

To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: <http://www.renesas.com>

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (<http://www.renesas.com>)

Send any inquiries to <http://www.renesas.com/inquiry>.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
 - “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.
 - “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
 - “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(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.

Application Note

V850ES/Jx3

Sample Program (Interrupt)

External Interrupt Generated by Switch Input

This document summarizes the operations of the sample program and describes how to use the sample program and how to set and use the interrupt function. In the sample program, an interrupt is generated by detecting the falling edge of the switch input and an LED lighting pattern is displayed according to the number of switch inputs.

Target devices

V850ES/JG3 microcontroller

V850ES/JJ3 microcontroller

CONTENTS

CHAPTER 1 OVERVIEW	3
1.1 Initial Settings.....	3
1.2 Main processing.....	3
1.3 Interrupt Servicing	4
CHAPTER 2 CIRCUIT DIAGRAM.....	5
2.1 Circuit Diagram	5
2.2 Peripheral Hardware	5
CHAPTER 3 SOFTWARE.....	6
3.1 File Configuration	6
3.2 On-Chip Peripheral Functions Used.....	7
3.3 Initial Settings and Operation Overview	7
3.4 Flowchart	8
3.5 Differences Between V850ES/JJ3 and V850ES/JG3	10
3.6 ROMization	10
3.7 Security ID	12
3.8 On-Chip debug with MINICUBE2	14
3.9 Describing interrupt handler by using #pragma directives.....	17
CHAPTER 4 SETTING REGISTERS	18
4.1 Setting Interrupt Pins and Pins for Lighting LEDs.....	19
4.1.1 Setting port 0 function control register (PFC0).....	20
4.1.2 Setting port 0 mode control register (PMC0)	21
4.1.3 Setting external interrupt falling and rising edge specification register 0 (INTF0, INTRO)	22
4.1.4 Setting interrupt control register (PIC0)	23
4.1.5 Setting port CM	25
4.2 Interrupt Servicing	27
CHAPTER 5 RELATED DOCUMENTS.....	28
APPENDIX A PROGRAM LIST	29

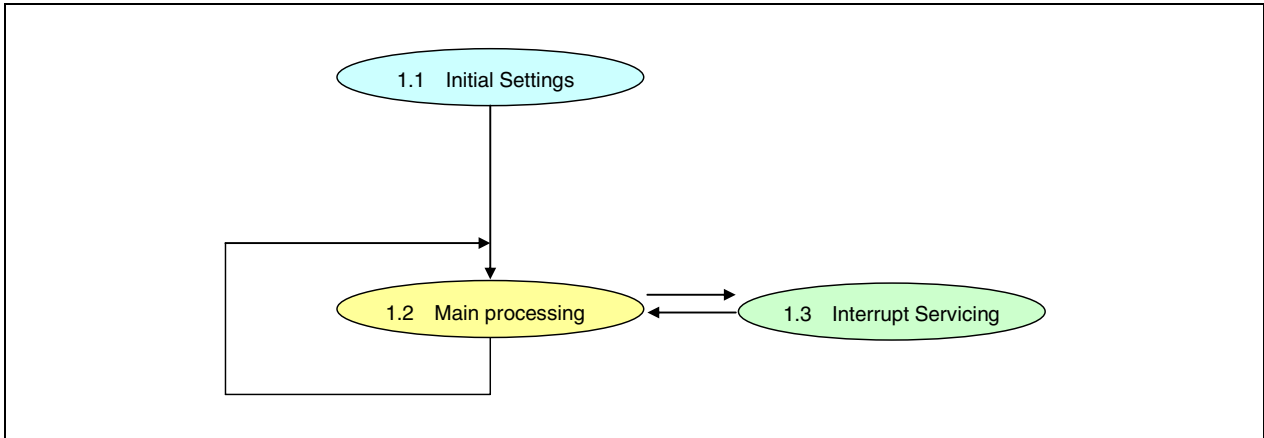
- The information in this document is current as of September, 2009. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.
 - No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
 - NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.
 - Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
 - While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
 - NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific". The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
 - "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.
 - "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).
 - "Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.
- The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.
- (Note 1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
- (Note 2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

(M8E0909)

CHAPTER 1 OVERVIEW

In this sample program, an example of using the interrupt function is presented. An LED lighting pattern is displayed according to the number of switch inputs, by detecting the falling edge of the switch input and servicing interrupts.

The main software contents are shown below.



1.1 Initial Settings

<Settings of on-chip peripheral functions>

- Setting wait operations <wait: 2> for bus access to on-chip peripheral I/O registers
- Setting on-chip debug mode register to normal operation mode
- Stopping the internal oscillator and watchdog timer 2
- Setting not to divide the CPU clock frequency
- Setting the system clock to 32 MHz by multiplying the input clock by 8 using the PLL

<Pin settings>

- Setting unused pins
- Setting external interrupt pins (edge specification, priority specification, unmasking)
- Setting LED output pins (specifying values to turn off and output from LED1 and LED2)

<ROMization>

- ROMization processing (initialization of variables with initial values)

1.2 Main processing

- Enabling interrupts by using the EI instruction
- Executing an infinite loop that executes no processing (waiting for an interrupt generated by switch input)

1.3 Interrupt Servicing

Interrupts are serviced by detecting the falling edge of the INTPO pin, caused by switch input. In interrupt servicing, the LED lighting pattern is changed by confirming that the switch is on, after about 10 ms have elapsed after the falling edge of the INTPO pin was detected.

The switch being off, after about 10 ms have elapsed after the falling edge of the INTPO pin was detected, is identified as chattering noise and the LED lighting pattern is not changed.

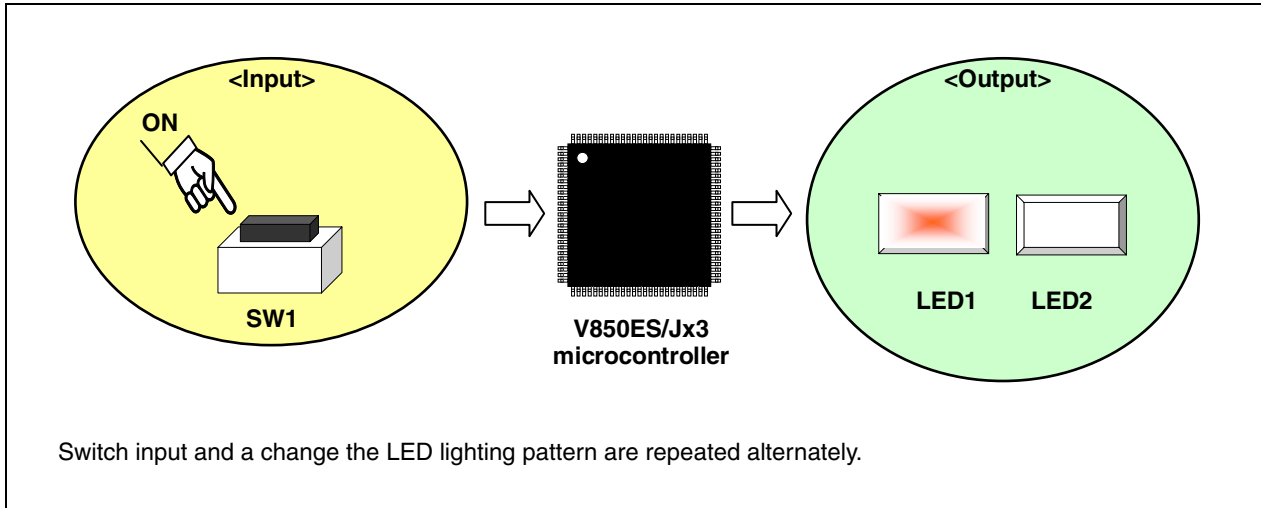


Table 1-1. LED Lighting Patterns

Switch (SW1) Input count ^{Note}	LED1	LED2
0	OFF	OFF
1	ON	OFF
2	OFF	ON
3	ON	ON

Note Inputs 0 to 3 are repeated from the fourth input.

Remark See each product user's manual (V850ES/Jx3) for cautions when using the device.



[Column] What is chattering?

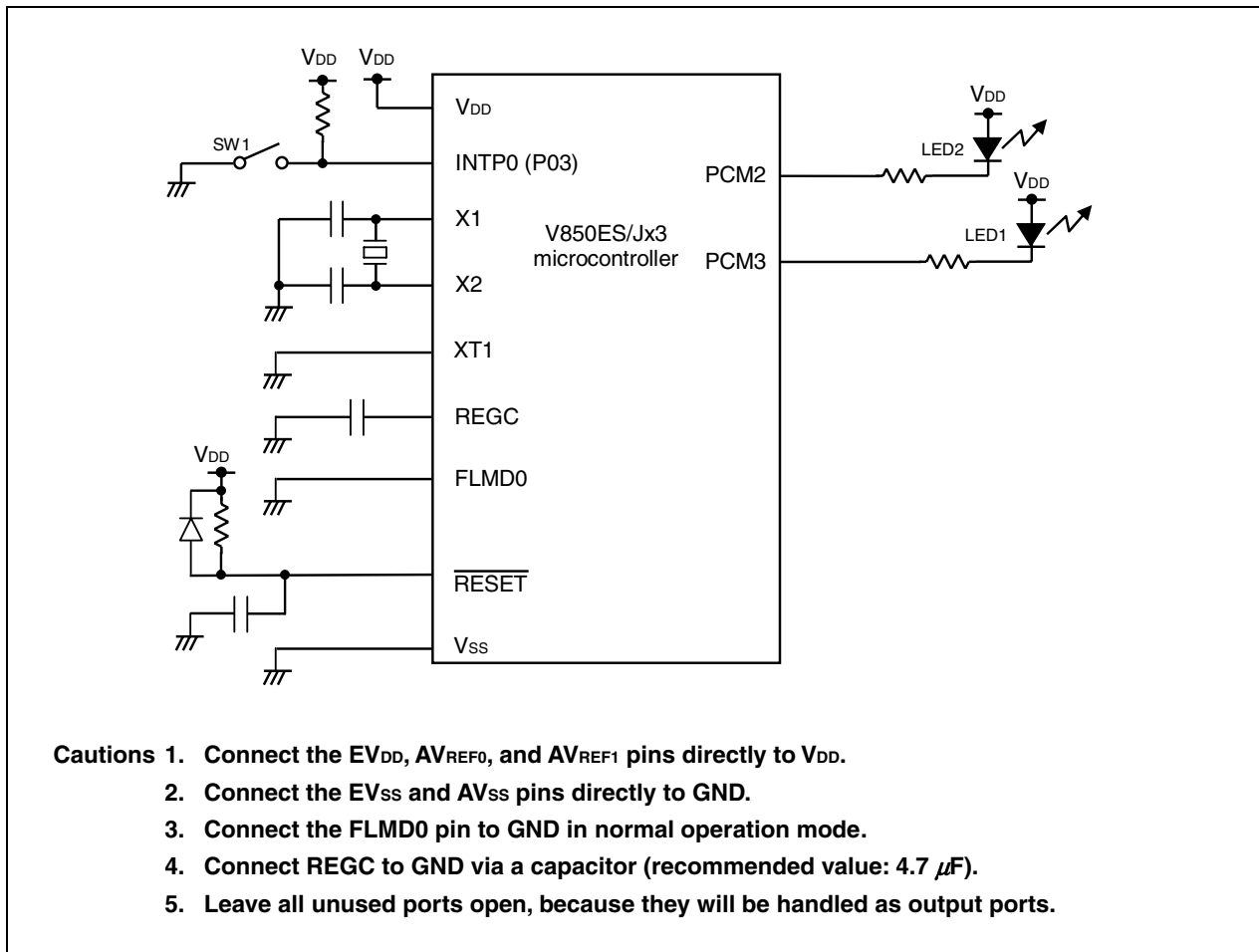
Chattering is a phenomenon that an electric signal alternates between being on and off when a connection flip-flops mechanically immediately after a switch is switched.

CHAPTER 2 CIRCUIT DIAGRAM

This chapter describes a circuit diagram and the peripheral hardware to be used in this sample program.

2.1 Circuit Diagram

The circuit diagram is shown below.



2.2 Peripheral Hardware

The peripheral hardware to be used is shown below.

(1) Switch (SW1)

This switch is used as an interrupt input to control the lighting of the LEDs.

(2) LEDs (LED1, LED2)



The LEDs are used as outputs corresponding to the number of switch inputs.

CHAPTER 3 SOFTWARE


This chapter describes the file configuration of the compressed files to be downloaded, on-chip peripheral functions of the microcontroller to be used, and the initial settings and an operation overview of the sample program. A flowchart is also shown.


3.1 File Configuration

The following table shows the file configuration of the compressed files to be downloaded.

File Name (Tree Structure)	Description	Compressed (*.zip) Files Included	
			
conf <ul style="list-style-type: none"> — crtE.s — APPNOTE_INT.dir — APPNOTE_INT.prj — APPNOTE_INT.prw 	Startup routine file ^{Note 1}	-	●
	Link directive file ^{Note 2}	●	●
	Project file for integrated development environment PM+	-	●
	Workspace file for integrated development environment PM+	-	●
src <ul style="list-style-type: none"> — main.c — minicube2.s 	C language source file including descriptions of hardware initialization processing and main processing of microcontroller	●	●
	Source file for reserving area for MINICUBE2	●	●

- Notes**
1. This is the startup file copied when “Copy sample for use (C)” is selected when “Specify startup file” is selected when creating a new workspace. (If the default installation path is used, the startup file will be a copy of C:\Program Files\NEC Electronics Tools\PM+*Version used*lib850\r32\crtE.s.)
 2. This is the link directive file automatically generated when “Copy sample for use (C)” is selected and “Memory usage: Internal memory only (I)” is checked when “Specify link directive file” is selected when creating a new workspace, and to which **a segment for MINICUBE2 is added**. (If the default installation path is used, C:\Program Files\NEC Electronics Tools\PM+*Version used*\bin\w_data\V850_i.dat is used as the reference file.)

Remark  : Only the source file is included.

 : The files to be used with integrated development environment PM+ are included.

3.2 On-Chip Peripheral Functions Used

The following on-chip peripheral functions of the microcontroller are used in this sample program.

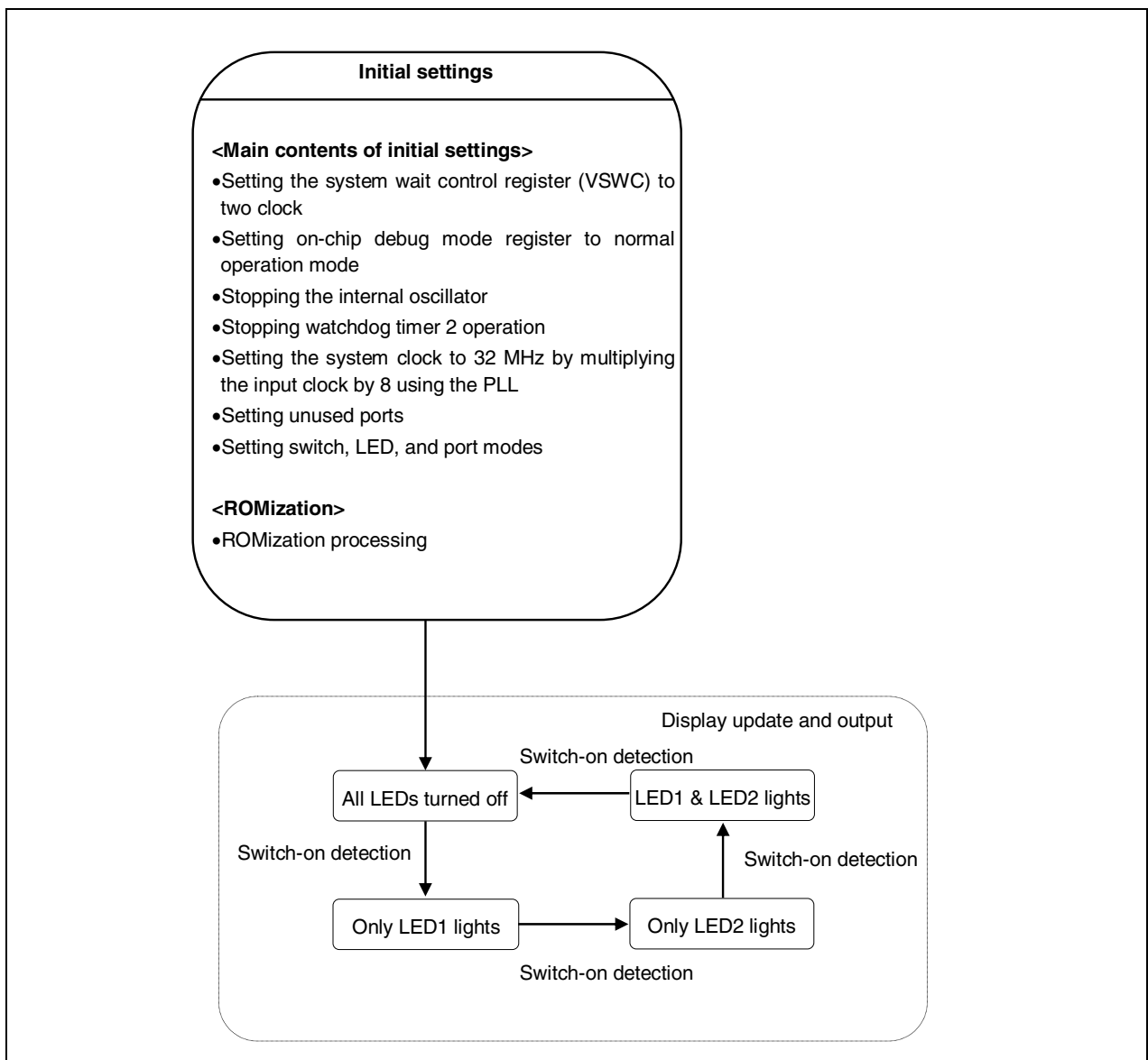
- External interrupt input (for switch input): INTP0 (SW1)
- Output ports (for lighting LEDs): PCM2 (LED2), PCM3 (LED1)

3.3 Initial Settings and Operation Overview

In this sample program, the selection of the clock frequency, setting for stopping the watchdog timer 2, setting of the I/O ports and external interrupt pins, and setting of interrupts are performed in the initial settings.

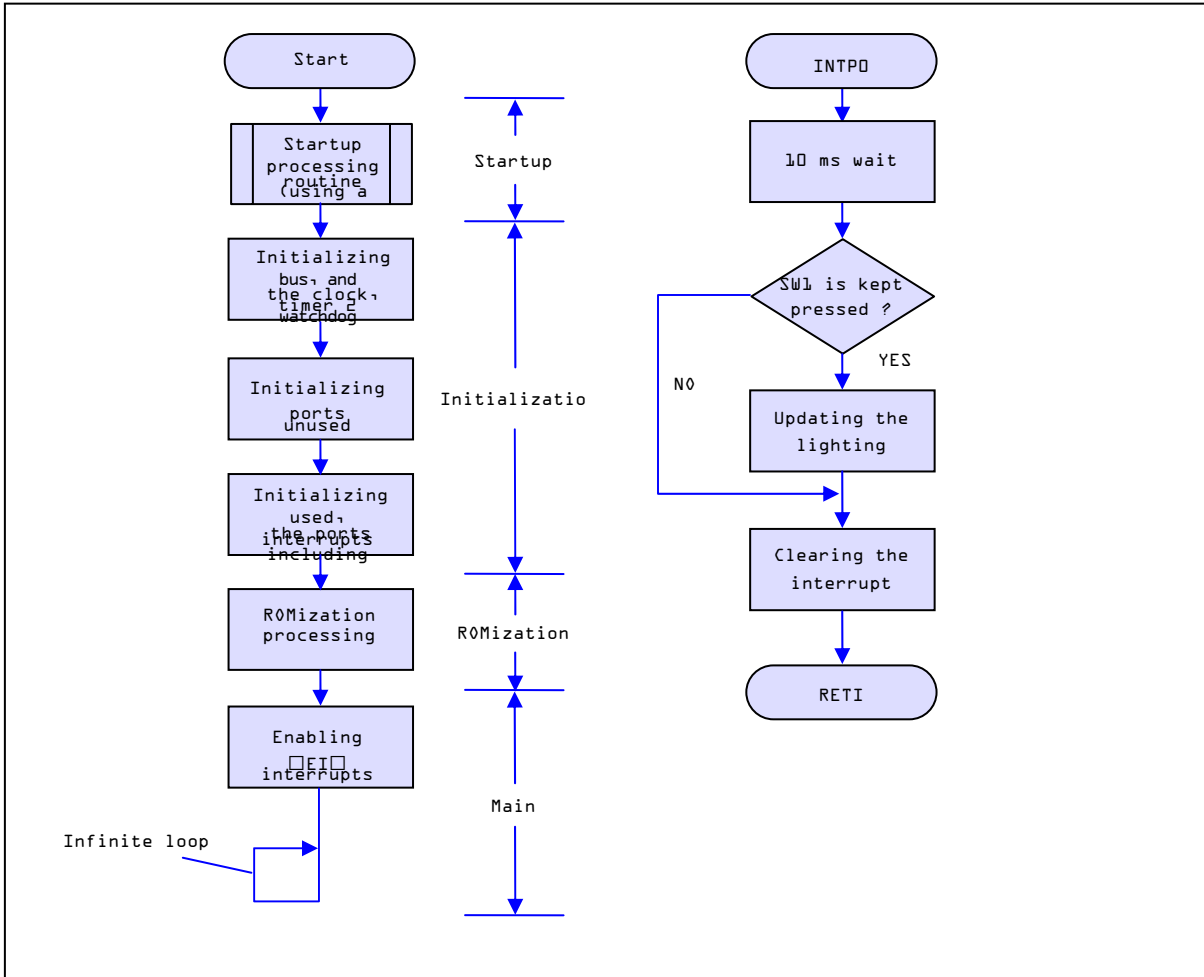
After completion of the initial settings, interrupts are serviced by detecting the falling edge of the switch input (SW1) and the lighting pattern of the two LEDs (LED1 and LED2) is controlled according to the number of switch inputs.

The details are described in the state transition diagram shown below.



3.4 Flowchart

A flowchart for the sample program is shown below.



**[Column] Contents of the startup routine**

The startup routine is a routine that is executed before executing the main function after the V850 is reset. Basically, the startup routine initializes the system after the system is reset. Specifically, the following are performed.

- Securing the argument area of the main function
- Securing the stack area
- Setting the RESET handler when reset is issued
- Setting the text pointer (tp)
- Setting the global pointer (gp)
- Setting the stack pointer (sp)
- Setting the element pointer (ep)
- Setting mask values to the mask registers (r20 and r21)
- Clearing the sbss and bss areas to 0
- Setting the CTBP value for the prologue epilogue runtime library of the function
- Setting r6 and r7 as arguments of the main function
- Branching to the main function

3.5 Differences Between V850ES/JJ3 and V850ES/JG3

The V850ES/JJ3 is the V850ES/JG3 with its functions, such as I/Os, timer/counters, and serial interfaces, expanded.

In this sample program, the initialization range of I/O initialization differs.

See **APPENDIX A PROGRAM LIST** for details of the sample program.

3.6 ROMization

In this sample program, ROMization information is copied after the on-chip peripheral functions are initialized.

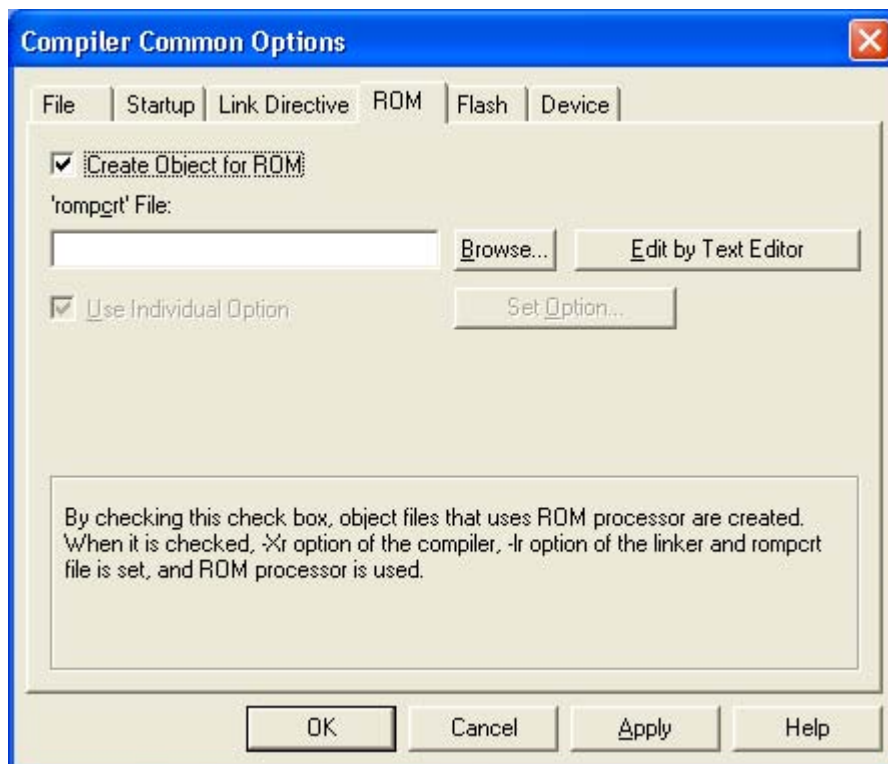
ROMization information is the information of the initial values of variables that have initial values (the section to which variables that have initial values are placed). Variables that have initial values (the section to which variables that have initial values are placed) will hold their software-based initial values for the first time by copying the ROMization information to the RAM^{Note}.

If a variable that has an initial value is used in the program to be created, ROMization information must be generated and copied. Furthermore, the ROMization information must be copied before using the variable that has an initial value.

Note The data allocated to a section that has a writable attribute is subject to packing by default in ROMization. Other data can also be packed. See the CA850 Help for details.

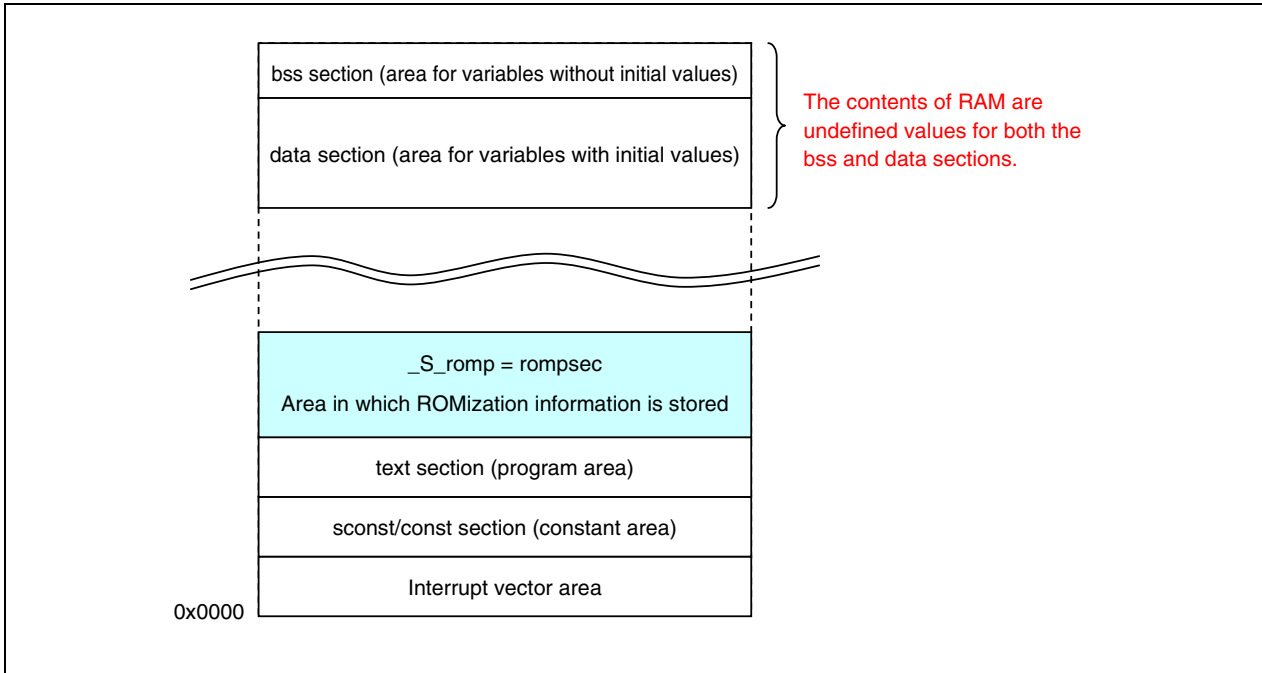
The ROMization procedure is described below.

Select the [ROM] tab, which is an option common to all PM+ compilers, and then check “Create Object for ROM”.



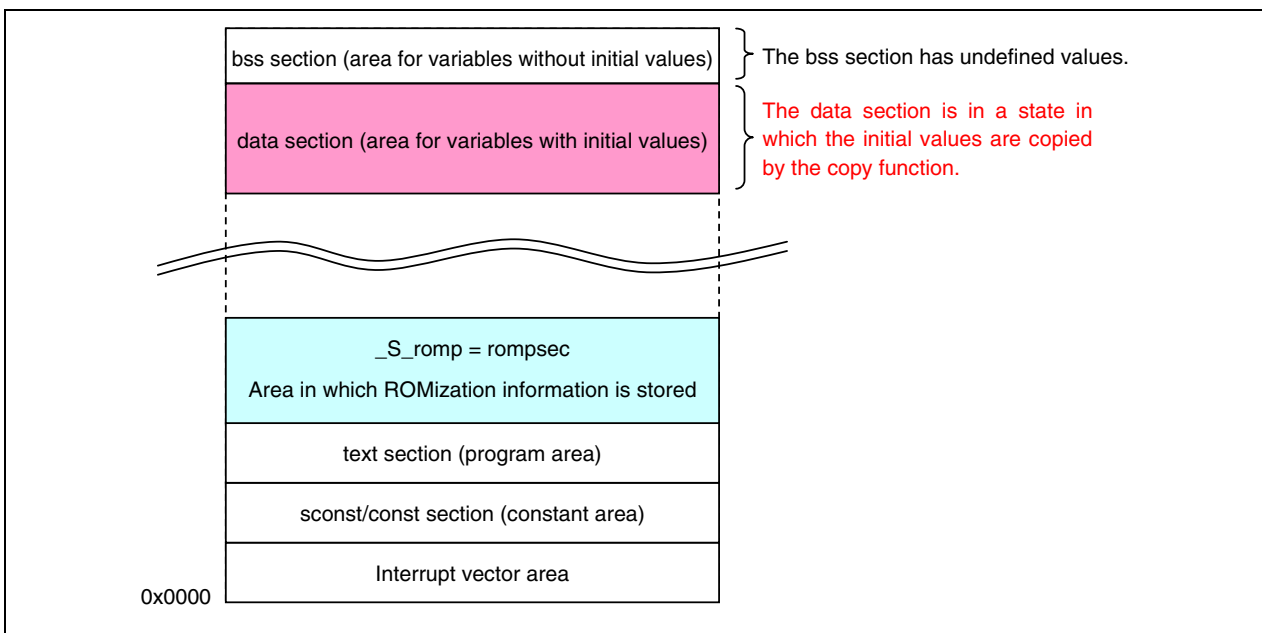
The section into which the ROMization information is to be stored (rompsec) will be automatically added immediately after the program area (.text) section. However, by checking “Create Object for ROM”, a code that indicates the same address as that of rompsec will be generated for the default label `_S_romp` defined by `romp.crt.o`, and the library `libr.a`, in which the copy function is stored, will be automatically linked.

An image of memory before the ROMization information is copied, which is created according to the procedure so far, is shown below.



The ROMization information must be copied, because the contents of the data section, which is the area for variables that have initial values will stay undefined if memory remains as is.

An image of the memory after the `_rcopy()` function is called to copy the ROMization information is shown below.



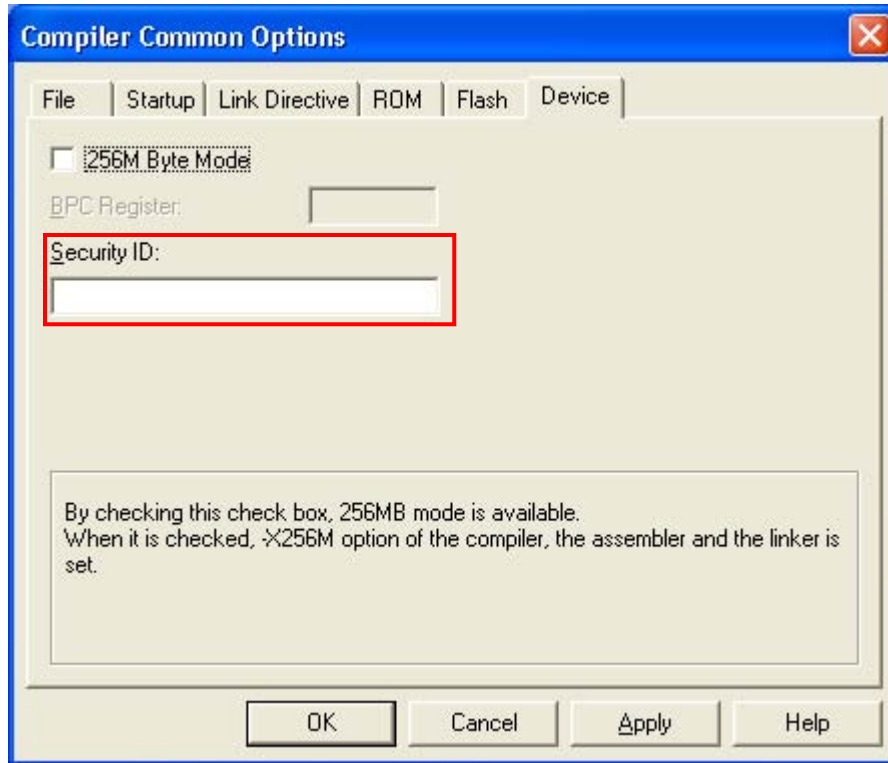
3.7 Security ID

The content of the flash memory can be protected from unauthorized reading by using a 10-byte ID code for authorization when executing on-chip debugging using an on-chip debug emulator.

The debugger authorizes the ID by comparing it with the ID code preset to the 10 bytes from 0x0000070 to 0x0000079 in the internal flash memory area.

If the IDs match, the security code will be unlocked and reading flash memory and using the on-chip debug emulator will be enabled.

In this sample program (complete-environment version), the security ID is not set and the default security ID value 0xFFFF FFFF FFFF FFFF FFFF is applied.

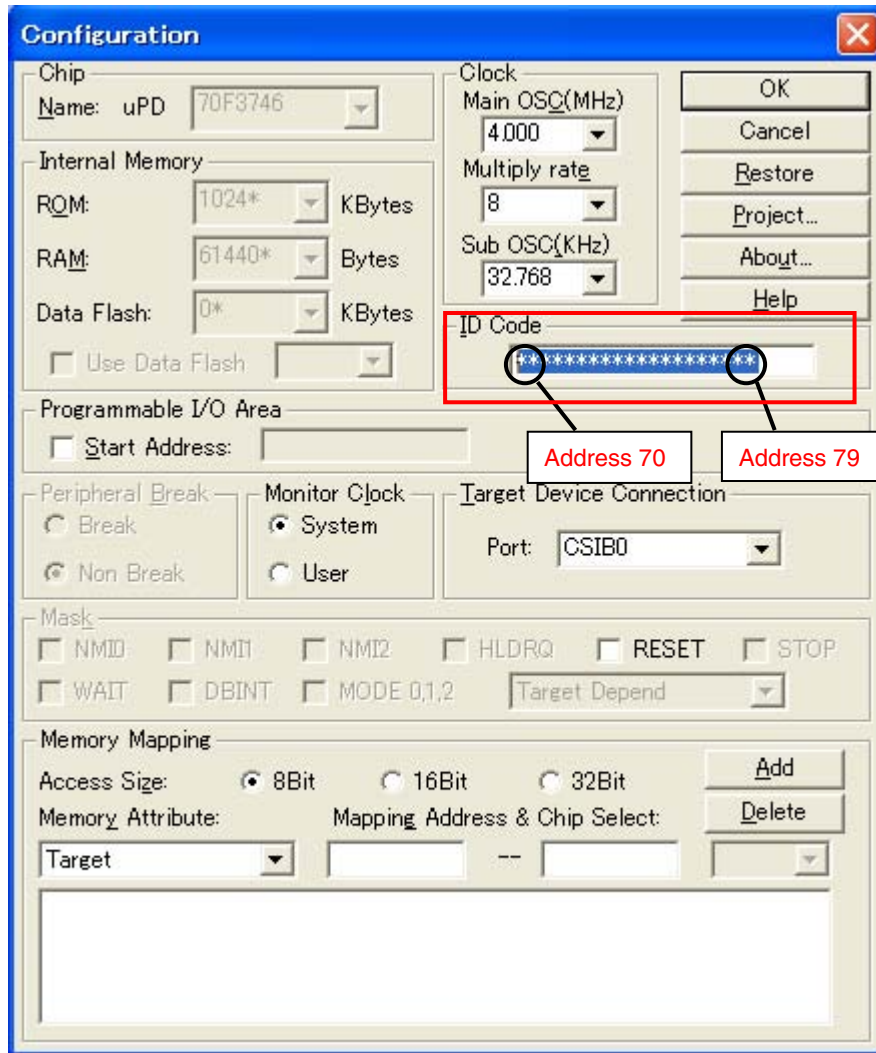


Remark Set the security ID for a device provided with flash memory in the “Security ID” field, which is an option common to all compilers.

Specify the ID as a hexadecimal number of 10 bytes or less starting with 0x.

If specifying this option or specifying the security ID by using an assembly description (.section SECURITY_ID) is omitted, 0xFFFF FFFF FFFF FFFF FFFF will be assumed to have been specified.

If a program is downloaded and operated by using this sample program (complete-environment version), 0xFF will be set to the security ID area of the microcontroller. Caution is therefore required, because the on-chip debug emulator can be used only if 0xFFFF FFFF FFFF FFFF FFFF (default value) is set in the ID code entry area when the debugger is connected the next time.



- Bit 7 (0x0000079) of the 10 bytes of the ID code is the on-chip debug emulator use enable flag (0: Disables use, 1: Enables use).
- When the on-chip debug emulator is started, the debugger requests ID entry. The debugger will be started if the ID code entered in the debugger matches the ID code embedded in addresses 0x0000070 to 0x0000079.
- Even if the ID codes match, debugging cannot be executed if the on-chip debug emulator use enable flag is set to "0".

3.8 On-Chip debug with MINICUBE2

The following describes how to set an on-chip debug using MINICUBE2 with pins for CSIB0(SIB0, SOB0, $\overline{\text{SCKB0}}$, and HS(PCM0)) as debug interfaces. These items need to be set in the user program or using the compiler options.

See the following user's manuals for details of how to set the items.

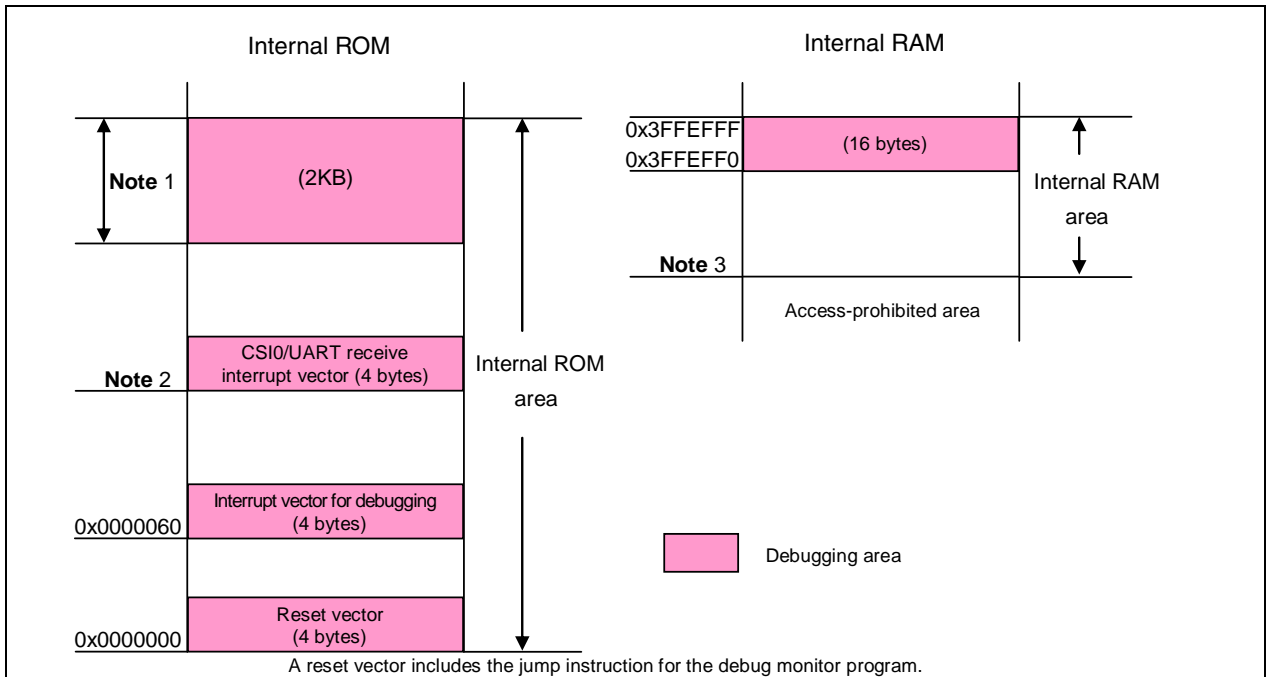
- V850ES/JJ3 32-bit Single-Chip Microcontrollers
Hardware User's Manual
- V850ES/JG3 32-bit Single-Chip Microcontrollers
Hardware User's Manual

- QB-MINI2 On-Chip Debug Emulator with Programming Function
User's Manual

The shaded portions in Figure 3-1 are the areas where the debug monitor program is allocated. The monitor program performs initialization processing for debug communication interface and RUN or break processing for the CPU. The internal ROM area must be filled with 0xFF. In using the NEC Electronics compiler CA850, it is necessary adding the assemble source file and link directive code for securing the area.

Remark It is not necessarily required to secure this area if the user program does not use this area. To avoid problems that may occur during the debugger startup, however, it is recommended to secure this area in advance, using the compiler.

Figure 3-1. Memory Spaces Where Debug Monitor Programs Are Allocated



Notes 1. Address values vary depending on the product.

	Internal ROM Size	Address Value
μPD70F3739 μPD70F3743	384 KB	0x005F800 - 0x005FFFF
μPD70F3740 μPD70F3744	512 KB	0x007F800 - 0x007FFFF
μPD70F3741 μPD70F3745	768 KB	0x00BF800 - 0x00BFFFF
μPD70F3742 μPD70F3746	1024 KB	0x00FF800 - 0x00FFFFFF

2. The Address values depending on the debug interfaces. It starts at 0x0000290 when CSIB0 is used, and at 0x00002F0 when CSIB3 is used, and at 0x0000310 when UAR0A0 is used.

3. Address values vary depending on the product.

	Internal RAM Size	Address Value
μPD70F3739 μPD70F3743	32 KB	0x3FF7000
μPD70F3740 μPD70F3744	40 KB	0x3FF5000
μPD70F3741 μPD70F3745	60 KB	0x3FF0000
μPD70F3742 μPD70F3746		

Remark The red value indicates the value set in the sample program.

- How to secure areas

In this sample program, the following shows examples for securing the area, using the CA850. Add the assemble source file and link directive code, as shown below.

(a) Assemble source (Add the following code as an assemble source file.)

```
-- Secures 2 KB space for monitor ROM section
.section "MonitorROM", const
.space 0x800, 0xff

-- Secures interrupt vector for debugging
.section "DBG0"
.space 4, 0xff

-- Secures interrupt vector for serial communication
-- Change the section name according to the serial communication mode used
-- In this sample program, the CSIB0 is used
.section "INTCB0R"
.space 4, 0xff

-- Secures 16-byte space for monitor RAM section
.section "MonitorRAM", bss
.lcomm monitorramsym, 16, 4
```

(b) Link directive (Add the following code to the link directive file.)

The following shows an example when the internal ROM has 1024 KB (end address is 0x00FFFFFF) and internal RAM has 60 KB (end address is 0x3FFEFF).

```
MROMSEG : !LOAD ?R V0x0ff800{
           MonitorROM      = $PROGBITS ?A MonitorROM;
};

MRAMSEG : !LOAD ?RW V0x03ffeff0{
           MonitorRAM      = $NOBITS ?AW MonitorRAM;
};
```

3.9 Describing interrupt handler by using #pragma directives

The format in which an interrupt handler is described does not differ from ordinary C functions, but the functions described in C must be recognized as an interrupt handler by the CA850. With the CA850, an interrupt handler is specified using the #pragma interrupt directive and __interrupt qualifier.

- When specifying interrupt handler

```
#pragma interrupt Interrupt-request-name Function-name Allocation-method

__interrupt Function-definition, or Function-declaration
```

- Program example

```
#pragma interrupt INTP0 f_int_intp0          /* In the case of external interrupt pin input
                                           edge detection
                                           □Interrupt source□□INTP0
                                           □functions□ □f_int_intp0      */
:

__interrupt
void f_int_intp0( void )                   /* The definition of the function " f_int_intp0" */
{
```

And CA850 can specify the following #pragma directives.

- Description with assembler instruction

```
#pragma asm
      assembler instruction
#pragma endasm
```

Assembler directives can be described in a C language source program.

- Peripheral I/O register name validation specification

```
#pragma ioreg
```

The peripheral I/O registers of a device are accessed by using peripheral function register names. This specification can use peripheral function register name as variable in C language source.

See the following user's manuals for details of how to set the items.

- CA850 Ver.3.20 C Compiler Package for C Language

CHAPTER 4 SETTING REGISTERS

This chapter describes how to set the interrupt pins and pins for lighting LED, as well as interrupt servicing.

For other initial settings, see the **V850ES/Jx3 Sample Program (Initial Settings) LED Lighting Switch Control**

Application Note.

Among the peripheral functions that are stopped after reset is released, those that are not used in this sample program are not set.

For how to set registers, see each product user's manual.

- V850ES/JJ3 32-bit Single-Chip Microcontroller
Hardware User's Manual
- V850ES/JG3 32-bit Single-Chip Microcontroller
Hardware User's Manual

See the following user's manuals for details of extended descriptions in C language.

- CA850 C Compiler Package C Language User's Manual

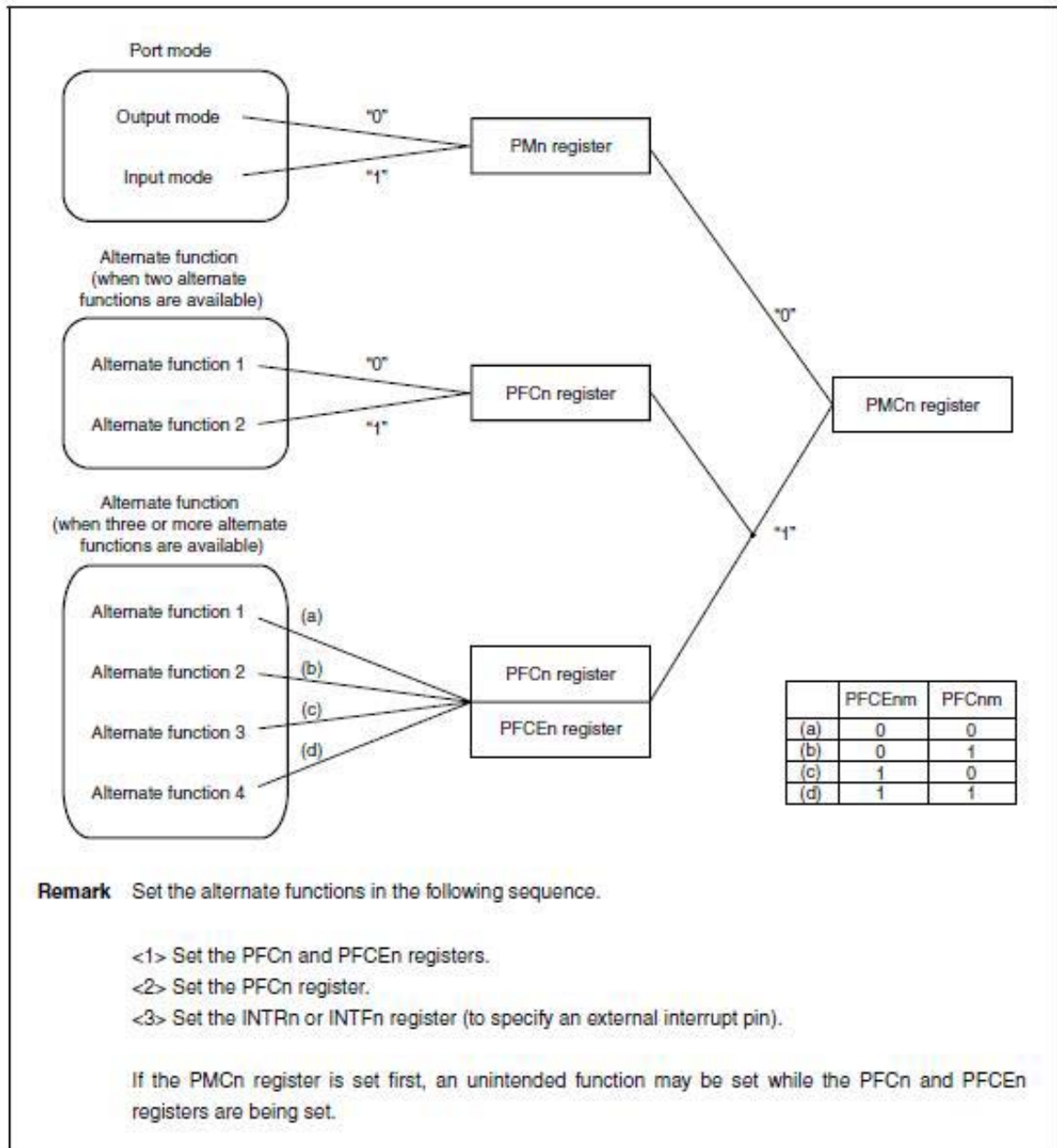
4.1 Setting Interrupt Pins and Pins for Lighting LEDs

In this sample program, the P03 pin is used as an external interrupt pin for SW1 input and interrupts are set to detect falling edges.

The PCM2 and PCM3 pins are set as output ports, because they are used as pins for lighting LEDs.

Set a port as illustrated below.

Figure 4-1. Setting of Each Register and Pin Function



4.1.1 Setting port 0 function control register (PFC0)

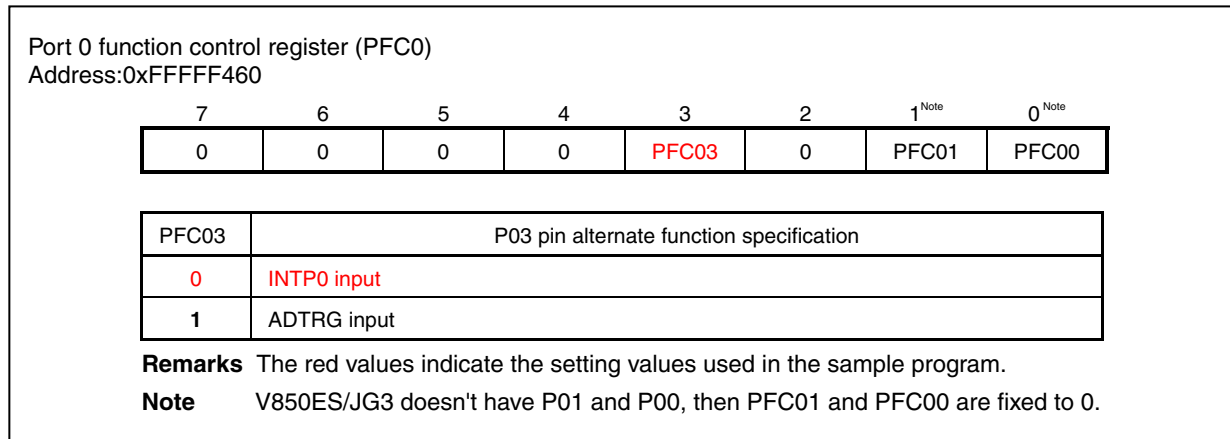
The PFC0 register specifies the alternate function of a port pin to be used if the pin has two alternate functions.

In this sample program, the alternate function of the P03 pin is set to INTPO input for using as an external interrupt pins.

The register can be read or written in 8-bit or 1-bit units.

Reset sets this register to 0x00.

Figure 4-2. PFC0 Register Format



The setting value of PFC0 is 0x00.

• Program example

```
PFC0 = 0x00;           /* Sets the P03 pin as INTPO input. */
```

4.1.2 Setting port 0 mode control register (PMC0)

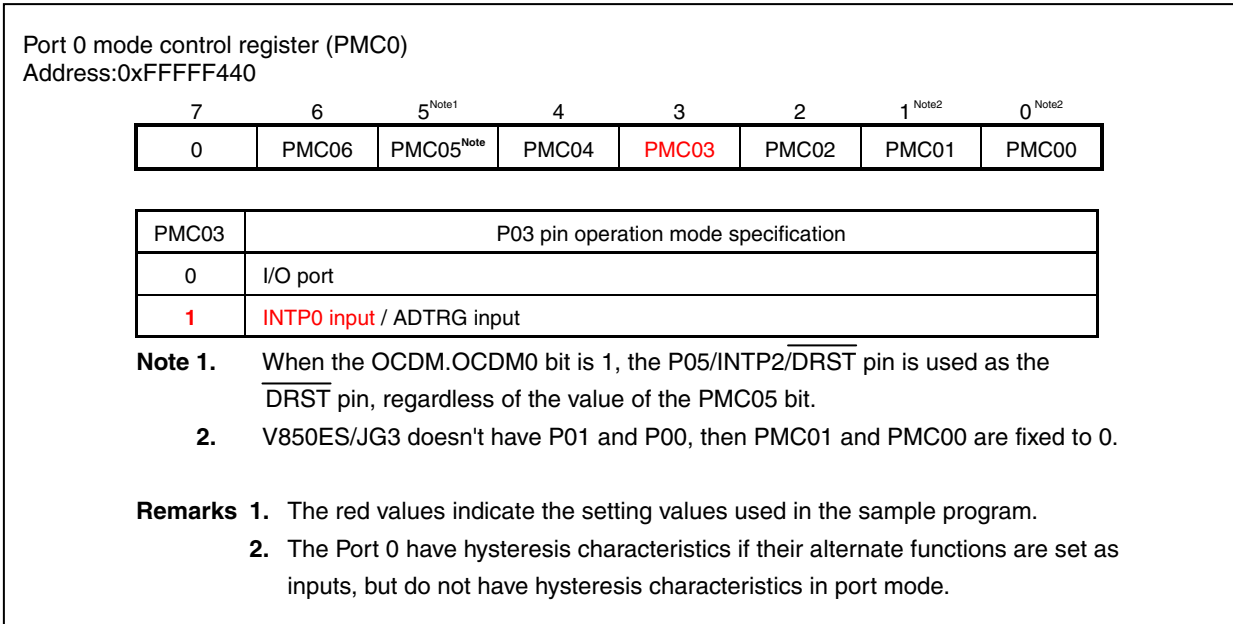
The PMC0 register can be used to specify the operation mode of the Port 0.

In this sample program, the operation mode of the P03 pin is set to INTP0 (external interrupt) input.

This register can be read or written in 8-bit or 1-bit units.

Reset sets this register to 0x00.

Figure 4-3. PMC0 Register Format



The setting value of PMC0 is 0x08.

• Program example

```
PMC0 = 0x08;          /* Sets the input of the INTP0 pin. */
```

4.1.3 Setting external interrupt falling and rising edge specification register 0 (INTF0, INTR0)

The INTF0 and INTR0 registers are 8-bit registers that are used to specify the detection of the NMI pin by using bit 2 and the detection of the rising and falling edges of external interrupt pins (INTP0 to INTP3) by using bits 3 to 6.

In this sample program, falling edges are set to be detected.

These registers can be read or written in 8-bit or 1-bit units.

Reset sets these registers to 0x00.

Figure 4-4. INTF0 Register Format

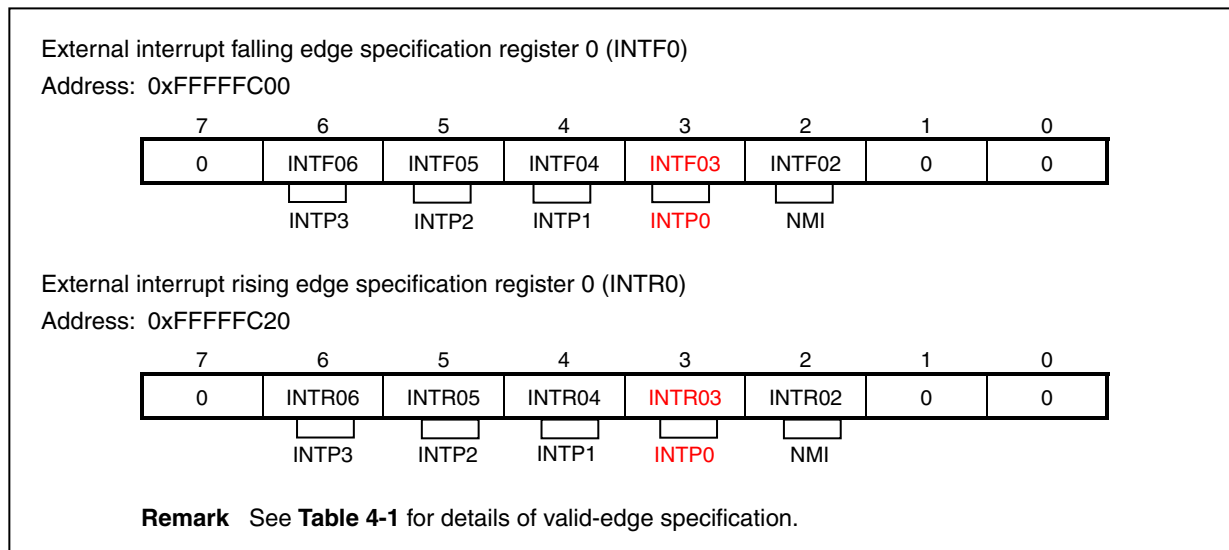


Table 4-1. Valid-Edge Specification

INTF03	INTR03	Valid-Edge (Falling) Specification
0	0	No edge detection
0	1	Detects the rising edge.
1	0	Detects the falling edge.
1	1	Detects both edges.

Remark The red values indicate the setting values used in the sample program.

The setting values of INTF0 and INTR0 are 0x08 and 0x00, respectively.

• Program example

```
INTF0 = 0x08;           /* Sets the input of the INTP0 pin. */
INTR0 = 0x00;
```


4.1.4 Setting interrupt control register (PIC0)

The PIC0 register is assigned for each interrupt request signal (maskable interrupt) and is used to set the control conditions for each interrupt.

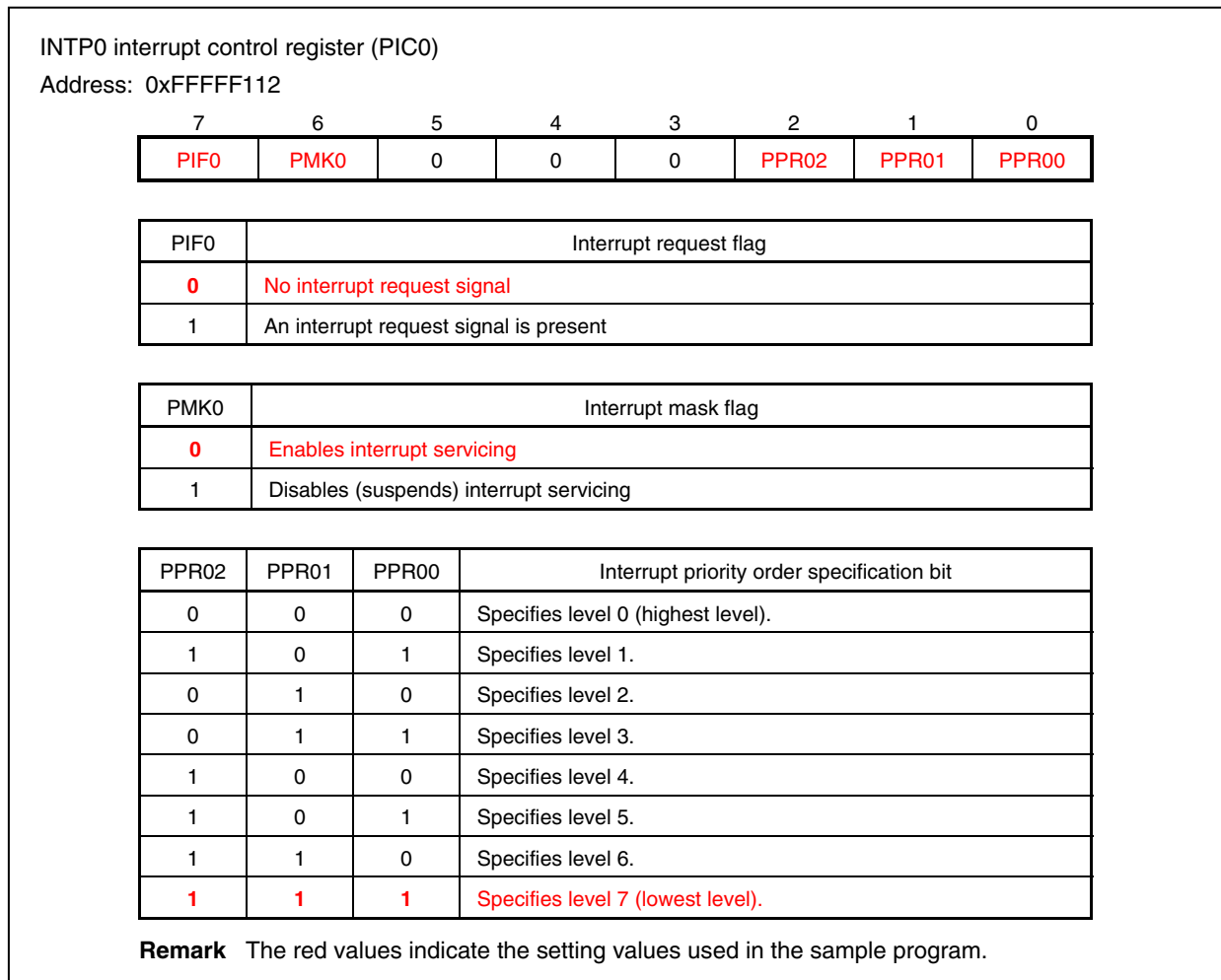
In this sample program, INTPO is set to enabled interrupt at the lowest priority level.

This register can be read or written in 8-bit or 1-bit units.

Reset sets this register to 0x47.

Caution Disable interrupts (DI) before reading the PIC0.PIF0 bit. If the PIC0.PIF0 bit is read while interrupts are enabled (EI), the correct value may not be read if acknowledging an interrupt and reading the bit conflict.

Figure 4-5. PIC0 Register Format



[Column] Interrupt request flag PIF0
Interrupt request flag PIF0 of the PIC0 register is set to 1 when an interrupt source is generated and automatically reset by hardware when an interrupt request signal is acknowledged.

The setting value of PIC0 is 0x07.

- Program example

```
PIC0 = 0x07;          /* Sets the priority level of INTP0 to level 7 and unmask  
                      INTP0. */
```

4.1.5 Setting port CM

Port CM is a port whose I/O can be controlled in 1-bit units.

In this sample program, the PCM2 and PCM3 pins are set to operate as output ports after the output to LED1 or LED2 is set not to light LED1 or LED2.

This register can be read or written in 8-bit or 1-bit units.

Reset sets the PCM register to 0x00, the PMCM register to 0xFF and the PMCCM register to 0x00.

Figure 4-6. PCM Register Format

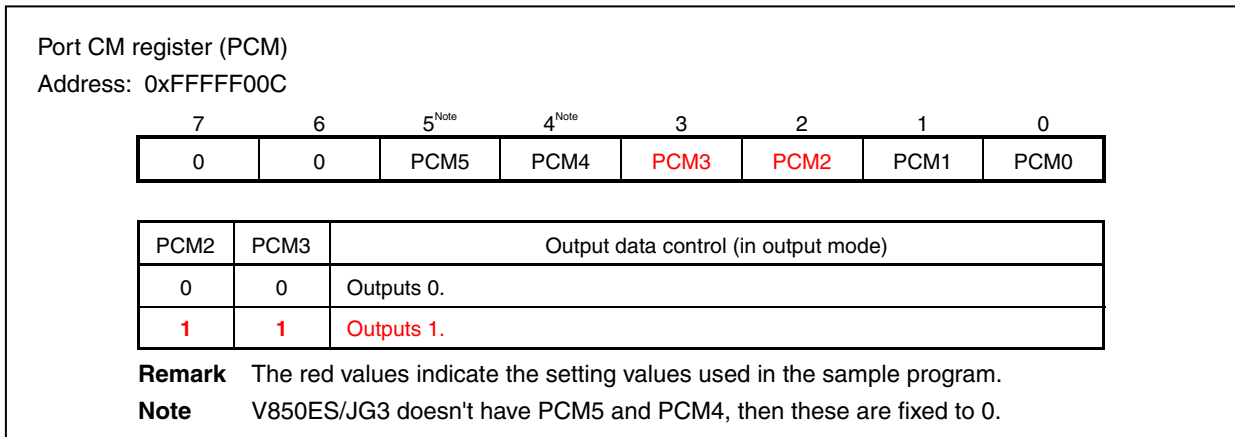
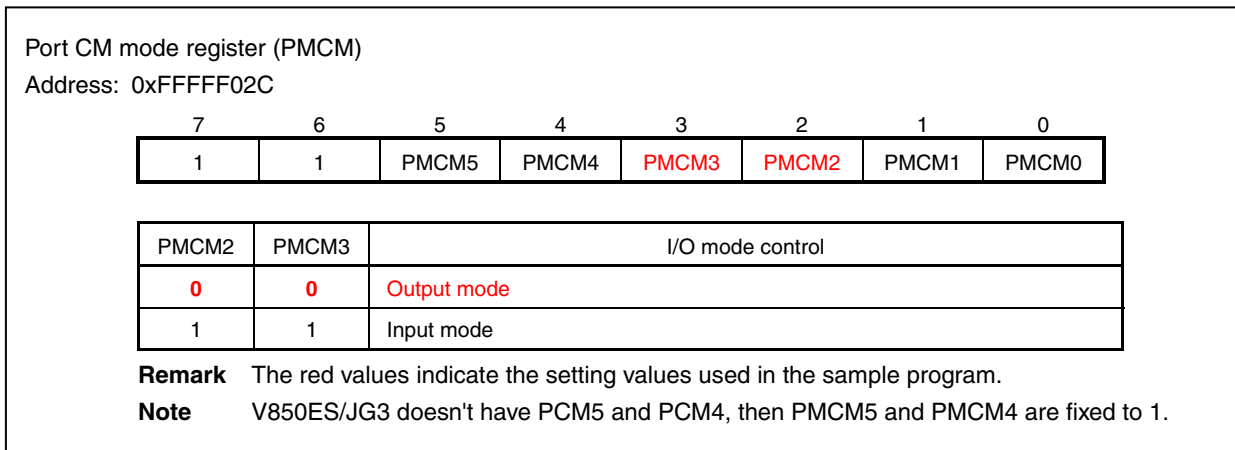
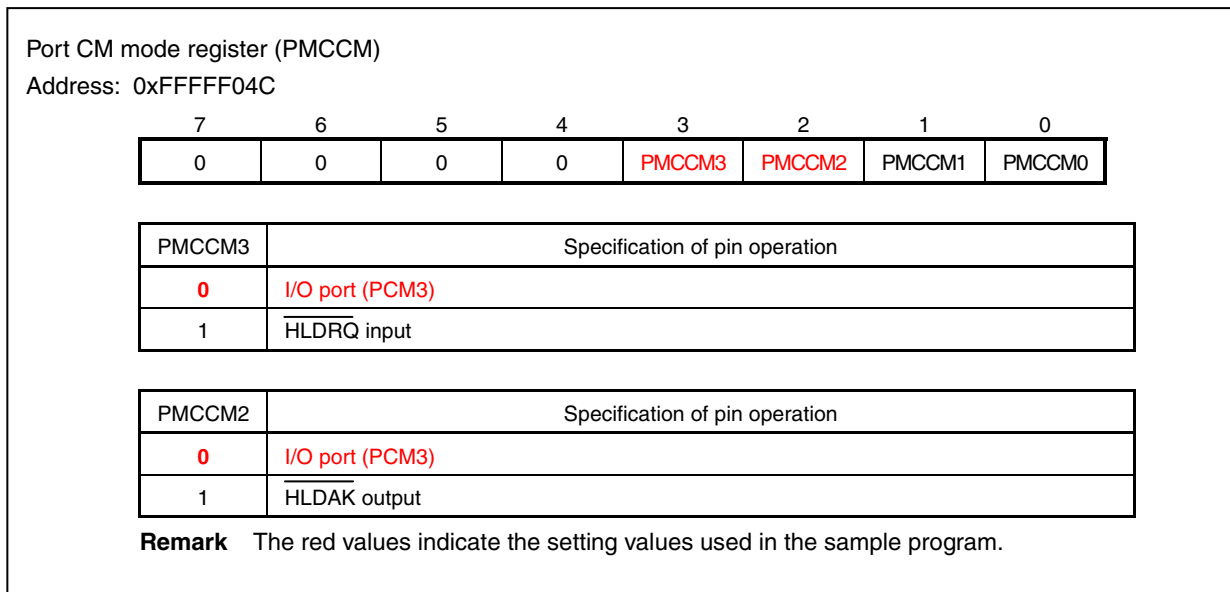


Figure 4-7. PMCM Register Format



[Column] Output port control
 Generally, to initialize a port as an output port, set the output latch before setting the port mode (I/O) (set Pn, then PMn).
 If the port mode is set first, the value set to the output latch at that time will be output from the pin and unintended pin output may be momentarily performed.

Figure 4-8. PMCCM Register Format



In this sample program, the output latches of PCM2 and PCM3 are preset to high-level output in the initial settings and then PCM2 and PCM3 are operated as output ports, in order to use PCM2 and PCM3 as output ports for lighting the LEDs.

The LEDs are lit when low level is output from PCM2 and PCM3 (see **2.1 Circuit Diagram**).

The setting values of PCM, PMCM and PMCCM are 0x0C, 0xC0 and 0x00, respectively.

- Program example

```

PCM = 0x0C;           /* Sets values to turn off lighting to LED1 and LED2. */
PMCM = 0xC0;         /* Sets the PCM pin to output. */
PMCCM = 0x00;        /* Sets the PMCCM to I/O port. */

```

4.2 Interrupt Servicing

In this sample program, chattering is eliminated, the lighting pattern is updated, and interrupt requests are cleared in interrupt servicing.

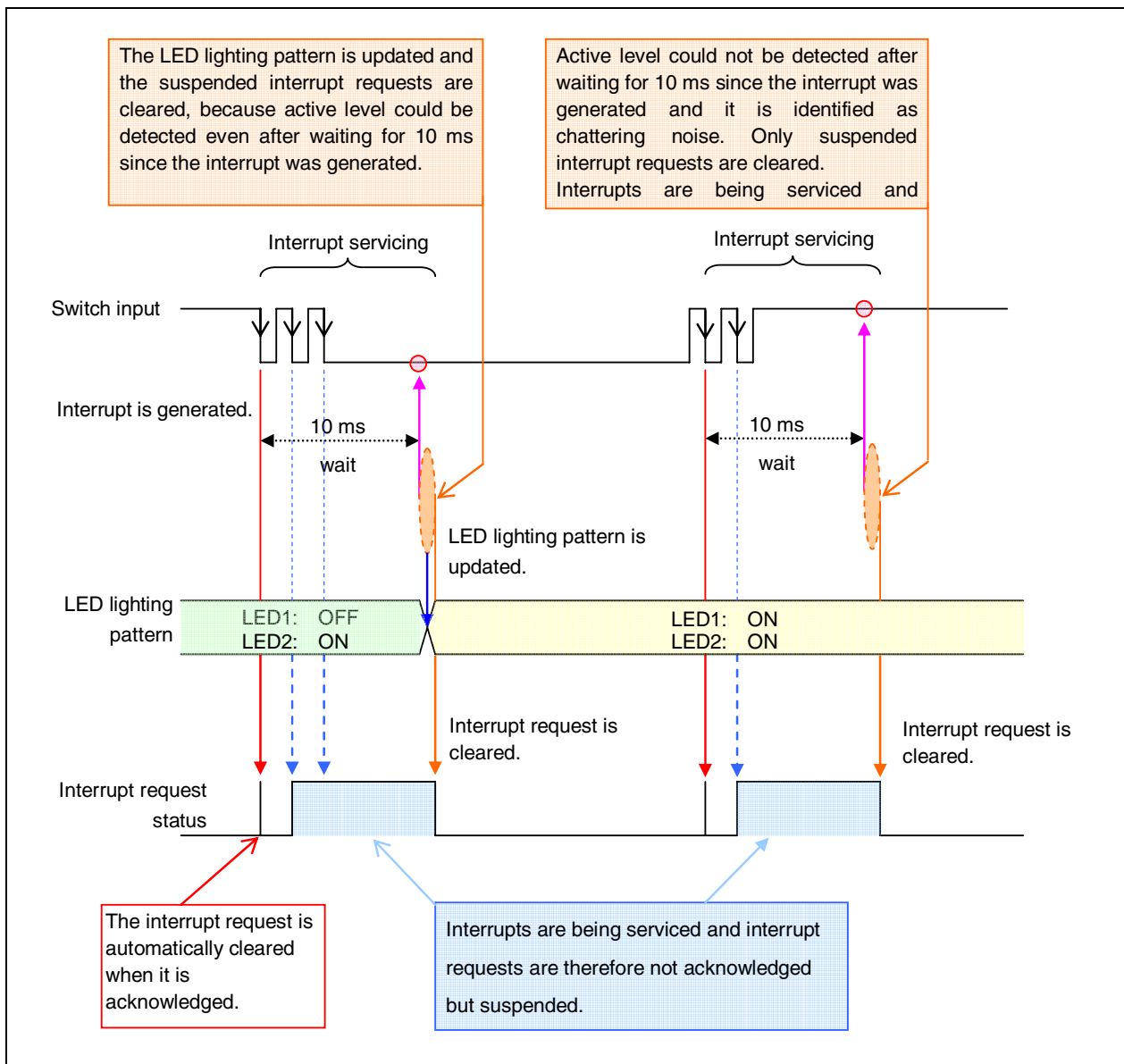
If an interrupt is generated, a 10 ms wait will be performed in interrupt servicing. If the switch input status is low level (active level) after the wait, switch input is assumed to be performed, and if the switch input status is high level (inactive level), it is identified as chattering noise and switch input is assumed not to be performed.

If switch input is performed, the current LED lighting pattern will be updated to the next LED lighting pattern.

See **Table 1-1 LED Lighting Patterns** for details of updating the LED lighting pattern.

After the LED lighting pattern is updated, the suspended interrupt requests that may have been generated due to chattering noise are cleared.

A timing chart of interrupt servicing triggered by switch input is shown below.



CHAPTER 5 RELATED DOCUMENTS

Document	document number
V850ES/JJ3 Hardware User's Manual	U18376E
V850ES/JG3 Hardware User's Manual	U18708E
V850ES 32-Bit Microprocessor Core for Architecture	U15943E
PM+ Ver. 6.30 User's Manual	U18416E
CA850 Ver. 3.20 C Compiler Package Operation	U18512E
CA850 Ver. 3.20 C Compiler Package C Language	U18513E
CA850 Ver.3.20 C Compiler Package for Link Directives	U18515E
QB-MINI2 User's Manual	U18371E
ID850QB Ver.3.40 Integrated Debugger for Operation	U18604E

Document Search URL <http://www.necel.com/search/en/index.html#doc>

APPENDIX A PROGRAM LIST

The V850ES/Jx3 microcontroller source program is shown below as a program list example.

```
● minicube2.s
#-----
#
#   NEC Electronics      V850ES/Jx3 series
#
#-----
#   V850ES/JJ3 JG3 sample program
#-----
#   Interrupts
#-----
#[History]
#   2009.9.--   Released
#-----
#[Overview]
#   This sample program secures the resources required when using MINICUBE2.
#   (Example of using MINICUBE2 via CSIB0)
#-----

-- Securing a 2 KB space as the monitor ROM section
.section "MonitorROM", const
.space 0x800, 0xff

-- Securing an interrupt vector for debugging
.section "DBG0"
.space 4, 0xff

-- Securing a reception interrupt vector for serial communication
.section "INTCB0R"
.space 4, 0xff

-- Securing a 16-byte space as the monitor RAM section
.section "MonitorRAM", bss
.lcomm monitorrmsym, 16, 4
```

Set the section name according to use serial interface.

when interface is CSIB3 : INTCB3R
when interface is UARTA0 : INTUA0R

```

● AppNote_LED.dir
# Sample link directive file (not use RTOS/use internal memory only)
#
# Copyright (C) NEC Electronics Corporation 2002
# All rights reserved by NEC Electronics Corporation.
#
# This is a sample file.
# NEC Electronics assumes no responsibility for any losses incurred by customers or
# third parties arising from the use of this file.
#
# Generated      : PM+ V6.31 [ 9 Jul 2007]
# Sample Version : E1.00b [12 Jun 2002]
# Device         : uPD70F3746 (C:\Program Files\NEC Electronics Tools\DEV\DF3746.800)
# Internal RAM   : 0x3ff0000 - 0x3ffefff
#
# NOTICE:
#     Allocation of SCONST, CONST and TEXT depends on the user program.
#
#     If interrupt handler(s) are specified in the user program then
#     the interrupt handler(s) are allocated from address 0 and
#     SCONST, CONST and TEXT are allocated after the interrupt handler(s).

SCONST : !LOAD ?R {
        .sconst      = $PROGBITS      ?A .sconst;
};

CONST  : !LOAD ?R {
        .const       = $PROGBITS      ?A .const;
};

TEXT   : !LOAD ?RX {
        .pro_epi_runtime = $PROGBITS   ?AX .pro_epi_runtime;
        .text          = $PROGBITS     ?AX .text;
};

### For MINICUBE2 ###
MROMSEG : !LOAD ?R V0x0ff800{
        MonitorROM    = $PROGBITS ?A MonitorROM;
};

SIDATA : !LOAD ?RW V0x3ff0000 {

```

Address values vary depending on the product internal ROM size.

This is an example of the product that's internal ROM size is 1024KB.

```

### For MINICUBE2 ###
MROMSEG : !LOAD ?R V0x0ff800{
        MonitorROM    = $PROGBITS ?A MonitorROM;
};

```

Difference from the default link directive file(additional code).

A reserved area for MINICUBE2 is secured.


```

.tidata.byte    = $PROGBITS    ?AW .tidata.byte;
.tibss.byte     = $NOBITS      ?AW .tibss.byte;
.tidata.word    = $PROGBITS    ?AW .tidata.word;
.tibss.word     = $NOBITS      ?AW .tibss.word;
.tidata         = $PROGBITS    ?AW .tidata;
.tibss          = $NOBITS      ?AW .tibss;
.sidata         = $PROGBITS    ?AW .sidata;
.sibss          = $NOBITS      ?AW .sibss;
};

```

```

DATA : !LOAD ?RW V0x3ff0100 {
    .data      = $PROGBITS    ?AW .data;
    .sdata     = $PROGBITS    ?AWG .sdata;
    .sbss      = $NOBITS      ?AWG .sbss;
    .bss       = $NOBITS      ?AW .bss;
};

```

```

### For MINICUBE2 ###
MRAMSEG : !LOAD ?RW V0x03ffeff0{
    MonitorRAM = $NOBITS ?AW MonitorRAM;
};

```

Difference from the default link directive file(additional code).

A reserved area for MINICUBE2 is secured.

```

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;

```

```

● main.c
/*-----*/
/*
/* NEC Electronics      V850ES/Jx3 series
/*
/*-----*/
/* V850ES/JJ3 sample program
/*-----*/
/* Interrupts
/*-----*/
/* [History]
/* 2009.09.-- Released
/*-----*/
/* [Overview]
/* This sample program presents an example of using the interrupt function.
/* An LED lighting pattern that accords with the number of switch inputs is displayed
   by detecting the falling edge of the switch input and generating an interrupt.
/* A chattering elimination time of 10 ms is provided for the switch inputs.
/*
/* Among the peripheral functions that are stopped after reset is released, those that
   are not used in this sample program are not set.
/*
/*
/* <Main setting contents>
/* • Using pragma directives to enable setting the interrupt handler and describing
   peripheral I/O register names
/* • Defining a wait adjustment value of 10 ms for chattering
/* • Performing prototype definitions
/* • Defining the LED lighting pattern table
/* • Setting a bus wait for on-chip peripheral I/O registers, stopping the watchdog
   timer 2, and setting the clock
/* • Initializing unused ports
/* • Initializing external interrupt ports (falling edge) and LED output ports
/* • ROMization
/* • Updating the LED lighting pattern in interrupt servicing
/* (Chattering elimination time during switch input: 10 ms)
/*

```

```

/* <Switch input and LED lighting>
/*
/* +-----+
/* |Number of times the switch is pressed | LED2 | LED1 |
/* | (P03/INTP0) | (PCM2) | (PCM3) |
/* |-----|
/* | 0 times | OFF | OFF |
/* | 1 time | OFF | ON |
/* | 2 times | ON | OFF |
/* | 3 times | ON | ON |
/* +-----+
/* *Inputs 0 to 3 are repeated from the fourth input.
/*
/*
/*[I/O port settings]
/*
/* • Input port : P03(INTP0)
/* • Output ports : PCM2, PCM3
/* • Unused ports : P00 to P02, P04 to P06, P10 and P11, P30 to P39,P40 to P42,
/* P50 to P55, P60 to P615, P70 to P715, P80 and P81, P90 to P915,
/* PCD0 to PCD3, PCM0 and PCM1, PCM4 and PCM5, PCS0 to PCS7,
/* PCT0 to PCT7, PDH0 to PDH7, PDL0 to PDL15
/* *Preset all unused ports as output ports (low-level output).
/*
/*-----*/

/*-----*/
/* pragma directives */
/*-----*/
#pragma ioreg /* Enables describing to peripheral I/O registers. */
#pragma interrupt INTP0 f_int_intp0 /* Specifies the interrupt handler. */

/*-----*/
/* Constant definitions */
/*-----*/
#define LIMIT_10ms_WAIT (0x9D58) /* Defines the constant for a 10 ms wait
adjustment. */

```

```

/*-----*/
/*  Prototype definitions  */
/*-----*/
        void main( void );           /* Main function          */
static   void f_init( void );        /* Initialization function */
static   void f_init_clk_bus_wdt2( void ); /* Clock bus WDT initialization */
static   void f_init_port_func( void ); /* Port/alternate-function initialization
                                         function          */

/*-----*/
/* Setting the LED pattern table */
/*-----*/
const unsigned char LED_TBL[] ={ 0x08, /* Specification of LED1 display pattern */
                                0x0C }; /* Specification of LED1 and LED2 display
                                         pattern */

/*****/
/*      Main module      */
/*****/
void main(void)
{
    extern unsigned int _S_romp;      /* Externally references ROMization
                                         symbols.          */

    f_init();                        /* Executes initialization.          */

    _rcopy( &_S_romp, -1 );          /* Executes ROMization processing.   */

    __EI();                          /* Enables interrupts.          */

    while(1);                        /* Main loop (infinite loop)        */

    return;
}

```



[Column] LED display changing pattern

When CSIB0 is used for MINICUBE2 debugging, the debug monitor program may use PCM.PCM0 which is used for LED control. In this sample program, the value of PCM is modified by bit-manipulation to avoid breaking the value of PCM0 at interrupt module f_int_intp0.
 (The data array LED_TBL stores the bit specification value for using bit manipulation)

```
PCM ^= LED_TBL[led_ptn_cnt]; /* Turn over the bits which are specified by LED_TBL */
```

```

/*-----*/
/* Initialization module */
/*-----*/
static void f_init( void )
{
    f_init_clk_bus_wdt2(); /* Sets a bus wait for on-chip peripheral I/O registers,
                           stops WDT2, and sets the clock. */

    f_init_port_func(); /* Sets the port/alternate function. */

    return;
}

```

```

/*-----*/
/* Initializing clock bus WDT2 */
/*-----*/
static void f_init_clk_bus_wdt2( void )
{
    VSWC = 0x11; /* Sets a bus wait for on-chip peripheral I/O
                 registers. */

    /* Specifies normal operation mode for OCDM. */

```

```

#pragma asm
    push r10
    mov 0x00, r10
    st.b r10, PRCMD
    st.b r10, OCDM
    pop r10
#pragma endasm

```

Caution must be exercised because access to a special register must be described in assembly language.

```

    RCM = 0x01; /* Stops the internal oscillator. */
    WDTM2 = 0x00; /* Stops the watchdog timer 2. */

    PLLON = 0; /* stops the PLL */
    /* PLL multiplication is set to 8 */

```

```

#pragma asm
    push r10
    mov 0x0B, r10
    st.b r10, PRCMD
    st.b r10, CKC
    pop r10
#pragma endasm

```

```

    PLLON = 1; /* Enable PLL operation */

    while( LOCK ); /* Wait for PLL stabled */
    SELPLL = 1; /* Set PLL mode */

```

```

/* Setting the PCC register */
/* Sets to not divide the clock. */

#pragma asm
  push r10
  mov 0x80, r10
  st.b r10, PRCMD
  st.b r10, PCC
  pop r10
#pragma endasm

return;
}

/*-----*/
/* Setting the port/alternate function */
/*-----*/
static void f_init_port_func( void )
{
  P0   = 0x00;          /* Sets P00 to P06 to output low level.      */
  PM0  = 0x88;
  PFC0 = 0x00;          /* Sets the P03 pin as INTPO input */
  PMC0 = 0x08;

  P1   = 0x00;          /* Sets P10 and P11 to output low level.     */
  PM1  = 0xFC;

  P3   = 0x0000;        /* Sets P30 to P39 to output low level.     */
  PM3  = 0xFC00;
  PMC3 = 0x0000;

  P4   = 0x00;          /* Sets P40 to P42 to output low level.     */
  PM4  = 0xF8;
#if(0) /* To use P4 as CSIB0 when using MINICUBE2,          */
  /* P4 is not initialized as an unused pin (QB-V850ESJJ3-TB) */
  PMC4 = 0x00;
#endif

  P5   = 0x00;          /* Sets P50 to P55 to output low level.     */
  PM5  = 0xC0;
  PMC5 = 0x00;

  P6   = 0x0000;        /* Sets P60 to P615 to output low level.     */
  PM6  = 0x0000;
  PMC6 = 0x0000;
}

```

With V850ES/JG3, the setting value is 0x8B

With V850ES/JG3, these are not set because the registers do not exist.

```

P7H = 0x00; /* Sets P70 to P711 to output low level. */
P7L = 0x00;
PM7H = 0x00; /* With V850ES/JG3, the setting value is 0xF0. */
PM7L = 0x00;

P8 = 0x00; /* Sets P80 and P81 to output low level. */
PM8 = 0xFC; /* With V850ES/JG3, these are not set because the registers do not exist. */
PMC8 = 00;

P9 = 0x0000; /* Sets P90 to P915 to output low level. */
PM9 = 0x0000;
PMC9 = 0x0000;

PCD = 0x00; /* Sets PCD0 to PCD3 to output low level. */
PMCD = 0xF0; /* With V850ES/JG3, these are not set because the registers do not exist. */

PCM = 0x0C; /* Sets PCM0,PCM1, PCM4 and PCM5 to output low level and values to turn off the LEDs to PCM2 and PCM3. */

PMCM = 0xC0; /* With V850ES/JG3, the setting value is 0xF0. */
PMCCM = 0x00;

PCT = 0x00; /* Sets PCT0 to PCT7 to output low level. */
PMCT = 0x00; /* With V850ES/JG3, the setting value is 0xAC. */
PMCCT = 0x00;

PDH = 0x00; /* Sets PDH0 to PDH7 to output low level. */
PMDH = 0x00; /* With V850ES/JG3, the setting value is 0xC0. */
PMCDH = 0x00;

PDL = 0x0000; /* Sets PDL0 to PDL15 to output low level. */
PMDL = 0x0000;
PMCDL = 0x0000;

/* Setting the interrupt function */
INTF0 = 0x08; /* Specifies the falling edge of INTP0. */
INTR0 = 0x00; /* ↓ */
PIC0 = 0x07; /* Sets the priority of INTP0 to level 7 and unmask INTP0. */

return;
}

```

```
/*
*****
/*      Interrupt module      */
*****
__interrupt
void f_int_intp0( void )
{
    static unsigned int led_ptn_cnt = 0;
    unsigned int loop_wait;

    /* 10 ms wait for chattering */
    for( loop_wait = 0 ; loop_wait < LIMIT_10ms_WAIT ; loop_wait++ )
    {
        __nop();
    }

    if( ( P0 & 0x08 ) == 0x00 )    /* Recognizes SW1 even after chattering is
                                   eliminated.*/
    {
        PCM ^= LED_TBL[led_ptn_cnt]; /* Change the updated lighting pattern.      */
        led_ptn_cnt ^= 0x01;         /* Turn over the lighting change pattern. */
    }

    PIF0 = 0;                      /* FailSafe: Clears multiple requests.      */

    return;
}
```