccess

**Outline**
Function for accessing specified access area

**Header**
main.h

**Declaration**
static void Area_Access(uint32_t *p_access_operand, void (*prg)())

**Description**
This function calls a function to access the area specified by the arguments.

**Arguments**
- uint32_t *p_access_operand : Operand address to be accessed
- void (*prg)() : Start address of execute access

**Return Value**
None

---

### areaSearch

**Outline**
Acquisition of area corresponding to address specified by area search function and access-control information of area

**Header**
None

**Declaration**
static uint32_t areaSearch(uint32_t *p_access_address)

**Description**
This function calls the Change_PSW_PM_to_SuperVisorMode function. It determines the area corresponding to the address specified by an argument and gets the access-control information of that area. Then it calls the Change_PSW_PM_to_UserMode function.

**Arguments**
- uint32_t *p_access_address : Area search function target address

**Return Value**
Area corresponding to address specified by argument and access-control information of area

---

### operand_Access_Read

**Outline**
Function for reading specified operand

**Header**
None

**Declaration**
static void operand_Access_Read(uint32_t *p_access_operand)

**Description**
This function performs read access to the address specified by an argument.

**Arguments**
- uint32_t *p_access_operand : Address of read access

**Return Value**
None

---

### operand_Access_Write

**Outline**
Function for writing to specified operand

**Header**
None

**Declaration**
static void operand_Access_Write(uint32_t *p_access_operand)

**Description**
This function performs write access to the address specified by an argument.

**Arguments**
- uint32_t *p_access_operand : Address of write access

**Return Value**
None
Area_A_prg
Outline Function for executing program in monitor area A
Header None
Declaration static void Area_A_prg(void)
Description This function executes a program in monitor area A.
Arguments None
Return Value None

Area_B_prg
Outline Function for executing program in monitor area B
Header None
Declaration static void Area_B_prg(void)
Description This function executes a program in monitor area B.
Arguments None
Return Value None

Area_C_prg
Outline Function for executing program in monitor area C
Header None
Declaration static void Area_C_prg(void)
Description This function executes a program in monitor area C.
Arguments None
Return Value None

Change_PSW_PM_to_UserMode
Outline Changing from supervisor mode to user mode
Header None
Declaration void Change_PSW_PM_to_UserMode(void)
Description This function switches the processor mode from supervisor mode to user mode.
Arguments None
Return Value None

Change_PSW_PM_to_SuperVisorMode
Outline Changing from user mode to supervisor mode
Header None
Declaration static void Change_PSW_PM_to_SuperVisorMode(void)
Description This function triggers unconditional trap exception handling. After unconditional trap exception handling ends, the processor mode returns to supervisor mode.
Arguments None
Return Value None
**init**

**Outline**
Call initial settings functions

**Header**
None

**Declaration**
void init(void)

**Description**
This function initializes peripheral functions and makes memory-protection area settings, etc., by calling the chip_init function, mpu_init function, cmt0_init function, and output_setupLED function.

**Arguments**
None

**Return Value**
None

---

**chip_init**

**Outline**
MCU initial settings, operating clock settings, RAM initialization

**Header**
main.h

**Declaration**
static void chip_init(void)

**Description**
This function makes clock settings, port settings for using the LEDs and switches, initial settings for the RAM to be used, etc.

**Arguments**
None

**Return Value**
None

---

**mpu_init**

**Outline**
MPU initialization

**Header**
iodefine_mpu.h

**Declaration**
static void mpu_init(void)

**Description**
This function makes memory-protection area settings and enables the MPU function.

**Arguments**
None

**Return Value**
None

---

**verify_mpu_reg**

**Outline**
Verify MPU-related registers

**Header**
iodefine_mpu.h

**Declaration**
static uint8_t verify_mpu_reg(volatile __evenaccess uint32_t *target, uint32_t expect_val)

**Description**
The verify_mpu_reg function confirms that the values written to the MPU registers are correct. It has no direct effect on the operation of the program.

**Arguments**
volatile __evenaccess uint32_t *p_target : Address of verify target register
uint32_t expect_val : Expected value of verified register

**Return Value**
MPU register write result
E_OK: Successful write
WRITE_FAULT: Write failure
### cmt0_init

**Outline**  
CMT0 settings

**Header**  
None

**Declaration**  
static void cmt0_init(void)

**Description**  
This function sets the system timer period.

**Arguments**  
None

**Return Value**  
None

### Excep_BRK

**Outline**  
Unconditional trap exception handler

**Header**  
vect.h

**Declaration**  
void Excep_BRK(void)

**Description**  
This function is the unconditional trap exception handler. After exception handling completes, return occurs with the CPU in supervisor mode.

**Arguments**  
None

**Return Value**  
None

### Excep_Memory_ProtectionError

**Outline**  
Memory-protection error access exception handler

**Header**  
vect.h

**Declaration**  
void Excep_Memory_ProtectionError(void)

**Description**  
The access exception handler reads the address of the instruction that triggered the memory-protection error and calls the MPU_Error function.

**Arguments**  
None

**Return Value**  
None

### MPU_Error

**Outline**  
Memory-protection error information storage and error display

**Header**  
vect.h

**Declaration**  
void MPU_Error (void)

**Description**  
After storing the memory-protection error information and displaying an error indication, this function enters an infinite loop.

**Arguments**  
None

**Return Value**  
None

### Excep_CMTU0_CMT0

**Outline**  
CMT0 compare match exception handler

**Header**  
vect.h

**Declaration**  
void Excep_CMTU0_CMT0 (void)

**Description**  
This function is the CMT0 compare match exception handler. It sets the system timer elapsed flag.

**Arguments**  
None

**Return Value**  
None
5.7 Flowcharts

5.7.1 Startup Program

Figure 5.8 is a flowchart of the startup program.

```
PowerON_Reset_PC

Set interrupt table register
  set_intb()

Set floating point status word
  set_fpsw()

Initialize RAM area sections
  _INITSCT()

Call initial setting function
  init()

Set I bit, enable interrupts
  set_psw()

Main processing routine
  main()

PSW
  I bit ← 1: Interrupts enabled

end
```

Figure 5.8 Startup Program
5.7.2 Main Processing Routine

Figure 5.9 is a flowchart of the main processing routine.

```
main
  Change to user mode
    Change_PSW_PM_to_UserMode()
  Initialize static variables
  System timer period elapsed?
    Yes
    Clear system timer period flag
    Switch input detect
      get_sw()
    Access control
      Memory_Protection()
    LED control
      control_LED()
    No
```

Figure 5.9 Main Processing Routine
5.7.3 Switch Input Detect

Figure 5.10 is a flowchart of the switch input detect function.

```
get_sw

Read port 0, to which switches are connected

SW1 input detected?
  Yes
  SW2 input detected?
    Yes
    Switch input valid?
      Yes
      Set switch input to valid
      return
      Return switch input valid/invalid state
    No
    SW3 input detected?
      Yes
      Chattering elimination processing and determination
      No (no switch input)
    No
    No
  No
  SW2 input detected?
    Yes
    Switch input valid?
      Yes
      Set switch input to valid
      return
      Return switch input valid/invalid state
    No
    SW3 input detected?
      Yes
      Chattering elimination processing and determination
      No (no switch input)
    No
    No
  No
```

Figure 5.10 Switch Input Detect
5.7.4 Access Area and Access Method Setting and Access Control

Figure 5.11 is a flowchart of the access area and access method setting and access control function.

![Flowchart of Access Area and Access Method Setting and Access Control](image-url)
5.7.5 **Valid Switch Input LED Display Update and LED Blink Update**

Figure 5.12 is a flowchart of the valid switch input LED display update and LED blink update function.

![Flowchart](image)

**Figure 5.12  Valid Switch Input LED Display Update and LED Blink Update**
5.7.6 Access Method Setting LED Control

Figure 5.13 is a flowchart of the access method setting LED control function.

![Flowchart of Access Method Setting LED Control](image-url)

**Figure 5.13** Access Method Setting LED Control
5.7.7 Access Result Display LED Control

Figure 5.14 is a flowchart of the access result display LED control function.

![Flowchart of Access Result Display LED Control](image-url)
5.7.8 Specified Access Area Determination

Figure 5.15 is a flowchart of the specified access area determination function.
5.7.9 Function for Accessing Specified Access Area

Conditional compilation is used so that the operation of this function differs according to the correspondence between the PRG_TYPE definition and the resulting sample code operation, as shown in Table 5.1. The conditional compilation conditions are described below.

(I) Read Access to Specified Area

Figure 5.16 is a flowchart of the function for accessing the specified access area (READ_ACCESS), which operates when the selection is fixed at read access to the specified area.

![Flowchart](image-url)
(2) Write Access to Specified Area

Figure 5.17 is a flowchart of the function for accessing the specified access area (WRITE_ACCESS), which operates when the selection is fixed at write access to the specified area.

![Flowchart](image-url)

**Figure 5.17 Write Access to Specified Area (WRITE_ACCESS)**
(3) **Execute Access to Specified Area**

Figure 5.18 is a flowchart of the function for accessing the specified access area (EXECUTE_ACCESS), which operates when the selection is fixed at execute access to the specified area.

![Flowchart of Execute Access to Specified Area](image-url)
5.7.10 Acquisition of Area Corresponding to Address Specified by Area Search Function and Access-Control Information of Area

Figure 5.19 is a flowchart of the function for acquisition of the area corresponding to the address specified by the area search function and access-control information of the area.

```
areaSearch

Switch to supervisor mode
Change_PSW_PM_to_SupervisorMode()

Set area search target address
MPSA ← Area search target address

Perform area search operation
MPOPS
S bit ← 1: Perform area search operation.

Read area search result
Read MHITD.

Switch to user mode
Change_PSW_PM_to_UserMode()

return
Return area search result
```
5.7.11 Function for Reading Specified Operand
Figure 5.20 is a flowchart of the function for reading a specified operand.

![Flowchart for Reading Specified Operand]

5.7.12 Function for Writing to Specified Operand
Figure 5.21 is a flowchart of the function for writing to a specified operand.

![Flowchart for Writing to Specified Operand]
5.7.13 Function for Executing Program in Monitor Area A
Figure 5.22 is a flowchart of the function for executing a program in monitor area A.

![Flowchart for Area A Program]

5.7.14 Function for Executing Program in Monitor Area B
Figure 5.23 is a flowchart of the function for executing a program in monitor area B.

![Flowchart for Area B Program]

5.7.15 Function for Executing Program in Monitor Area C
Figure 5.24 is a flowchart of the function for executing a program in monitor area C.

![Flowchart for Area C Program]
### 5.7.16 Change from Supervisor Mode to User Mode

Figure 5.25 is a flowchart of the change from supervisor mode to user mode function.

![Flowchart: Change from Supervisor Mode to User Mode](image)

#### Change_PSW_PM_to_UserMode

- Get PSW, save to R1
- Set PM bit to 1 in acquired PSW, save on stack
- Save return destination address (PC) on stack
- RTE

**Figure 5.25 Change from Supervisor Mode to User Mode**

### 5.7.17 Change from User Mode to Supervisor Mode

Figure 5.26 is a flowchart of the change from user mode to supervisor mode function.

![Flowchart: Change from User Mode to Supervisor Mode](image)

#### Change_PSW_PM_to_SupervisorMode

- Read ISP
  - get_isp()
- Generate unconditional trap exception
  - brk()
- return

Read ISP to update PSW, which is saved on the stack by the unconditional trap exception handler.

**Figure 5.26 Change from User Mode to Supervisor Mode**
5.7.18 **Call Initial Settings Functions**

Figure 5.27 is a flowchart of the function that calls the initial settings functions.

![Flowchart](image)

- **init**
  - **Initial settings**
    - chip_init()
    - **MCU initial settings, operating clock settings, RAM initialization**
    - **MPU initialization**
      - mpu_init()
    - **CMT settings**
      - cmt0_init()
    - **LED initial display**
      - output_setupLED()
  - **return**

*Figure 5.27  Call Initial Settings Functions*
5.7.19 **MCU Initial Settings, Operating Clock Settings, RAM Initialization**

Figure 5.28 is a flowchart of the function that performs MCU initial settings, operating clock settings, and RAM initialization.

```
chip_init

Set ICLK to 8×, PCLK to 4×, and BCLK to 2×
  SCKCR register
  ICK bit ← 0000b
  PCK bit ← 0001b
  BCK bit ← 0010b

Set P05, P03, and P02 output data
  PORT0.DR ← 2Ch
  Note: Output after setting as output pins

Set P34 output data
  PORT3.DR ← 10h
  Note: Output after setting as output pins

Set P05, P03, and P02 as output pins
  PORT0.DDR ← 2Ch

Set P34 as output pin
  PORT3.DDR ← 10h

Enable input buffer of P07, P01, and P00
  PORT0.ICR ← 83h

Initial RAM to be used
  g_mpu_err_status ← NO_ERROR (no error)
  g_mpu_err_instruction_adrs ← NO_ERROR (no error)
  g_mpu_err_operand_adrs ← NO_ERROR (no error)
  g_mpu_err_area ← ALL_ENABLE (read, write, execute enabled)
  g_mpu_hit_area ← ALL_ENABLE (read, write, execute enabled)
  g_access_area ← AREA_A (area A)
  g_area_search_flg ← AREA_SEARCH_ENABLE (area search enabled)
  g_system_tmr_flg ← CLEAR

return
```

Figure 5.28  MCU Initial Settings, Operating Clock Settings, RAM Initialization
5.7.20  MPU Initialization

Figure 5.29 and Figure 5.30 are flowcharts (parts 1 and 2) of the MPU initialization function.

**Figure 5.29  MPU Initialization (1)**

```
mpu_init

Set area from SU section start to SI section end as area 0. Set as read enabled, write enabled, execute disabled.

Set area from I/O register start to I/O register end as area 1. Set as read enabled, write enabled, execute disabled.

Set area from PUUser_Area section start to PUUser_Area section end as area 2. Set as read disabled, write disabled, execute enabled.

Set area from BUUser_Area section start to BUUser_Area section end as area 3. Set as read enabled, write enabled, execute disabled.

Set area from PRead_Area section start to BRead_Area section end as area 4. Set as read enabled, write disabled, execute disabled.

Set area from PWrite_Area section start to BWrite_Area section end as area 5. Set as read disabled, write enabled, execute disabled.

Set area from PExecute_Area section start to CExecute_Area section end as area 6. Set as read disabled, write disabled, execute enabled.

RSPAGE0
RSPN ← SU section start address
REPAGE0
REPN ← SI section end address
UAC bit ← 110b: Read enabled, write enabled, execute disabled
V bit ← 1: Area setting enabled

RSPAGE1
RSPN ← I/O register start address used in user mode
REPAGE1
REPN ← I/O register end address used in user mode
UAC bit ← 110b: Read enabled, write enabled, execute disabled
V bit ← 1: Area setting enabled

RSPAGE2
RSPN ← User program area start address
REPAGE2
REPN ← User program area end address
UAC bit ← 001b: Read disabled, write disabled, execute enabled
V bit ← 1: Area setting enabled

RSPAGE3
RSPN ← User RAM area start address
REPAGE3
REPN ← User RAM area end address
UAC bit ← 110b: Read enabled, write enabled, execute disabled
V bit ← 1: Area setting enabled

RSPAGE4
RSPN ← Area A start address
REPAGE4
REPN ← Area A end address
UAC bit ← 100b: Read enabled, write disabled, execute disabled
V bit ← 1: Area setting enabled

RSPAGE5
RSPN ← Area B start address
REPAGE5
REPN ← Area B end address
UAC bit ← 010b: Read disabled, write enabled, execute disabled
V bit ← 1: Area setting enabled

RSPAGE6
RSPN ← Area B start address
REPAGE6
REPN ← Area B end address
UAC bit ← 001b: Read disabled, write disabled, execute enabled
V bit ← 1: Area setting enabled
```
Set background area as read disabled, write disabled, execute disabled

Enable memory-protection function

Verify MPU registers
Verify_mpu_reg()

MPBAC
UBAC bit ← 000b: Read disabled, write disabled, execute disabled

MPEN
MPEN bit ← 1: Memory-protection function enabled

Perform a verification check on the above MPU-related registers to confirm that the written values were set correctly.

Figure 5.30 MPU Initialization (2)
5.7.21 Verify MPU-Related Registers

Figure 5.31 is a flowchart of the verify MPU-related registers function.

```
verify_mpu_reg  [Arguments]
volatile __evenaccess uint32_t *p_target: Verify target address
uint32_t expect_val: Expected value

Read verify target address value

Matches expected value?

No

Yes

Verify result: OK

Verify result: Fail

return

Return verify result
```

Figure 5.31 Verify MPU-Related Registers
5.7.22 CMT0 Settings

Figure 5.32 is a flowchart of the CMT0 settings function.

```
cmt0_init
  → Cancel CMT0 module stop
      → MSTPCRA
          → MSTPA15 bit ← 0
  → Set CMT0 clock source to PCLK/32
      → CMCR
          → CKS bit ← 01b
  → Set CMT0 period
      → CMCOR ← SYSTEM_TIME
  → Clear CMT0.CMCNT
      → CMCNT ← 0000h
  → Set interrupt priority
      → IPR03 ← 01h
  → Clear interrupt request flag
      → IR028 ← 00h
  → Enable interrupt
      → IER03
          → IEN4 bit ← 1
  → Start count operation of CMT0.CMCNT counter
      → CMSTR0
          → STR0 ← 1
  → return
```

Figure 5.32 CMT0 Settings
5.7.23 Unconditional Trap Exception Handler

Figure 5.33 is a flowchart of the unconditional trap exception handler.

Excep_BRK

Read g_rd_isp

Get ISP information before transition to exception handling.

Set to address where PSW saved in exception handling is stored

Clear PM bit of saved PSW

After exception handling, return in supervisor mode.

return

Figure 5.33 Unconditional Trap Exception Handler
5.7.24 Memory-Protection Error Access Exception Handler

Figure 5.34 is a flowchart of the memory-protection error access exception handler.

![Flowchart of Memory-Protection Error Access Exception Handler]

- **Excep_Memory_ProtectionError**
- Save address of instruction triggering memory-protection error to `g_mpu_err_instruction_adrs`
- Save memory-protection error information and display error `MPU_Error()`
- **return**

*Figure 5.34 Memory-Protection Error Access Exception Handler*
5.7.25 Memory-Protection Error Information Storage and Error Display

Figure 5.35 is a flowchart of the function for memory-protection error information storage and error display.

![Flowchart of Memory-Protection Error Information Storage and Error Display](image-url)
5.7.26 CMT0 Compare Match Exception Handler

Figure 5.36 is a flowchart of the CMT0 compare match exception handler.

![Flowchart of CMT0 Compare Match Exception Handler](image)

Figure 5.36  CMT0 Compare Match Exception Handler
6. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

7. Reference Documents

User’s Manual: Hardware
RX62N, RX621 Group User’s Manual: Hardware Rev.1.30
The latest version can be downloaded from the Renesas Electronics website.

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

User’s Manual: Development Tools
RX Family C/C++ Compiler, Assembler, Optimizing Linkage Editor—Compiler Package V. 1.01 User’s Manual, Rev. 1.00
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/

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

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Date</th>
<th>Page</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.00</td>
<td>Jul 01, 2014</td>
<td>—</td>
<td>First edition issued</td>
</tr>
</tbody>
</table>
General Precautions in the Handling of MPU/MCU Products

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

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

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

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

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

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different type number, confirm that the change will not lead to problems.
   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
Notice

1. Descriptions of circuits, software and 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 depend on the product's quality grade, as indicated below.

   - "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home 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-crisis systems, and safety equipment etc.

   Renesas Electronics products are either intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.) or may cause serious property damages (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 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, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardwares and softwares 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 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 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 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 damages (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 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.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distribute, 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. The contents and conditions set forth in 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.

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

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

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

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