RX Family

Using the Exception Vector Table and Software Configurable Interrupts

Abstract

This document describes using the exception vector table and software configurable interrupts for the RX Family MCU.

The Exception Vector Table section explains how to change addresses placed on the exception vector table. The Software Configurable Interrupts section describes the method to generate software configurable interrupts by assigning interrupt factors to interrupt vector numbers.

In addition, for locations where there is no specific description, about the RX65N group is explained. Refer to “6 Porting Sample Codes to Other RX Family” and User's Manual: Hardware of each microcomputer when another microcomputer is used.

Target Device

- RX Family MCU loaded with the exception vector table and software configurable interrupts

Contents

1. Exception Vector Table and Software Configurable Interrupts

1.1 Exception Vector Table

In the exception vector table, the individual vectors for exception events are allocated to the 124-byte area where the value indicated by the exception table register (EXTB) is used as the starting address. The addresses where the exception vector table is allocated can be changed by setting an arbitrary address in the EXTB register.

In this document, the EXTB register value is changed from 0xFFFFFF80 to 0x0000FF80. Figure 1.1 shows the Exception Vector Table of RX65N Group and RX651 Group.

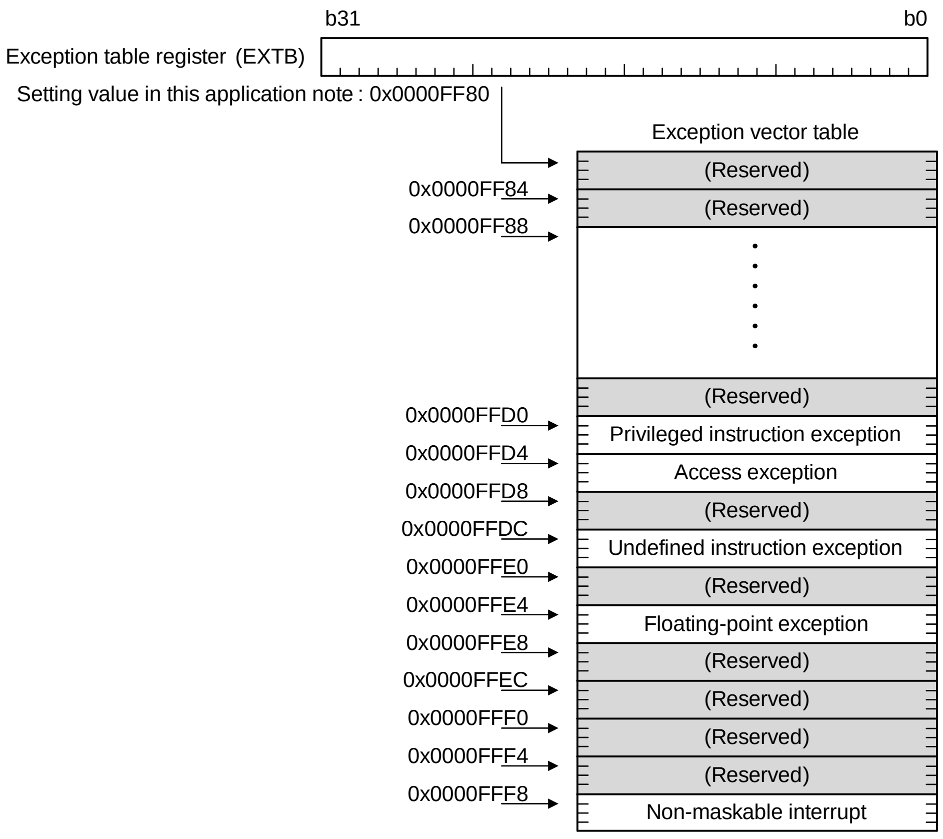


Figure 1.1 Exception Vector Table of RX65N Group and RX651 Group

1.2 Software Configurable Interrupts

An interrupt source selected from multiple peripheral sources can be assigned to each interrupt vector number from 128 to 255 as a software configurable interrupt. The software configurable interrupts are divided into software configurable interrupt B and software configurable interrupt A depending on the peripheral operating clock.

Software configurable interrupts are assigned by setting the interrupt source numbers in registers SLIBXRn, SLIBRn, and SLIARn.

Registers SLIBXRn and SLIBRn are used to assign interrupt vector numbers to interrupt sources for software configurable interrupt B. The SLIBXRn register is for interrupt vector numbers 128 to 143. The SLIBRn register is for interrupt vector numbers 144 to 207.

The SLIARn register is used to assign interrupt vector numbers from 208 to 255 to interrupt sources for software configurable interrupt A.

In this application note, the SLIAR208 register is set to interrupt source number 1 (interrupt vector number 208 is assigned to TGIA0).

Figure 1.2 shows the Software Configurable Interrupt Assignment for RX65N Group and RX651 Group.

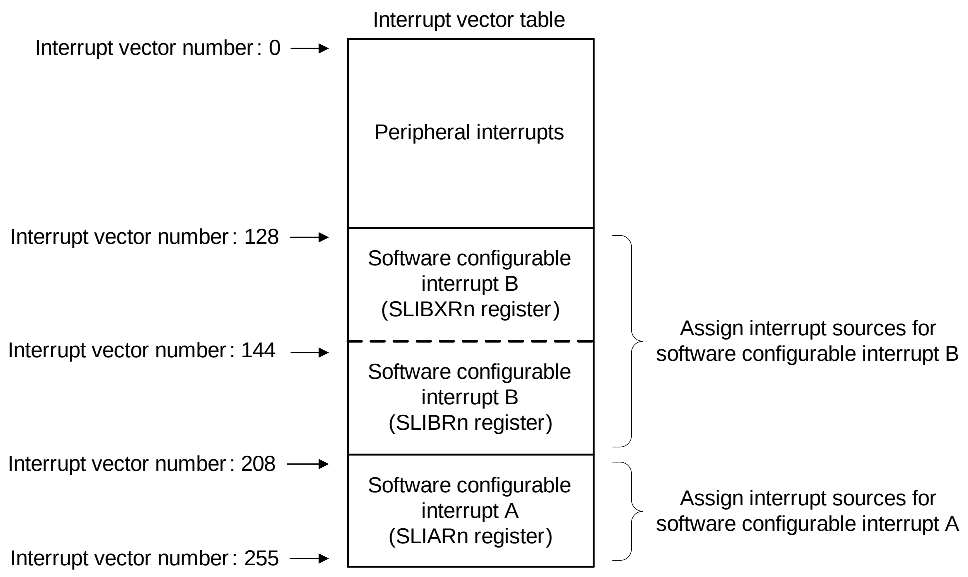

Interrupt vector table
Interrupt vector number : 0
Peripheral interrupts
Interrupt vector number : 128
Software configurable interrupt B (SLIBXRn register)
Interrupt vector number : 144
Software configurable interrupt B (SLIBRn register)
Interrupt vector number : 208
Software configurable interrupt A (SLIARn register)
Interrupt vector number : 255
Assign interrupt sources for software configurable interrupt B
Assign interrupt sources for software configurable interrupt A

Figure 1.2 Software Configurable Interrupt Assignment for RX65N Group and RX651 Group

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

Item	**Contents**
MCU used	R5F565NEDDFC (RX65N group)
Operating frequencies	Main clock: 24 MHz PLL: 240 MHz (main clock divided by 1 and multiplied by 10) System clock (ICLK): 120 MHz (PLL divided by 2) Peripheral module clock A (PCLKA): 120 MHz (PLL divided by 2)
Operating voltage	3.3 V
Integrated development environment	Renesas Electronics Corporation e^2 studio Version: 7.3.0
C compiler	Renesas Electronics Corporation C/C++ Compiler Package for RX Family V3.01.00
	Compile options The rom option is used. -rom=RAM_EXFUNC=RAM_EXFUNC_COPY -rom=RAM_EXVECT=RAM_EXVECT_COPY
iodefine.h version	2.0a
Endian	Little endian
Operating mode	Single-chip mode
Processor modes	User mode is used for moving the exception vector table. Supervisor mode is used for software configurable interrupts.
Sample code version	Version1.10
Board used	Renesas Starter Kit+ for RX65N-2MB (part number: RTK50565N2SxxxxxBE)

3. Reference Application Note

For additional information associated with this document, refer to the following application note.

- RX65N Group, RX651 Group Initial Settings (R01AN3034)

4. Hardware Configuration

4.1 Pins Used

Table 4.1 lists the Pins Used and Their Functions and Figure 4.1 shows the Connection Diagram.

Table 4.1 Pins Used and Their Functions

Pin Name	I/O	Function
P73	Output	LED0 output (exception handling: privileged instruction exception)
PG5	Output	LED3 output (software configurable interrupt: input capture/compare match interrupt)

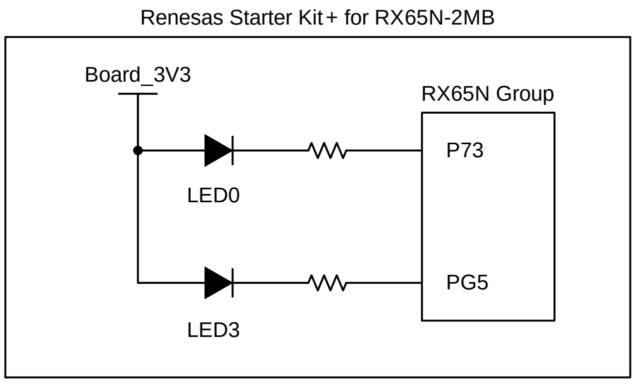


Figure 4.1 Connection Diagram

5. Software

5.1 Operation Overview of the Exception Vector Table

In this document, one program is used to introduce the following two processes: moving the exception vector table and the software configurable interrupt handling. Set the SEL_INT constant in the r_int_config.h file to switch the processes.

When moving the exception vector table, set EXCEP_HANDL in the SEL_INT constant. In the sample code, the default is EXCEP_HANDL.

5.1.1 Moving the Exception Vector table

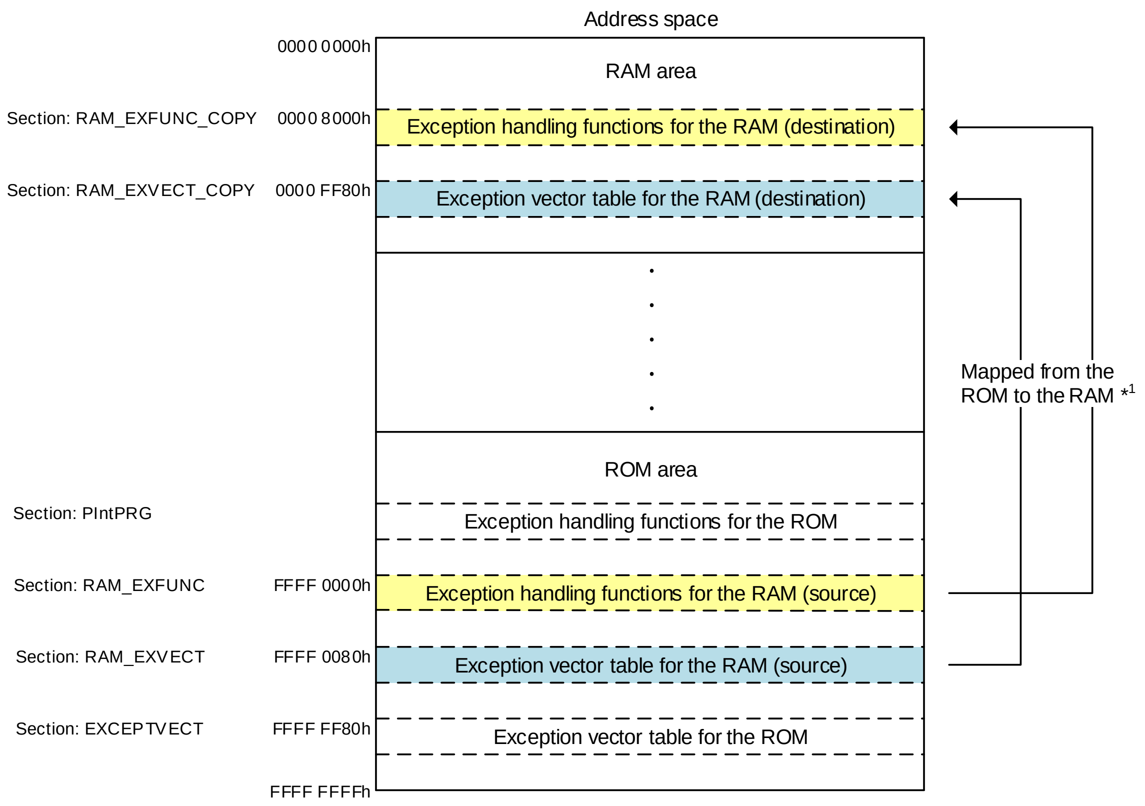
When moving the exception vector table, the following processing is performed to execute exception handling on the RAM.

(1) Moves the exception handling function to the RAM area (0x00008000).
(2) Moves the content of the exception vector table to the RAM area (0x0000FF80).
(3) Changes the EXTB value from the ROM area (default value: 0xFFFFFF80) to the RAM area (0x0000FF80).
(4) Switches the processor mode from supervisor mode to user mode, and executes the privileged instruction to generate an interrupt (privileged instruction interrupt)*1.
(5) Lights up LED0 by interrupt handling.

Note 1. The processor interrupt priority level is set to 2 but since the privileged instruction can only be executed in supervisor mode, there is no change in the value.

Using this function allows exception processing to be executed on the RAM even when it is not possible to access to the ROM during flash rewrite.

Figure 5.1 shows Address Space in this application note.


Address space
0000 0000h
RAM area
Section: RAM_EXFUNC_COPY
0000 8000h
Exception handling functions for the RAM (destination)
Section: RAM_EXVECT_COPY
0000 FF80h
Exception vector table for the RAM (destination)
Mapped from the ROM to the RAM *1
ROM area
Section: PIntPRG
Exception handling functions for the ROM
Section: RAM_EXFUNC
FFFF 0000h
Exception handling functions for the RAM (source)
Section: RAM_EXVECT
FFFF 0080h
Exception vector table for the RAM (source)
Section: EXCEPTVECT
FFFF FF80h
Exception vector table for the ROM
FFFF FFFFh

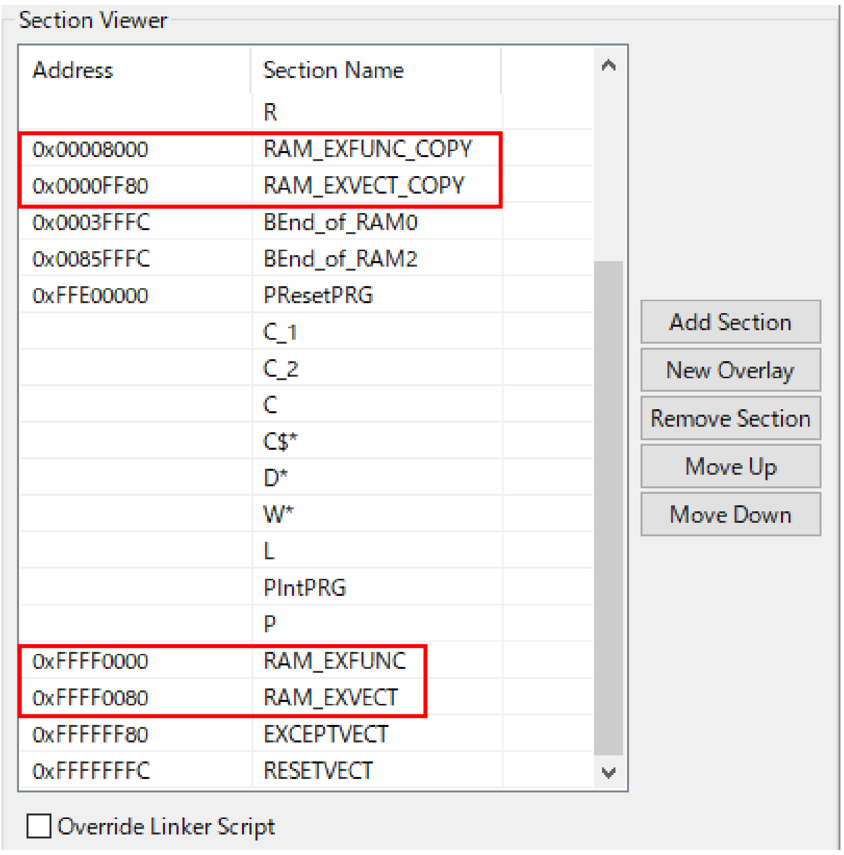
Note 1. Refer to 5.1.2 Moving the Exception Vector Table to the RAM Area for details on the procedure to move the RAM exception vector table and RAM exception handling functions from the ROM to the RAM.

Figure 5.1 Address Space

5.1.2 Moving the Exception Vector Table to the RAM Area

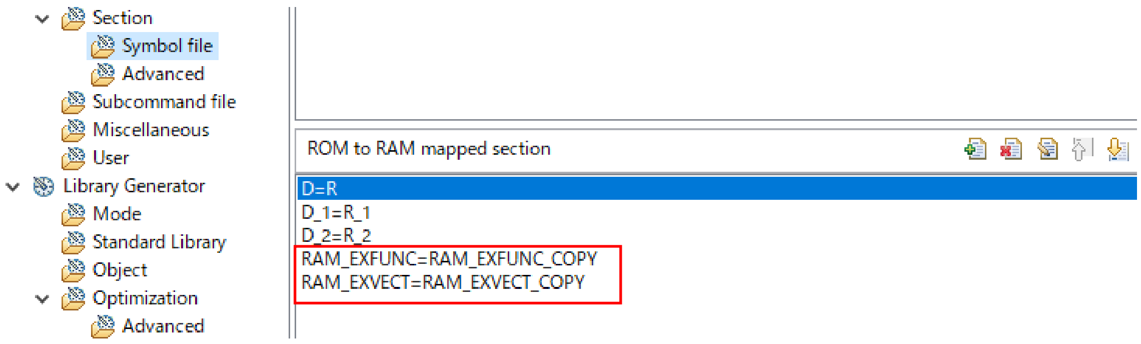
This section describes the procedure to prepare the exception vector table and exception handling functions for the RAM and change the addresses for allocating the exception vector table from the ROM area (default: 0xFFFFFF80) to the RAM area (0x0000FF80).

1) Set sections
   1-1) Right-click the project and select “Properties”.
   1-2) Select “C/C++ Build > Setting > Linker > Section” from the properties.
   1-3) Click “…” displayed in the upper right corner.
   1-4) Set sections for the exception vector table and exception handling functions in the RAM area and ROM area
   1-5) Click “OK”.


Section Viewer

Address	Section Name
	R
0x00008000	RAM_EXFUNC_COPY
0x0000FF80	RAM_EXVECT_COPY
0x0003FFFC	BEnd_of_RAM0
0x0085FFFC	BEnd_of_RAM2
0xFFE00000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*
	W*
	L
	PIntPRG
	P
0xFFFF0000	RAM_EXFUNC
0xFFFF0080	RAM_EXVECT
0xFFFFFF80	EXCEPTVECT
0xFFFFFFFC	RESETVECT

   1-6) Select “C/C++ Build > Setting > Symbol file” from the properties.
   1-7) Use the rom option to relocate the define symbol in the ROM section to the address in the RAM section.
   1-8) Click “Apply and Close”

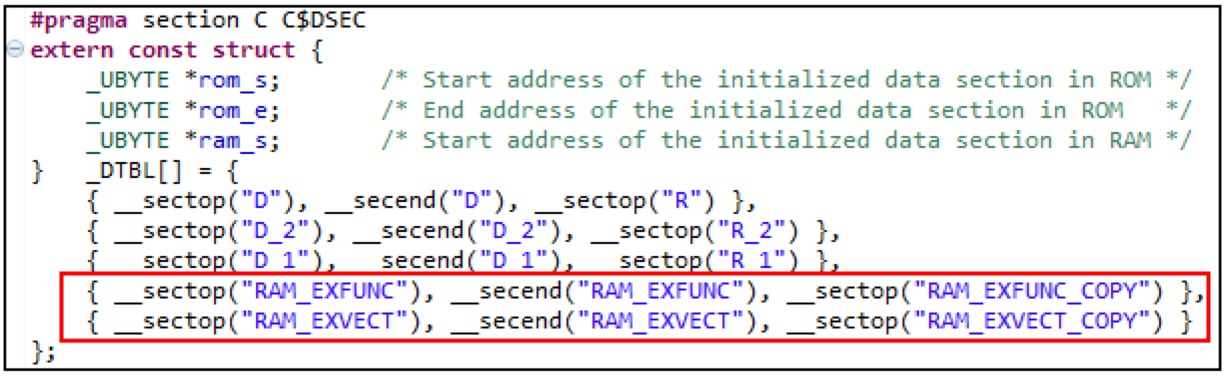


ROM to RAM mapped section

```
D=R
D_1=R_1
D_2=R_2
RAM_EXFUNC=RAM_EXFUNC_COPY
RAM_EXVECT=RAM_EXVECT_COPY
```

2) Allocate the exception vector table and exception handling functions for the RAM to the ROM
   2-1) Declare #pragma section C RAM_EXVECT in the ram_except_vector_table.c file.
   Allocate the RAM exception vector table to the RAM_EXVECT section.
   2-2) Declare #pragma section P RAM_EXFUNC in the ram_except_handlers.c file.
   Allocate the RAM exception handling functions to the RAM_ EXFUNC section.

3) Initialize the RAM area section
   3-1) Add the section to be initialized to the section initialization table (DTBL) of the dbsct.c file.
   The section described in the initialization table (DTBL) is initialized by calling the initialization routine (_INITSCT). In this application note, the section declared in step (2) is initialized using this step. RAM_EXFUNC is copied to RAM_EXFUNC_COPY, and RAM_EXVECT is copied to RAM_EXVECT_COPY.
   The initialization table (DTBL) of the RX65N group and RX651 group are shown below.



```
#pragma section C C$DSEC
extern const struct {
    _UBYTE *rom_s;       /* Start address of the initialized data section in ROM */
    _UBYTE *rom_e;       /* End address of the initialized data section in ROM   */
    _UBYTE *ram_s;       /* Start address of the initialized data section in RAM */
}   _DTBL[] = {
    { __sectop("D"), __secend("D"), __sectop("R") },
    { __sectop("D_2"), __secend("D_2"), __sectop("R_2") },
    { __sectop("D_1"), __secend("D_1"), __sectop("R_1") },
    { __sectop("RAM_EXFUNC"), __secend("RAM_EXFUNC"), __sectop("RAM_EXFUNC_COPY") },
    { __sectop("RAM_EXVECT"), __secend("RAM_EXVECT"), __sectop("RAM_EXVECT_COPY") }
};
```

4) Set the EXTB register to 0x0000FF80 (start address)

5.2 Operation Overview of the Software Configurable Interrupts

In this document, one program is used to introduce the following two processes: moving the exception vector table and the software configurable interrupt handling. Set the SEL_INT constant in the r_int_config.h file to switch the processes.

When performing software configurable interrupt handling, set SOFTWARE_CONFIG_INT in the SEL_INT constant. In the sample code, the default is EXCEP_HANDL.

5.2.1 Software Configurable Interrupt Handling

In software configurable interrupt handling, the following processing is performed to generate software configurable interrupts.

(1) Initial settings
The port, clock, and peripheral functions are initialized.
(2) Initialize the peripheral functions (software configurable interrupt)
The IER1A.IEN0 bit is set to “0” (interrupt request is disabled), and the SLIAR208 register (interrupt vector number 208) is set to “01h” (TGIA0 (TGRA interrupt capture/compare match) is assigned). After that, the SLIPRCR.WPCR bit is set to “1” (writing to the SLIARn register is disabled), and the IR208.IR flag is set to “0”.
(3) Start the MTU0.TCNT counter
The IER1A.IEN0 bit is set to “1” (interrupt request is enabled), the TSTRA.CST0 bit is set to “1” (MTU0.TCNT count is started).
(4) Compare match interrupt
When the MTU0.TCNT counter and MTU0.TGRA register values match, the IR208.IR flag for the TGIA0 interrupt becomes “1” and a compare match interrupt request is generated. When the MTU0.TCNT counter and MTU0.TGRA register values match, the MTU0.TCNT counter becomes “0000h”, and starts incrementing again. When an interrupt request is accepted, the IR208.IR flag becomes “0”. Every time a compare match interrupt occurs at an interval of 250 ms, LED 3 switches ON/OFF.
(5) Stop the MTU0.TCNT counter
When the 20th interrupt occurs, the IER1A.IEN0 bit is set to “0” (interrupt request is disabled), and the TSTRA.CST0 bit is set to “0” (the MTU0.TCNT counter stops).

Table 5.1 shows Peripheral Functions Used and Intended Use, and Figure 5.2 shows Timing of Software Configurable Interrupts.

Table 5.1 Peripheral Functions Used and Intended Use

Peripheral functions	Intended use
MTU	Compare match
I/O port	LED ON

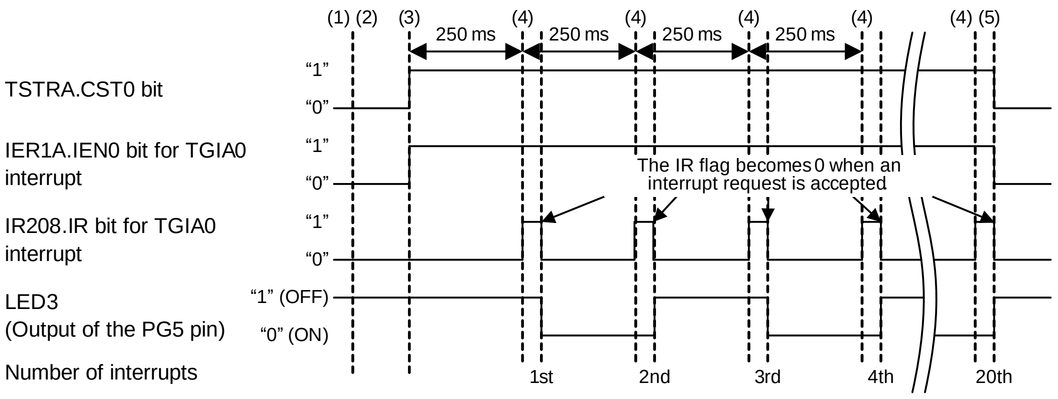

(1) (2)
(3)
(4)
(4) (5)
250 ms
TSTRA.CST0 bit
IER1A.IEN0 bit for TGIA0 interrupt
IR208.IR bit for TGIA0 interrupt
LED3 (Output of the PG5 pin)
Number of interrupts
“1”
“0”
“1” (OFF)
“0” (ON)
The IR flag becomes 0 when an interrupt request is accepted
1st
2nd
3rd
4th
20th

Figure 5.2 Timing of Software Configurable Interrupts

5.3 File Composition

Table 5.2 lists the Files Used in the Sample Code. Files generated by the integrated development environment are not included in this table.

Table 5.2 Files Used in the Sample Code

File Name	Outline	Remarks
main.c	Main processing	
r_init_stop_module.c	Stop processing for active peripheral functions after a reset	
r_init_stop_module.h	Header file for r_init_stop_module.c	
r_init_port_initialize.c	Nonexistent port initialization	
r_init_port_initialize.h	Header file for r_init_port_initialize.c	
r_init_clock.c	Clock initialization	
r_init_clock.h	Header file for r_init_clock.c	
r_int_config.c	Moving the exception vector table and initial setting of software configurable interrupts	
r_int_config.h	Header file for r_int_config.c	
r_ram_except_handlers.c	Exception handling function for RAM	
r_ram_except_handlers.h	Header file of exception handling function for RAM	
r_ram_except_vector_table.c	Exception vector table for RAM	

5.4 Option-Setting Memory

Table 5.3 lists the Option-Setting Memory Configured in the Sample Code. When necessary, set a value suited to the user system.

Table 5.3 Option-Setting Memory Configured in the Sample Code

Symbol	Address	Setting Value	Contents
OFS0	FE7F 5D04h to FE7F 5D07h	FFFF FFFFh	The IWDT is stopped after a reset. The WDT is stopped after a reset.
OFS1	FE7F 5D08h to FE7F 5D0Bh	FFFF FFFFh	The voltage monitor 0 reset is disabled after a reset. HOCO oscillation is disabled after a reset.
MDE	FE7F 5D00h to FE7F 5D03h	FFFF FFFFh	Little endian

5.5 Constants

Table 5.4 lists the Constants Used in the Sample Code.

Table 5.4 Constants Used in the Sample Code

Constant Name	Setting Value	Contents
EXCEP_HANDL	0	Exception handling
SOFTWARE_CONFIG_INT	1	Software configurable interrupt
SEL_INT	EXCEP_HANDL	Interrupt selection EXCEP_HANDL: Exception handling SOFTWARE_CONFIG_INT: Software configurable interrupt
OUTPUT_HIGH	1	High output
OUTPUT_LOW	0	Low output
LED_ON	OUTPUT_LOW	Output data during LED ON OUTPUT_HIGH : High output OUTPUT_LOW : Low output
LED_OFF	OUTPUT_HIGH	Output data during LED OFF OUTPUT_HIGH : High output OUTPUT_LOW : Low output
LED0_PODR	PORT7.PODR.BIT.B3	Port output data register of the port connected to LED0 (PODR)
LED0_PDR	PORT7.PDR.BIT.B3	Port direction register of the port connected to LED0 (PDR)
LED3_PODR	PORTG.PODR.BIT.B5	Port output data register of the port connected to LED3 (PODR)
LED3_PDR	PORTG.PDR.BIT.B5	Port direction register of the port connected to LED3 (PDR)
MTU_PCLK_HZ	120000000	Operation frequency of multi-function timer pulse unit (MTU)

5.6 Variables

Table 5.5 lists the Static Variables.

Table 5.5 Static Variables

Type	Variable Name	Contents	Function
static uint8_t	gs_int_cnt	Interrupt counter	mtu_init peria_inta208

5.7 Functions

Table 5.6 lists the Functions.

Table 5.6 Functions

Function Name	Outline
main	Main processing
R_INIT_StopModule	Stop processing for active peripheral functions after a reset
R_INIT_Port_Initialize	Nonexistent port initialization
R_INIT_Clock	Clock initialization
R_INT_Config	Moving the exception vector table and setting software configurable interrupts
port_init	Port initialization
exception_handling_main	Moving the exception vector table
mtu_init	MTU initialization
software_config_int_main	Software configurable interrupt setting
mtu_start	Start incrementing the MTU
mtu_stop	Stop incrementing the MTU
peria_inta208	Software configurable interrupt handling (interrupt vector number 208)
ram_excep_super_visor_inst	Privileged instruction exception handling
ram_dummy	Dummy processing

5.8 Function Specifications

The following tables list the sample code function specifications.

main	
Outline	Main processing
Header	None
Declaration	void main (void)
Description	Executes a privileged instruction or starts incrementing the MTU after a reset.
Arguments	None
Return Value	None

R_INIT_StopModule	
Outline	Stop processing for active peripheral functions after a reset
Header	r_init_stop_module.h
Declaration	void R_INIT_StopModule (void)
Description	Configures the setting to enter the module-stop state.
Arguments	None
Return Value	None
Remarks	Transition to the module-stop state is not performed in the sample code. Refer to the “RX65N Group, RX651 Group Initial Settings” application note for details on this function.

R_INIT_Port_Initialize	
Outline	Nonexistent port initialization
Header	r_init_port_initialize.h
Declaration	void R_INIT_Port_Initialize (void)
Description	Initializes port direction registers for ports that do not exist in products.
Arguments	None
Return Value	None
Remarks	The number of pins in the sample code is set for the 176-pin package (PIN_SIZE= 176). After this function is called, when writing in byte units to the PDR registers or PODR registers which have nonexistent ports, set the I/O select bits for nonexistent ports in the PDR registers to 1, and set the output data store bits for nonexistent ports in the PODR registers to 0. Refer to the “RX65N Group, RX651 Group Initial Settings” application note for details on this function.

R_INIT_Clock	
Outline	Clock initialization
Header	r_init_clock.h
Declaration	void R_INIT_Clock (void)
Description	Performs initial setting of the clock and sets ROM wait cycle.
Arguments	None
Return Value	None
Remarks	In the sample code, processing that does not use Sub Clock but uses 2 ROM WAIT cycles of System Clock and PLL is used. The set_ad_conversion_time function called by this function needs to be called when the PSW.I bit is “0” and also the ADCSR.ADST bit is “0”. For that reason, before calling this function, set the PSW.I bit to “0” (prohibits interrupt) and the ADCSR.ADST bit to “0”. Refer to the “RX65N Group, RX651 Group Initial Settings” application note for details on this function.

R_INT_Config	
Outline	Moving the exception vector table and setting software configurable interrupts
Header	r_int_config.h
Declaration	void R_INT_Config (void)
Description	Moves the exception vector table or sets software configurable interrupts.
Arguments	None
Return Value	None

port_init	
Outline	Port initialization
Header	r_int_config.h
Declaration	static void port_init (void)
Description	Makes initial setting of the port for LED0 and LED3 blinking.
Arguments	None
Return Value	None

exception_handling_main	
Outline	Moving the exception vector table
Header	r_int_config.h
Declaration	static void exception_handling_main (void)
Description	Moves the exception vector table.
Arguments	None
Return Value	None

mtu_init	
Outline	MTU initialization
Header	r_int_config.h
Declaration	static void mtu_init (void)
Description	Initializes the MTU.
Arguments	None
Return Value	None

software_config_int_main	
Outline	Software configurable interrupt setting
Header	r_int_config.h
Declaration	static void software_config_int_main (void)
Description	Sets the software configurable interrupt.
Arguments	None
Return Value	None

mtu_start	
Outline	Start incrementing the MTU counter
Header	r_int_config.h
Declaration	static void mtu_start (void)
Description	Clears the MTU0.TCNT counter, enables the TGIA0 interrupt request, and starts incrementing the MTU0.TCNT counter.
Arguments	None
Return Value	None

mtu_stop	
Outline	Stop incrementing the MTU counter
Header	r_int_config.h
Declaration	static void mtu_stop (void)
Description	Stops incrementing the MTU0.TCNT counter and disables the TGIA0 interrupt request.
Arguments	None
Return Value	None

peria_inta208	
Outline	Software configurable interrupt handling (interrupt vector number: 208)
Header	r_int_config.h
Declaration	void peria_inta208 (void)
Description	Switches LED3 between on and off. Calls the MTU incrementing stop function when the 20th interrupt occurs.
Arguments	None
Return Value	None

ram_excep_super_visor_inst	
Outline	Privileged instruction exception handling
Header	r_ram_except_handlers.h
Declaration	ram_excep_super_visor_inst (void)
Description	Handles a privileged instruction exception. Enters an infinite loop after LED0 is turned on.
Arguments	None
Return Value	None

ram_dummy	
Outline	Dummy processing
Header	r_ram_except_handlers.h
Declaration	ram_dummy (void)
Description	This function performs no operation.
Arguments	None
Return Value	None

5.9 Flowcharts

5.9.1 Main Processing

Figure 5.3 shows the Main Processing.

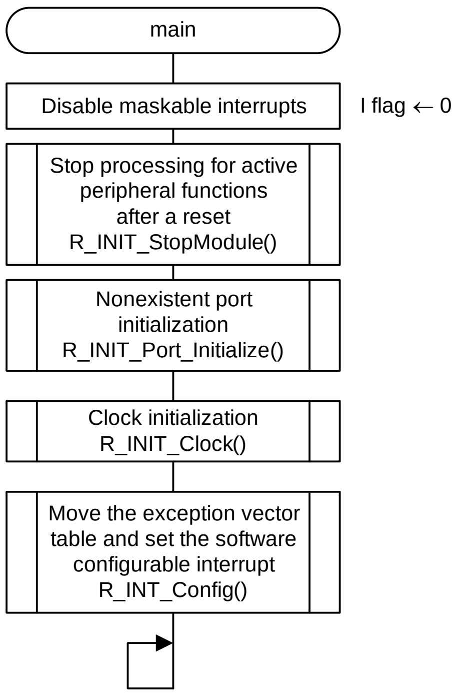

main
Disable maskable interrupts
I flag ← 0
Stop processing for active peripheral functions after a reset
R_INIT_StopModule()
Nonexistent port initialization
R_INIT_Port_Initialize()
Clock initialization
R_INIT_Clock()
Move the exception vector table and set the software configurable interrupt
R_INT_Config()

Figure 5.3 Main Processing

5.9.2 Moving the Exception Vector Table and Setting Software Configurable Interrupts

Figure 5.4 shows the flow chart of Moving the Exception Vector Table and Setting Software Configurable Interrupts.

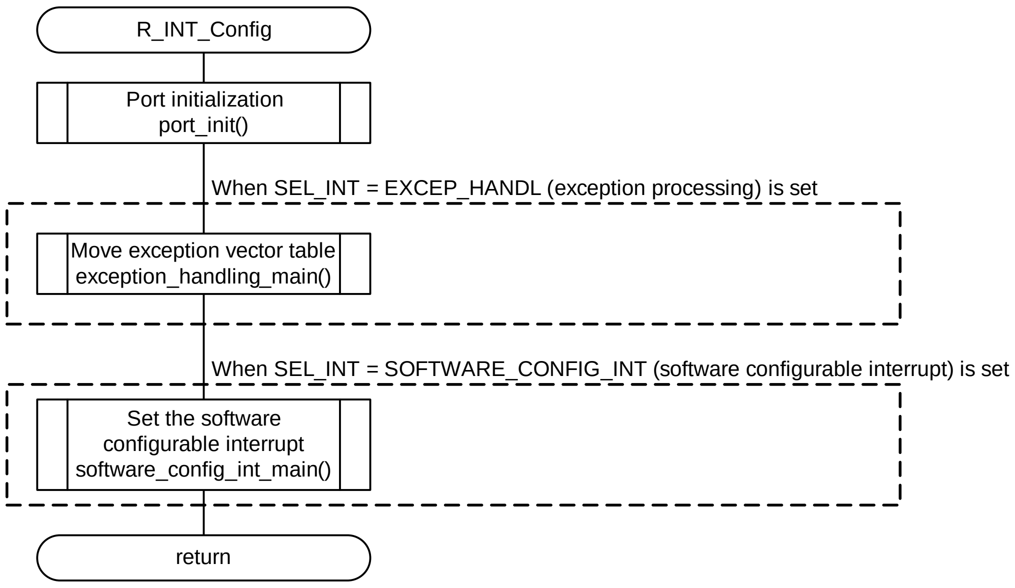

R_INT_Config
Port initialization
port_init()
When SEL_INT = EXCEP_HANDL (exception processing) is set
Move exception vector table
exception_handling_main()
When SEL_INT = SOFTWARE_CONFIG_INT (software configurable interrupt) is set
Set the software configurable interrupt
software_config_int_main()
return

Figure 5.4 Moving the Exception Vector Table and Setting Software Configurable Interrupts

5.9.3 Port Initialization

Figure 5.5 shows the Port Initialization.

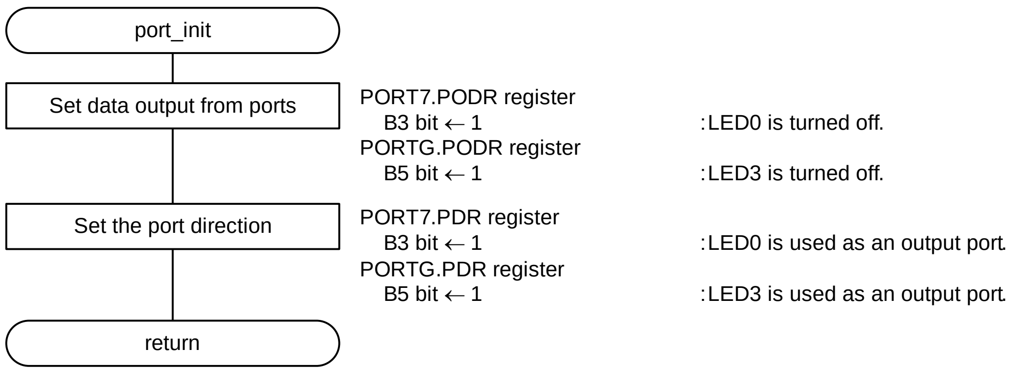
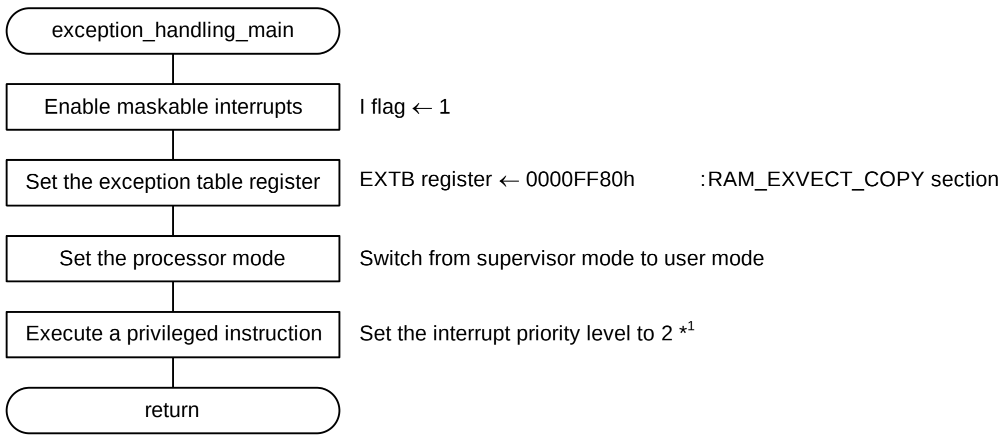


Figure 5.5 Port Initialization

5.9.4 Moving the Exception Vector Table

Figure 5.6 shows Moving the Exception Vector Table.

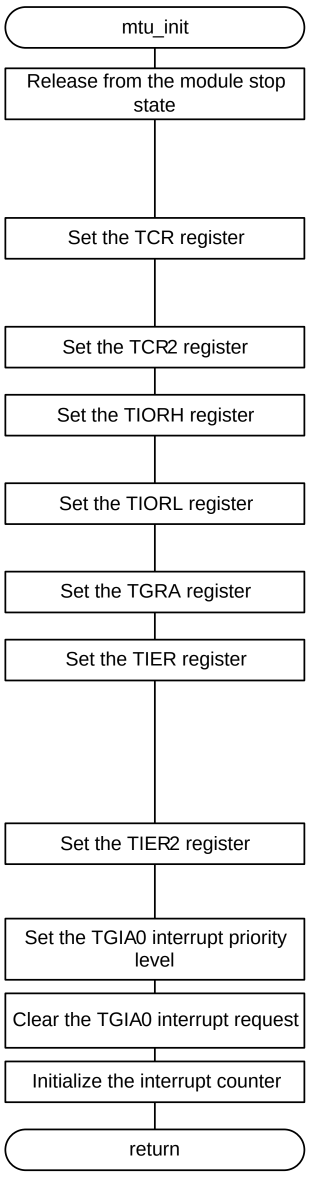

exception_handling_main
Enable maskable interrupts
I flag ← 1
Set the exception table register
EXTB register ← 0000FF80h :RAM_EXVECT_COPY section
Set the processor mode
Switch from supervisor mode to user mode
Execute a privileged instruction
Set the interrupt priority level to 2 *1
return

Note 1. Since the processor mode is user mode, a privileged instruction exception occurs.

Figure 5.6 Moving the Exception Vector Table

5.9.5 MTU Initialization

Figure 5.7 shows the MTU Initialization.

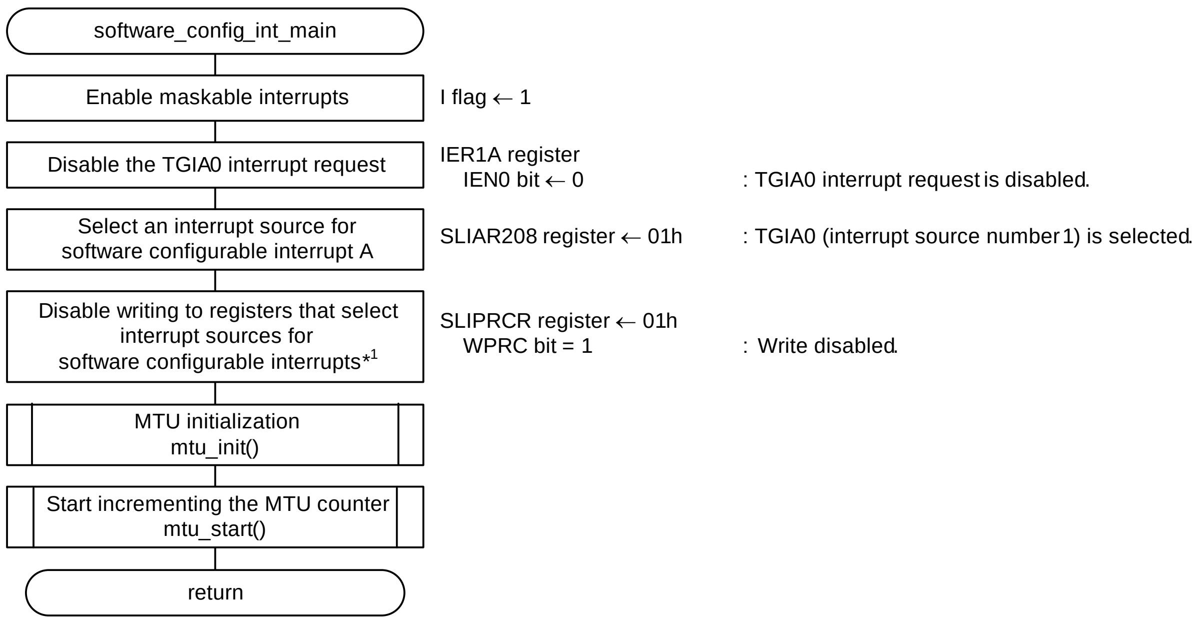


PRCR register ← A502h	
PRC1 bit = 1	: Enables writing to the related register
MSTPCRA register	
MSTPA9 bit ← 0	: Release from the MTU3 module stop state
PRCR register ← A500h	
PRC1 bit = 0	: Disables writing to the related register
MTU0.TCR register	
CCLR[2:0] bit ← 001b	: TGRA compare match/input capture clears TCNT
CKEG[1:0] bit ← 00b	: Counts at the rising edge
TPSC[2:0] bit ← 000b	: Counts with PCLKA/1024
MTU0.TCR2 register	
TPSC2[2:0] bit ← 101b	: Counts with PCLKA/1024
MTU0.TIORH register	
IOB[3:0] bit ← 0000b	: TGRB is the output compare register, MTIOC0B pin output is disabled
IOA[3:0] bit ← 0000b	: TGRA is the output compare register, MTIOC0A pin output is disabled
MTU0.TIORL register	
IOD[3:0] bot ← 0000b	: TGRD is the output compare register, MTIOC0D pin output is disabled
IOC[3:0] bot ← 0000b	: TGRC is the output compare register, MTIOC0C pin output is disabled
MTU0.TGRA register ← 7270h	: (120MHz / 1024) / (1 / 250ms) = 29296.875
MTU0.TIER register	
TTGE bit ← 0b	: A/D conversion start request occurrence is disabled
TCIEV bit ← 0b	: Interrupt request (TCIV) is disabled
TGIED bit ← 0b	: Interrupt request (TGID) is disabled
TGIEC bit ← 0b	: Interrupt request (TGIC) is disabled
TGIEB bit ← 0b	: Interrupt request (TGIB) is disabled
TGIEA bit ← 1b	: Interrupt request (TGIA) is enabled
MTU0.TIER2 register	
TTGE2 bit ← 0b	: A/D conversion start request by MTU0.TCNT and MTU0.TGRE compare match is disabled
TGIEF bit ← 0b	: Interrupt request (TGIF) disabled
TGIEE bit ← 0b	: Interrupt request (TGIE) disabled
IPR208 register	
IPR[3:0]bit ← 0001b	: TGIA0 interrupt priority level 1
IR208 register	
IR flag ← 0	: There is no TGIA0 interrupt request
Set the interrupt counter to 20	

Figure 5.7 MTU Initialization

5.9.6 Software Configurable Interrupt Setting

Figure 5.8 shows Software Configurable Interrupt Setting.


software_config_int_main
Enable maskable interrupts
I flag ← 1
Disable the TGIA0 interrupt request
IER1A register
IEN0 bit ← 0
: TGIA0 interrupt request is disabled.
Select an interrupt source for software configurable interrupt A
SLIAR208 register ← 01h
: TGIA0 (interrupt source number 1) is selected.
Disable writing to registers that select interrupt sources for software configurable interrupts*1
SLIPRCR register ← 01h
WPRC bit = 1
: Write disabled.
MTU initialization
mtu_init()
Start incrementing the MTU counter
mtu_start()
return

Note 1. After writing 1 to the WPRC bit, confirm that the WPRC bit is 1. Once the WPRC bit becomes 1, software cannot set the WPRC bit to 0. When setting other software configurable interrupt source select registers, set the registers before setting the WPRC bit to 1.

Figure 5.8 Software Configurable Interrupt Setting

5.9.7 Start Incrementing the MTU

Figure 5.9 shows the procedure to Start Incrementing the MTU.

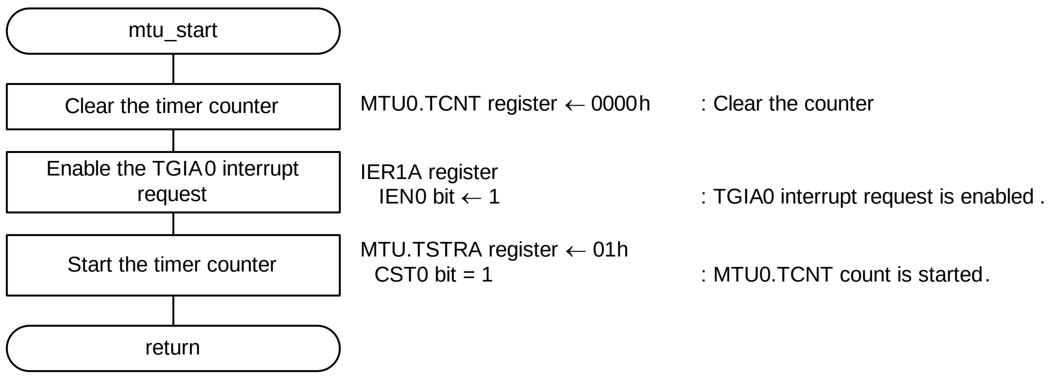


Figure 5.9 Start Incrementing the MTU

5.9.8 Stop Incrementing the MTU

Figure 5.10 shows the procedure to Stop Incrementing the MTU.

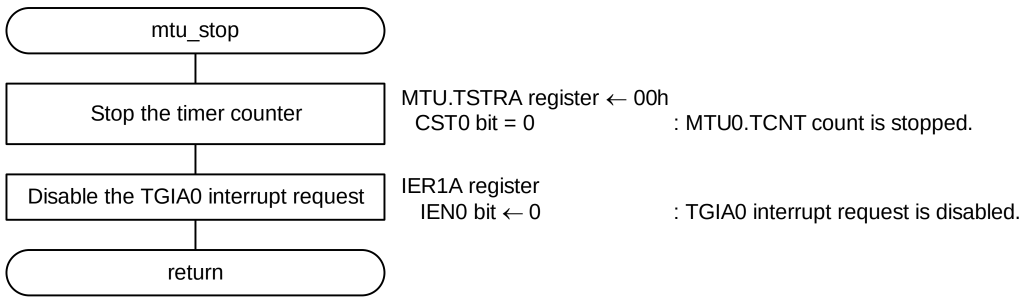

mtu_stop
Stop the timer counter
MTU.TSTRA register ← 00h
CST0 bit = 0
: MTU0.TCNT count is stopped.
Disable the TGIA0 interrupt request
IER1A register
IEN0 bit ← 0
: TGIA0 interrupt request is disabled.
return

Figure 5.10 Stop Incrementing the MTU

5.9.9 Software Configurable Interrupt Handling (Interrupt Vector Number 208)

Figure 5.11 shows Software Configurable Interrupt Handling (Interrupt Vector Number 208).

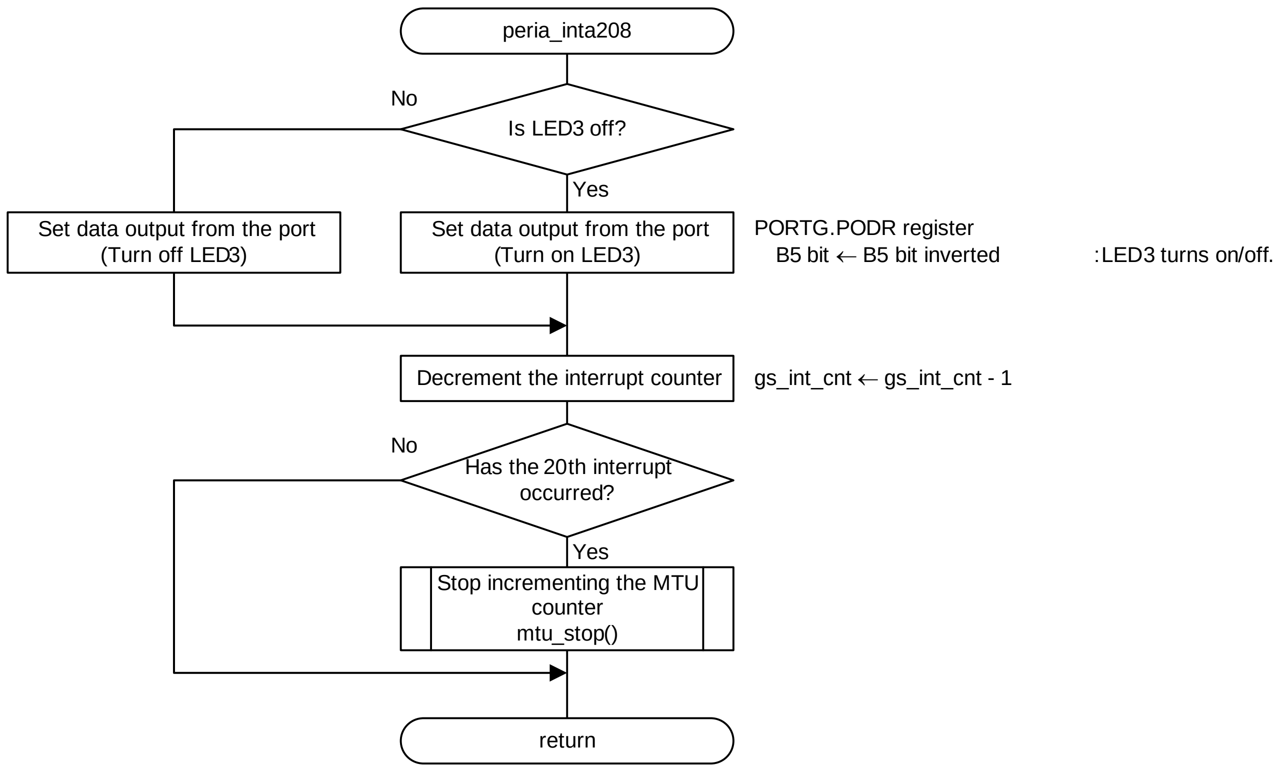

peria_inta208
Is LED3 off?
No
Yes
Set data output from the port (Turn off LED3)
Set data output from the port (Turn on LED3)
PORTG.PODR register
B5 bit ← B5 bit inverted
:LED3 turns on/off.
Decrement the interrupt counter
gs_int_cnt ← gs_int_cnt - 1
Has the 20th interrupt occurred?
No
Yes
Stop incrementing the MTU counter
mtu_stop()
return

Figure 5.11 Software Configurable Interrupt Handling (Interrupt Vector Number 208)

5.9.10 Privileged Instruction Exception Handling

Figure 5.12 shows Privileged Instruction Exception Handling.

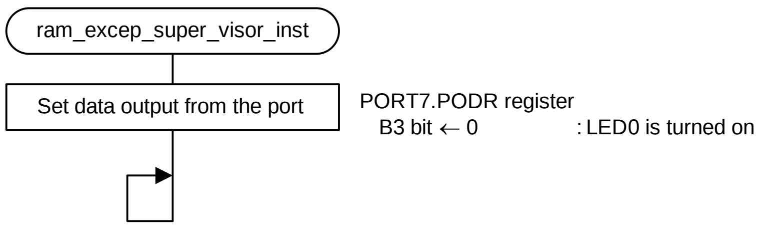


Figure 5.12 Privileged Instruction Exception Handling

5.9.11 Dummy Processing

Figure 5.13 shows the Dummy Processing.

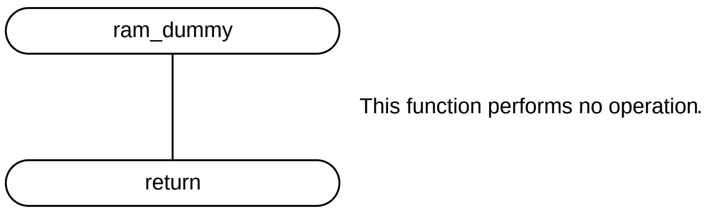


Figure 5.13 Dummy Processing

6. Porting Sample Codes to Other RX Family

Sample codes included in this application note can be ported to other RX Family loaded with the exception vector table and software configurable interrupts. This section describes an example to port sample codes to RX66T (Renesas Starter Kit for RX66T).

6.1 Before Porting

Confirm the following specifications before porting sample codes. If there is a difference in the specifications, the method described in this section may not be used. After making sure of the specifications, use this application.

- Specification of exception vector table and software configurable interrupts of the porting source and porting destination
- The MTU specification of the porting source and porting destination

6.2 Porting Procedure Flow

Figure 6.1 shows the Porting Procedure Flow.

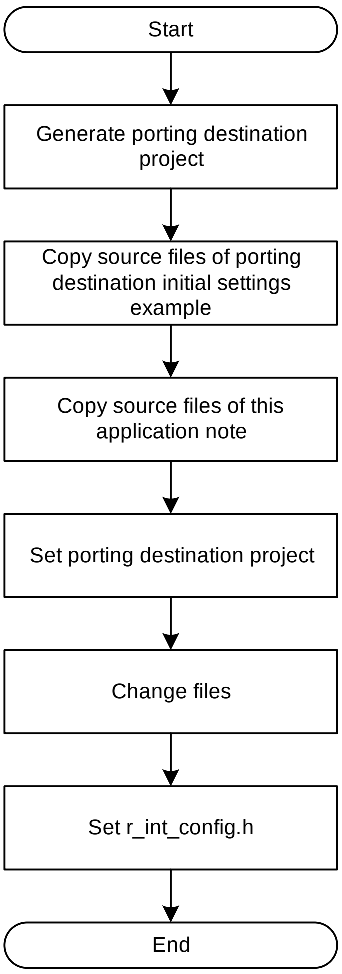

Start
Generate porting destination project
Copy source files of porting destination initial settings example
Copy source files of this application note
Set porting destination project
Change files
Set r_int_config.h
End

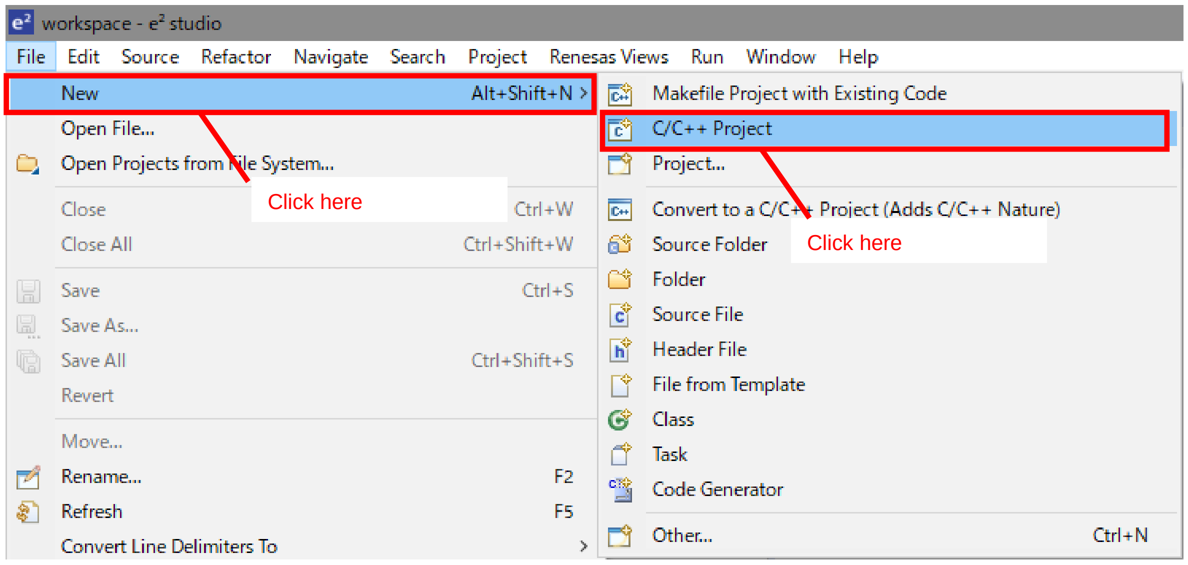
Figure 6.1 Porting Procedure Flow

6.3 Porting Procedure

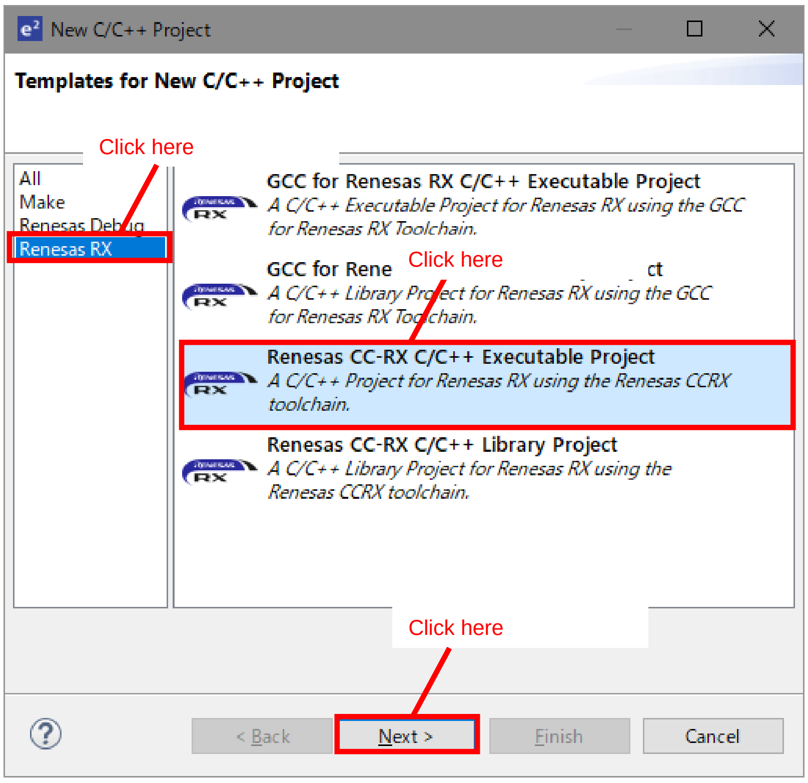
6.3.1 Generating a Porting Destination Project

Start e^2 studio and create a new project.

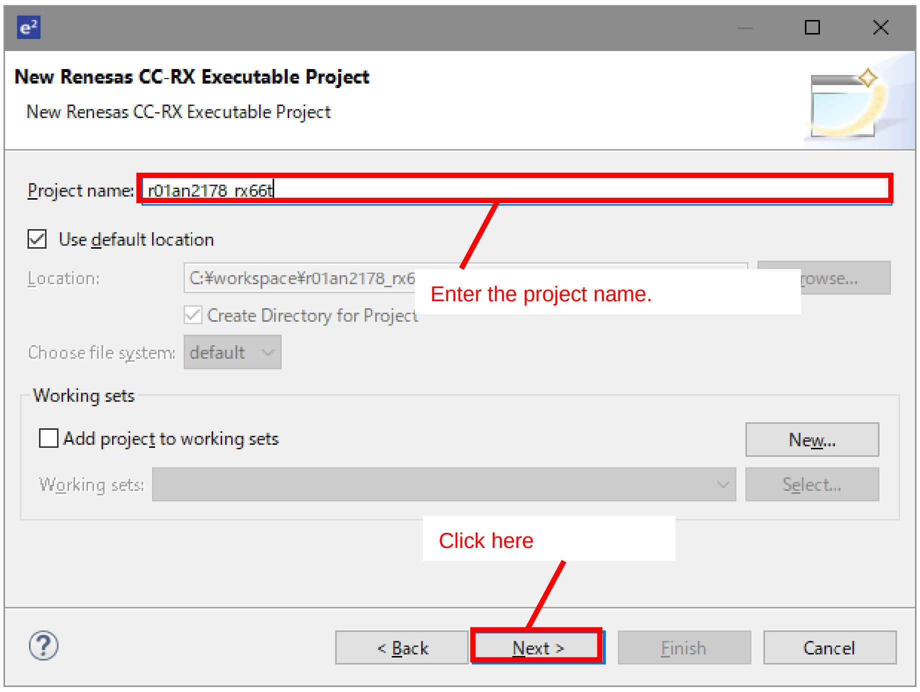
1) Generating a porting destination project
   1-1) Start e^2 studio and click [File].
   1-2) Click [C/C++ Project] of [New] to start the New C/C++ Project wizard.



   1-3) Click [Renesas RX].
   1-4) Click [Renesas CC-RX C/C++ Executable Project].
   1-5) Click [Next >].


New C/C++ Project
Templates for New C/C++ Project
Click here
All
Make
Renesas Debug
Renesas RX
GCC for Renesas RX C/C++ Executable Project
A C/C++ Executable Project for Renesas RX using the GCC for Renesas RX Toolchain.
GCC for Renesas RX C/C++ Library Project
A C/C++ Library Project for Renesas RX using the GCC for Renesas RX Toolchain.
Click here
Renesas CC-RX C/C++ Executable Project
A C/C++ Project for Renesas RX using the Renesas CCRX toolchain.
Renesas CC-RX C/C++ Library Project
A C/C++ Library Project for Renesas RX using the Renesas CCRX toolchain.
Click here
< Back
Next >
Finish
Cancel

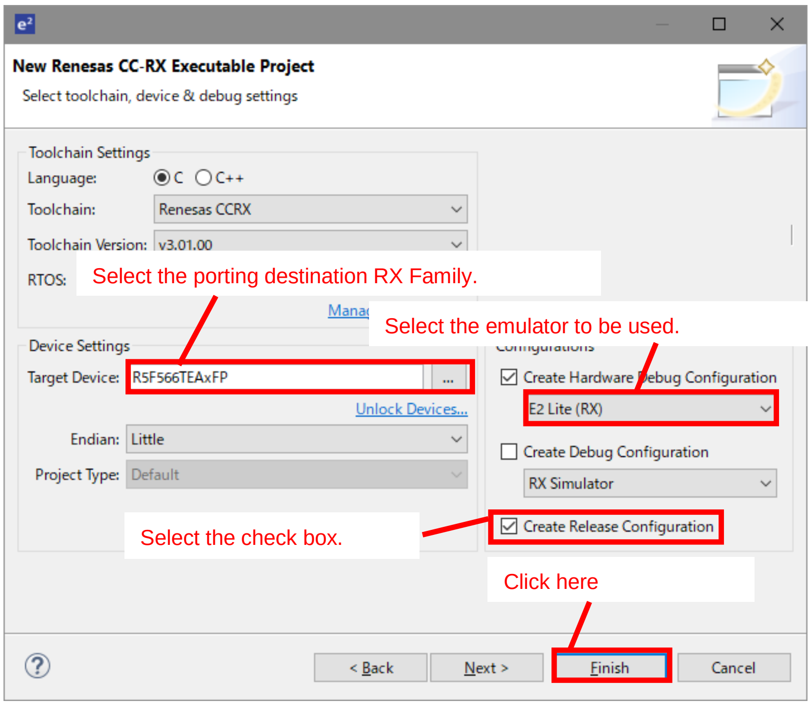
1-6) Enter the project name.
1-7) Click [Next >].



New Renesas CC-RX Executable Project
New Renesas CC-RX Executable Project

Project name: r01an2178_rx66t
☑ Use default location
Location: C:¥workspace¥r01an2178_rx6
☑ Create Directory for Project
Choose file system: default

1-8) Change [Target Device:] to [R5F566TEAxFP].
(When porting to another RX Family, change to the porting destination RX Family.)
1-9) Select the emulator to be used.
1-10) Tick the [Create Release Configuration] check box.
1-11) Click [Finish].



New Renesas CC-RX Executable Project
Select toolchain, device & debug settings

Toolchain Settings
Language: ◉ C ○ C++
Toolchain: Renesas CCRX
Toolchain Version: v3.01.00

Device Settings
Target Device: R5F566TEAxFP
Endian: Little
Project Type: Default

Configurations
☑ Create Hardware Debug Configuration
E2 Lite (RX)
☐ Create Debug Configuration
RX Simulator
☑ Create Release Configuration

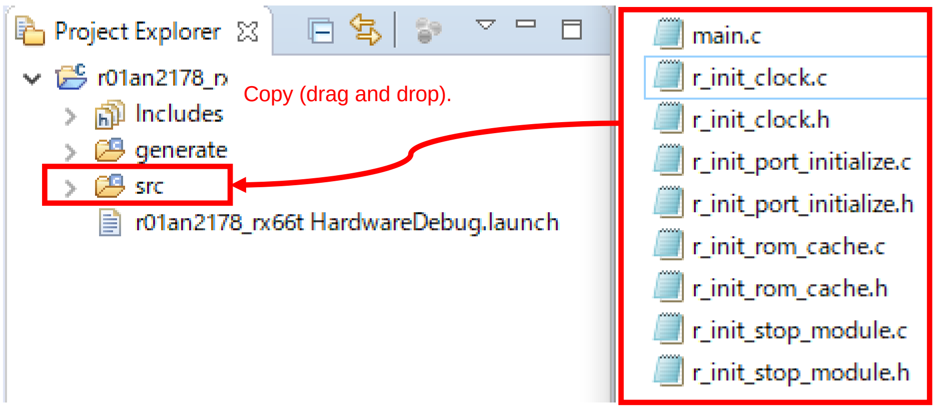
1-12) Delete [<Project name>.c] in the generated project.

6.3.2 Copying the Source Files of Porting Destination Initial Settings Example

Copy the source files of the initial settings example application note of the porting destination RX Family to the newly generated project.

1) Downloading the initial settings example application note
   1-1) From the Renesas Electronics' website, download [RX66T Group Initial Settings (R01AN4057)]. (When porting to another RX Family, download the initial settings example application note corresponding to the porting destination RX Family.)
   1-2) Extract the downloaded zip file to the desired folder.

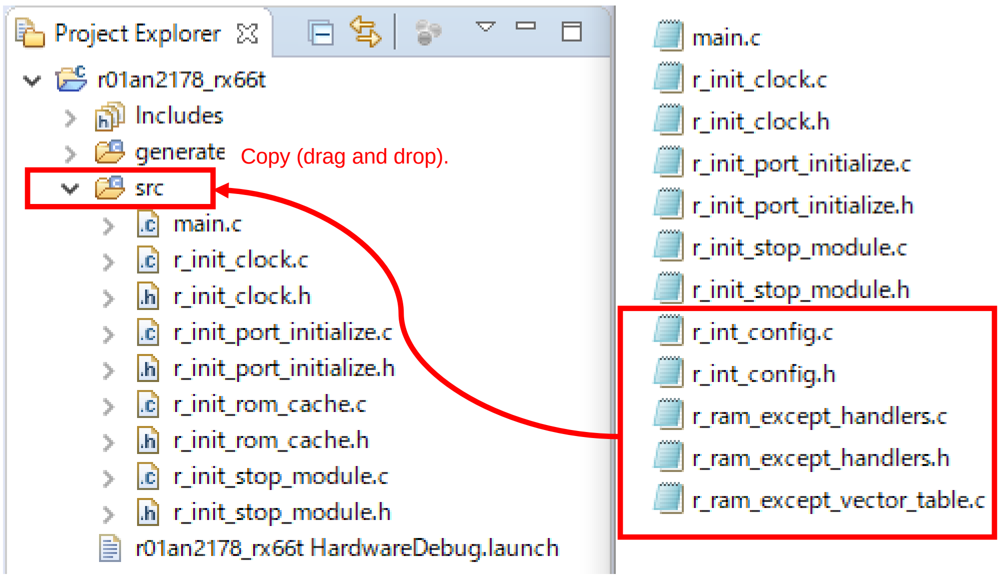
2) Copying the source files of the initial settings example application note to the project
   2-1) Use Explorer to open the extracted folder and copy all files from [r01an4057_rx66t] -> [r01an4057_src] to the generated project.



6.3.3 Copying the Source Files of the Application Note

Copy the source files of the application to the generated project.

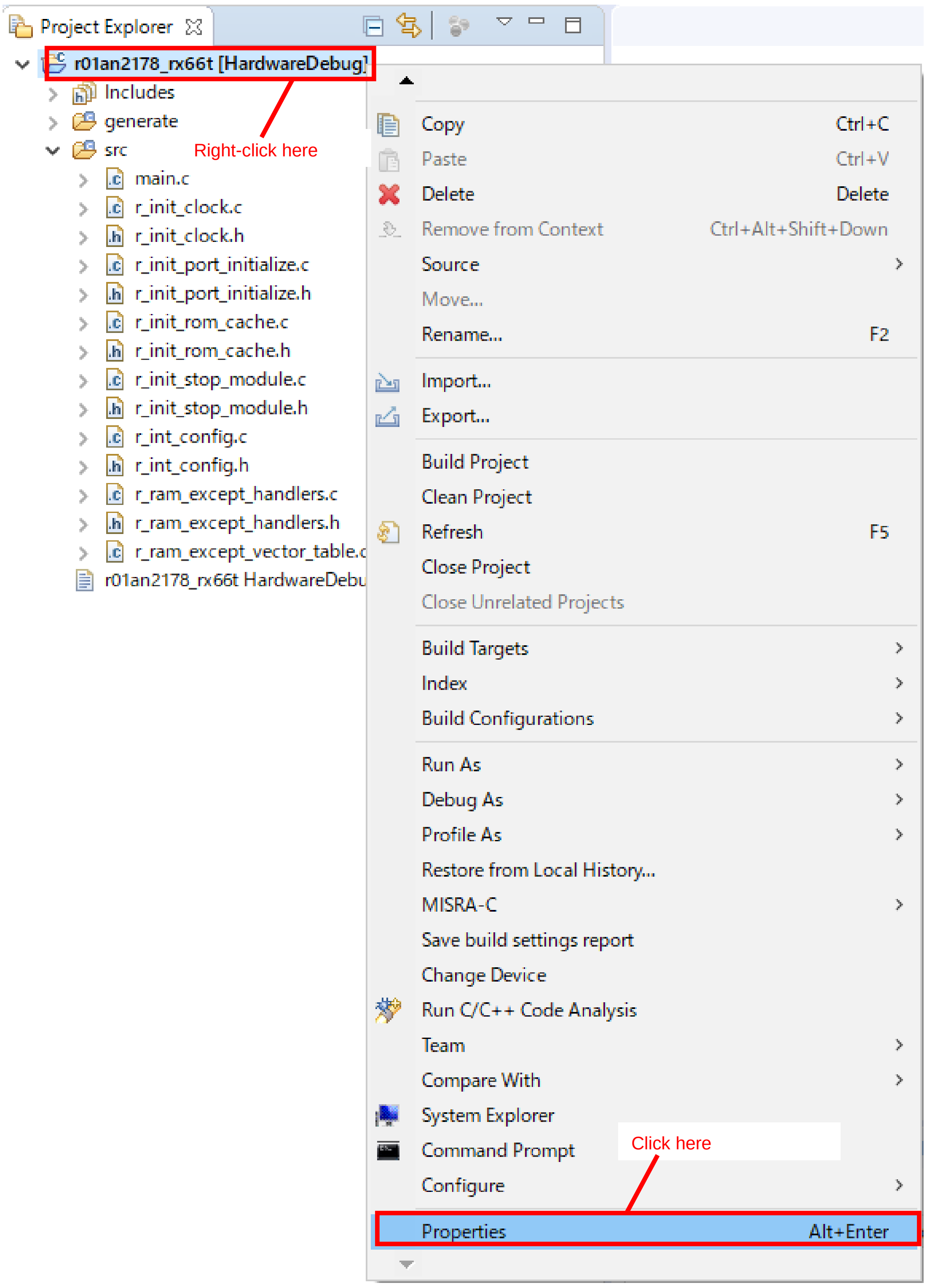
1) Copying the source files of the application to the project
   1-1) Copy [r_int_config.c], [r_int_config.h], [r_ram_except_handlers.c], [r_ram_except_handlers.h], and [r_ram_except_vector_table.c] from [r01an2178_rx65n_2m] -> [r01an2178_src] of this application to the project.



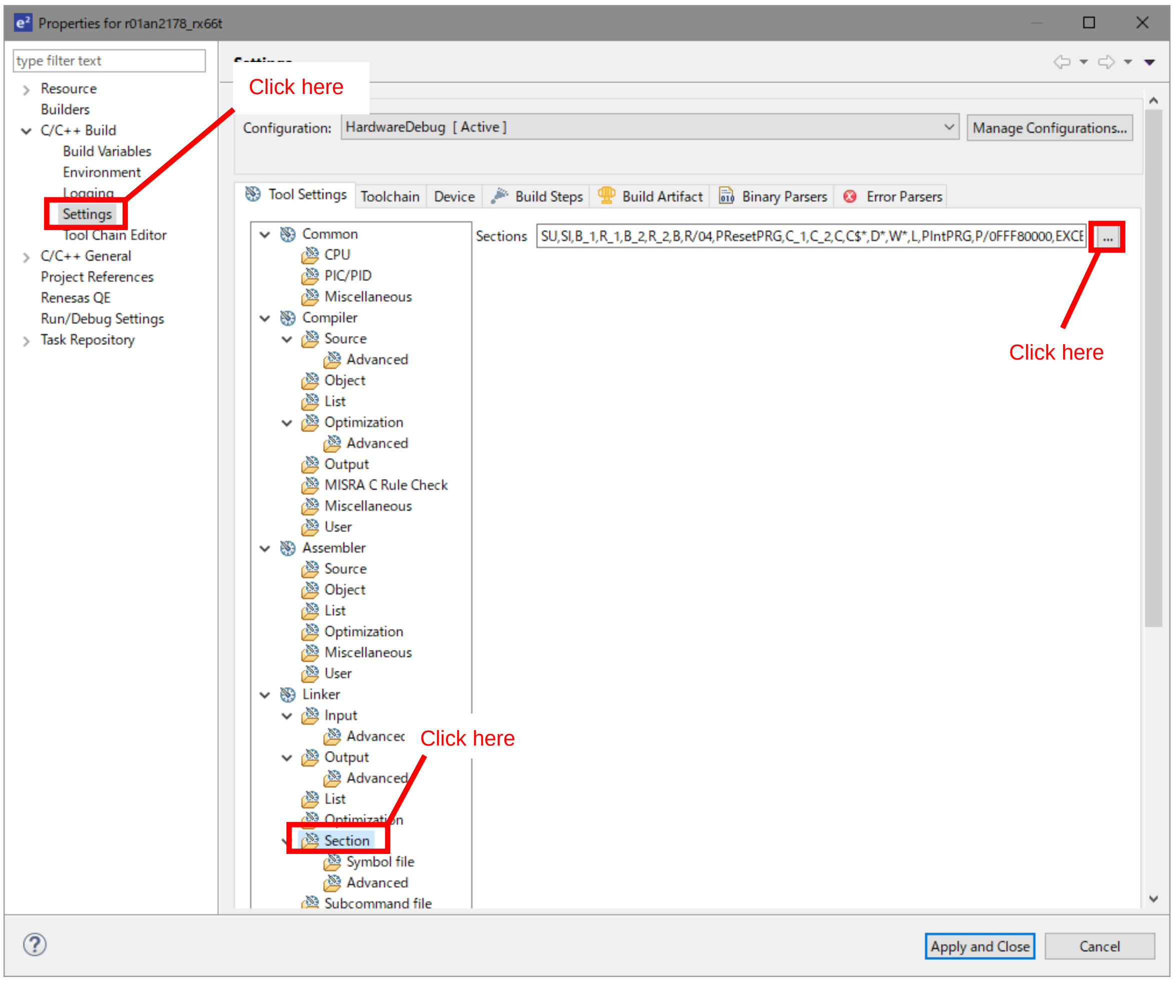
6.3.4 Setting Porting Destination Project

Change the build settings of the generated project.

1) Adding the final address section of the RAM
   1-1) Right-click the generated project and click [Properties].



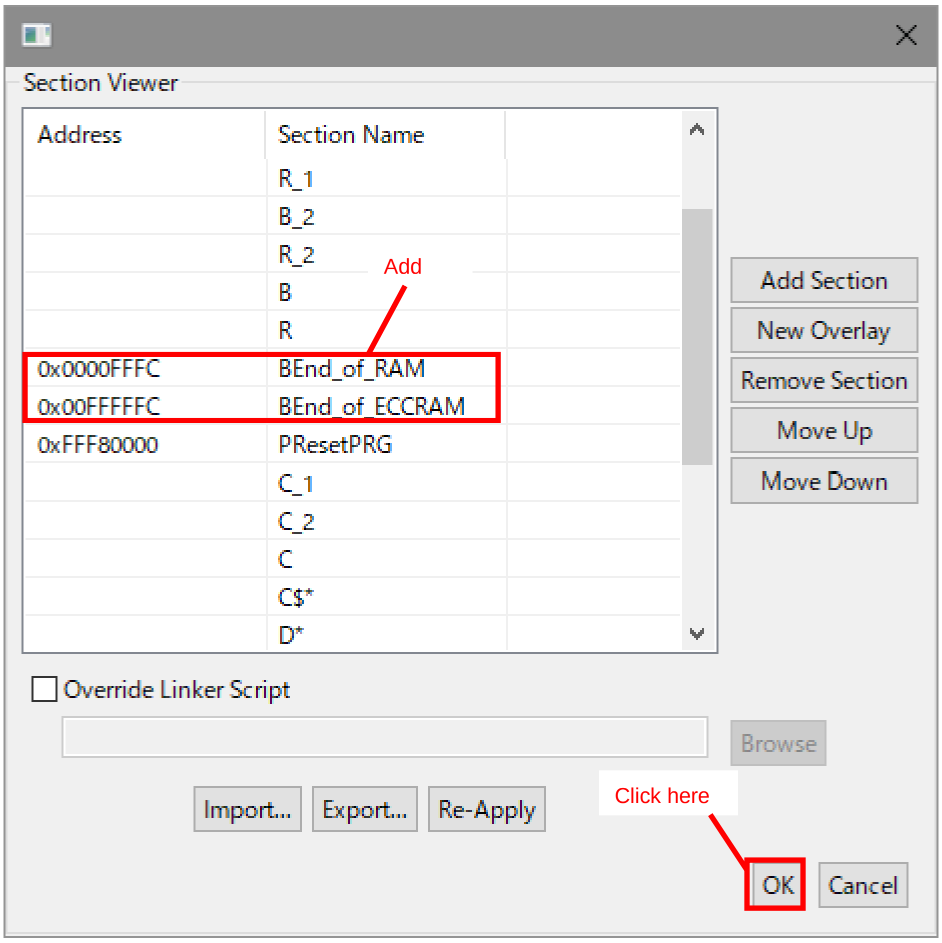
1-2) Click [C/C++ Build] -> [Settings].
1-3) Click [Tool Settings] -> [Linker] -> [Section].
1-4) Click […] at the right end of [Section].



Configuration: HardwareDebug [Active]

Sections SU,SI,B_1,R_1,B_2,R_2,B,R/04,PResetPRG,C_1,C_2,C,C$*,D*,W*,L,PIntPRG,P/0FFF80000,EXCE

1-5) Add the [End_of_RAM] section and the [End_of_ECCRAM] section.
1-6) Click [OK].



Address	Section Name
	R_1
	B_2
	R_2
	B
	R
0x0000FFFC	BEnd_of_RAM
0x00FFFFFC	BEnd_of_ECCRAM
0xFFF80000	PResetPRG
	C_1
	C_2
	C
	C$*
	D*

1-7) Click [Apply and Close].

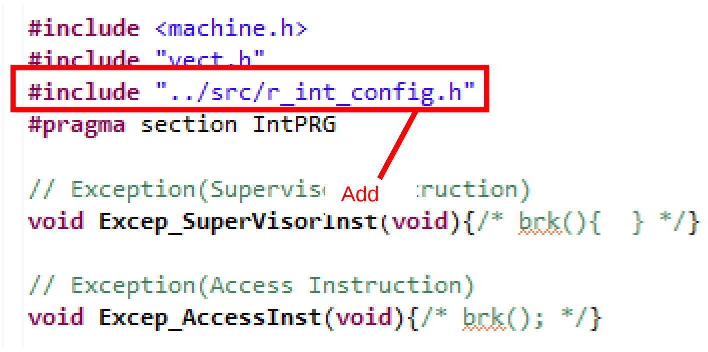
2) Adding a section for the exception vector table
   2-1) Refer to “5.1.2 Moving the Exception Vector Table to the RAM Area” to add a section for the exception vector table.

6.3.5 Changing Files

Change each source file copied in order to run the sample code of the application.

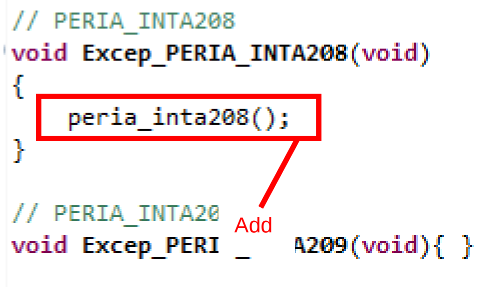
1) Changing the path to the include file.
   1-1) The include file path of the source file differs depending on the initial settings example; review and change the include file path according to the porting destination project.

2) Changing [intprg.c]
   2-1) To [intprg.c], add the include path to [r.int_config.h].



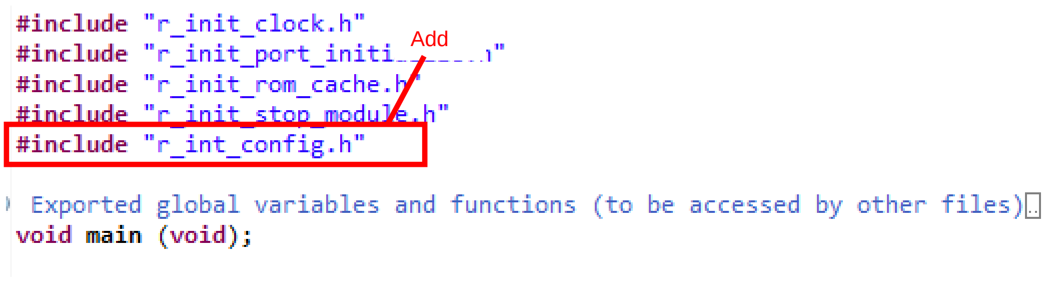
```
#include <machine.h>
#include "vect.h"
#include "../src/r_int_config.h"
#pragma section IntPRG

// Exception(Supervis       :ruction)
void Excep_SuperVisorInst(void){/* brk(){  } */}

// Exception(Access Instruction)
void Excep_AccessInst(void){/* brk(); */}
```

   2-2) To the [Excep_PERIA_INTA208] function of [intprg.c], add call processing of [peria_inta208].



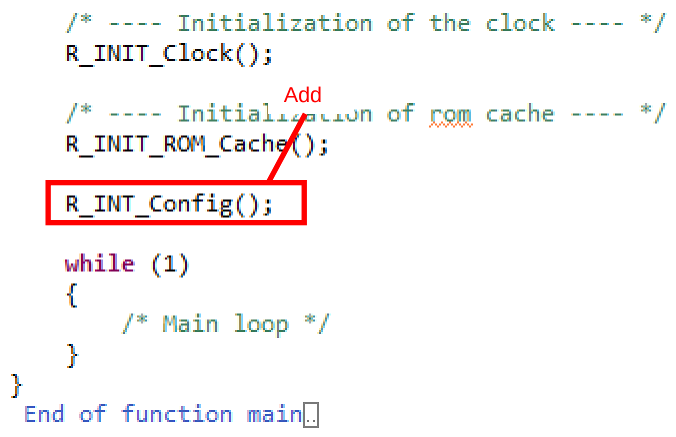
```
// PERIA_INTA208
void Excep_PERIA_INTA208(void)
{
    peria_inta208();
}

// PERIA_INTA20
void Excep_PERI _   A209(void){ }
```

3) Changing [main.c]
   3-1) To [main.c], add the include path to [r.int_config.h].



```
#include "r_init_clock.h"
#include "r_init_port_initi      1"
#include "r_init_rom_cache.h"
#include "r_init_stop_module.h"
#include "r_int_config.h"

 Exported global variables and functions (to be accessed by other files)
void main (void);
```

3-2) To the [main] function of [main.c], add call processing of the [R_INT_Config] function before the while statement.



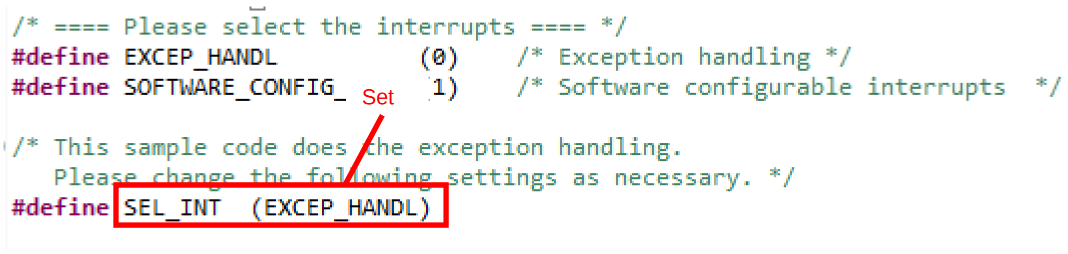
```
    /* ---- Initialization of the clock ---- */
    R_INIT_Clock();

    /* ---- Initialization of rom cache ---- */
    R_INIT_ROM_Cache();

    R_INT_Config();

    while (1)
    {
        /* Main loop */
    }
}
 End of function main
```

6.3.6 Setting r_int_config.h

Change [r_int_config.h] in accordance with the porting destination environment.

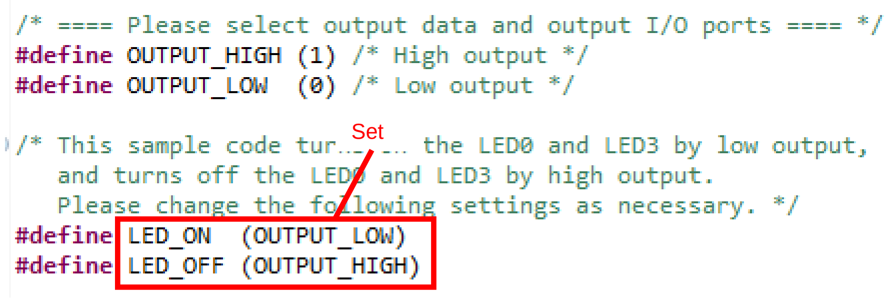
The setting value when porting to RX66T (Renesas Starter Kit for RX66T) is shown below. When porting to another RX Family, change to the setting value corresponding to the porting destination environment.

1) Setting the processing to be operated
    1-1) Set the SEL_INT constant in accordance with the processing to be operated. When operating moving processing of exception vector table, set EXCEP_HANDL in the SEL_INT constant. When performing software configurable interrupt handling, set SOFTWARE_CONFIG_INT in the SEL_INT constant.



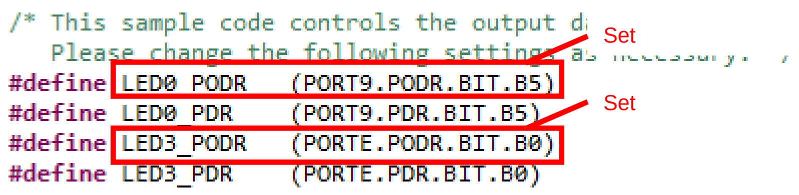
```
/* ==== Please select the interrupts ==== */
#define EXCEP_HANDL           (0)    /* Exception handling */
#define SOFTWARE_CONFIG_       1)    /* Software configurable interrupts  */

/* This sample code does the exception handling.
   Please change the following settings as necessary. */
#define SEL_INT  (EXCEP_HANDL)
```

2) Setting port output data during LED ON/OFF
    2-1) Set port output data during LED ON/OFF in the LED_ON constant and LED_OFF constant. Set [OUTPUT_LOW] in the LED_ON constant, and set [OUTPUT_HIGH] in the LED_OFF constant.



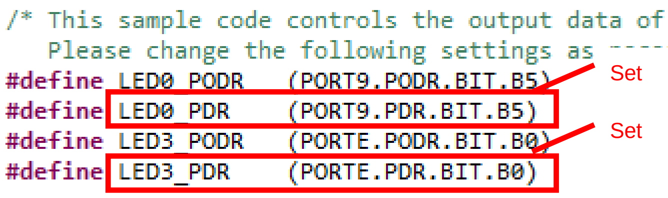
```
/* ==== Please select output data and output I/O ports ==== */
#define OUTPUT_HIGH (1) /* High output */
#define OUTPUT_LOW  (0) /* Low output */

/* This sample code tur... ... the LED0 and LED3 by low output,
   and turns off the LED0 and LED3 by high output.
   Please change the following settings as necessary. */
#define LED_ON  (OUTPUT_LOW)
#define LED_OFF (OUTPUT_HIGH)
```

3) Setting the port connected to the LED
    3-1) Set the port output data register connected to LED0 and LED3 in the LED0_PODR constant and LED3_PODR constant. Set [PORT9.PODR.BIT.B5] in the LED0_PODR constant, and set [PORTE.PODR.BIT.B0] in the LED3_PODR constant.



```
/* This sample code controls the output d
   Please change the following settings as necessary. */
#define LED0_PODR   (PORT9.PODR.BIT.B5)
#define LED0_PDR    (PORT9.PDR.BIT.B5)
#define LED3_PODR   (PORTE.PODR.BIT.B0)
#define LED3_PDR    (PORTE.PDR.BIT.B0)
```

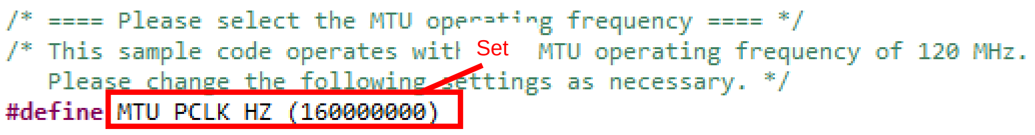
3-2) Set the port direction register connected to LED0 and LED3 in the LED0_PDR constant and LED3_PDR constant. Set [PORT9.PDR.BIT.B5] in the LED0_PDR constant, and set [PORTE.PDR.BIT.B0] in the LED3_PDR constant.



```
/* This sample code controls the output data of P73 and PG5.
   Please change the following settings as [illegible] */
#define LED0_PODR    (PORT9.PODR.BIT.B5)
#define LED0_PDR     (PORT9.PDR.BIT.B5)
#define LED3_PODR    (PORTE.PODR.BIT.B0)
#define LED3_PDR     (PORTE.PDR.BIT.B0)
```

4) Setting the operation frequency of the multi-function timer pulse unit (MTU)

4-1) Set the operation frequency of the multi-function timer pulse unit (MTU) in MTU_PCLK_HZ in units of Hz. Set [160000000].



```
/* ==== Please select the MTU operating frequency ==== */
/* This sample code operates with [illegible] MTU operating frequency of 120 MHz.
   Please change the following settings as necessary. */
#define MTU_PCLK_HZ (160000000)
```

7. Sample Code

Sample code can be downloaded from the Renesas Electronics website.

8. Reference Documents

User’s Manual: Hardware
RX65N Group, RX651 Group User's Manual: Hardware (R01UH0659)
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 Compiler CC-RX User’s Manual (R20UT3248)
The latest version can be downloaded from the Renesas Electronics website.

Revision History

		Description	
Rev.	Date	Page	Summary
1.00	Dec. 1, 2014	—	First edition issued
1.01	Nov. 2, 2015	—	RX71M Group is added to the target device
1.10	May.31.19	—	Changed the supported device to the RX Family loaded with the exception vector table and software configurable interrupts
		4, 9, 10, 11, 23	Changed the RAM_EXFUNC_COPY address. Changed the RAM_EXVECT_COPY address.
		5, 13, 19, 24, 25, 26	Changed the interrupt vector numbers and interrupt factor number of the software configurable interrupts. Changed the interrupt occurrence interval and the number of interrupts that occur.
		6	Changed Table 2.1 Operation Confirmation Conditions.
		7	Changed the pins used for LED output.
		14	Changed Table 5.3 Option-Setting Memory Configured in the Sample Code.
		15	Changed Table 5.4 Constants Used in the Sample Code.
		16	Changed Table 5.5 Static Variables.
		17	Changed 5.8 Function Specifications.
		21	Changed 5.9 Flowcharts.
		29	Added 6. Porting Sample Code to Other RX Family
		41	Changed 8. Reference Documents.
		Programs	• Changed the target device. • Deleted the file. main.h • Added files. r_int_config.c r_int_config.h • Added macro definitions. LED_ON LED_OFF LED0_PODR LED0_PDR LED3_PODR LED3_PDR MTU_PCLK_HZ • Added the function. R_INT_Config • Changed the functions. port_init mtu_init software_config_int_main mtu_start mtu_stop peria_inta208 ram_excep_super_visor_inst

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit 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. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance 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 the 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.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.