SH7450 Group, SH7451 Group

Timer Unit (TMU) Interrupt Sample Code

R01AN0570EJ0100
Rev. 1.00
Mar. 16, 2012

Abstract

This application note describes sample code that can be used by users who will be creating software for the SH7450 and SH7451 Group (referred to as the SH7450 Group in this document) microcontrollers for the first time. This sample code uses the SH7450 Group timer unit (TMU), TMU interrupts, and ports to turn on and turn off LEDs on the SH7450 evaluation bard (R0K474504C000BR or R0K474504C010BR).

Products

SH7450 Group, SH7451 Group

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

Contents

1. Specifications

This sample code uses SH7450's built-in TMU channel 0 (TM0) to continuously measure 1-second intervals and to update the pattern displayed on the SH7450 evaluation board LEDs every second. The LED pattern is updated in the TM0 interrupt handling by incrementing the port data for port G, which is connected to the LEDs, every second starting from 0.

Note: See the SH7450 evaluation board specifications for details on the evaluation board.

Table 1.1 lists the peripheral functions and their applications.

Table 1.1 Peripheral Functions and Their Applications

Peripheral Function	Application
TMU	Counts the 1-second intervals for updating the LEDs.
I/O port G	Outputs the LED display data (8 bits).
Interrupt controller (INTC)	Interrupt requests are controlled by the TM0 underflow.

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	SH7450 Group, SH7451 Group
Operating frequencies	Input frequency: 20 MHz CPU clock (Ick): 240 MHz SHwy clock (SHck): 80 MHz Peripheral clock (Pck): 40 MHz
Operating voltage	PVcc = Vcc = PLLVcc = Avcc = 5 V, Vdd = 1.5 V
Operating mode	Single-chip mode
Integrated development environment	Renesas Electronics High-performance Embedded Workshop (referred to as HEW below) Version 4.09.00.007
C/C++ compiler	Renesas Electronics C/C++ Compiler Package for SuperH RISC engine family V.9.04 Release 00
	Compile options: The following options are the HEW default settings. -cpu=sh4a -object="$(CONFIGDIR)\$(FILELEAF).obj" -debug -gbr=auto -chgincpath -errorpath -global_volatile=0 -opt_range=all -infinite_loop=0 -del_vacant_loop=0 -struct_alloc=1 -nologo
Sample code version	Version 1.00
Board used	SH7450 evaluation board (catalog number: R0K474504C000BR/R0K474504C010BR)

3. Reference Application Note

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

SH7450 Group/SH7451 Group Register Definition Header File (R01AN0190EJ0102)

4. Hardware

4.1 Pin Used

Table 4.1 lists the pin used and its function.

Table 4.1 Pin Used and Its Function

Pin Name	I/O	Function
Port G (bit 7 to bit 0)	Output	Control the LEDs on the SH7450 evaluation board

4.2 Register Settings

This section presents the main register settings used in the sample code. Note that these values are the values used in the sample code and that some of these values differ from the corresponding initial value.

(1) Status Register (SR)

Address	Setting Value	Bit		Description
—	H'4000 0000 or H'5000 0000 or H'7000 0000	31	—	0: Reserved bit
		30	MD	0: User mode 1: Privileged mode
		29	RB	0: R0_BANK0 to R7_BANK0 are accessed as general-purpose registers R0 to R7. R0_BANK1 to R7_BANK1 are accessed by the LDC/STC instructions. 1: R0_BANK1 to R7_BANK1 are accessed as general-purpose registers R0 to R7. R0_BANK0 to R7_BANK0 are accessed by the LDC/STC instructions.
		28	BL	0: Interrupt request are not masked 1: Interrupt request are masked
		27 to 16	—	0: Reserved bits
		15	FD	0: FPU instructions enabled
		14 to 10	—	0: Reserved bits
		9	M	0: Used with the DIV0S, DIV0U, and DIV1 instructions.
		8	Q	0: Used with the DIV0S, DIV0U, and DIV1 instructions.
		7 to 4	IMASK	0: Levels of IMASK and lower are masked.
		3, 2	—	0: Reserved bits
		1	S	0: Specifies the MAC instruction saturation operation.
		0	T	0: Indicates the true/false condition, carry, borrow, overflow, underflow, and other items.

(2) Port G Data Register (PGDR): Holds the port G I/O data

Address	Setting Value	Bit		Description
H'FFFF 5802	H'0000 to H'00FF	15 to 8	—	0: Reserved bits
		7 to 0	PG7DR to PG0DR	When set for output: the write values are output to the corresponding pins. When set for input: the pin states can be read out.

(3) Port G I/O Register (PGIOR): Sets the I/O direction of the port G pin

Address	Setting Value	Bit		Description
H'FFFF 5806	H'00FF	15 to 8	—	0: Reserved bits
		7 to 0	PG7IOR to PG0IOR	1: The corresponding pin is set to output.

(4) Port G Control Register 1 (PGCR1): Selects the functions of multiplexed pins of port G

Address	Setting Value	Bit		Description
H'FFFF 5816	H'0000	15	—	0: Reserved bit
		14 to 12	PG3MD	PG3: Mode bits 000: PG3 I/O (port)
		11	—	0: Reserved bit
		10 to 8	PG2MD	PG2: Mode bits 000: PG2 I/O (port)
		7	—	0: Reserved bit
		6 to 4	PG1MD	PG1: Mode bits 000: PG1 I/O (port)
		3	—	0: Reserved bit
		2 to 0	PG0MD	PG0: Mode bits 000: PG0 I/O (port)

(5) Port G Control Register 2 (PGCR2): Selects the functions of multiplexed pins of Port G

Address	Setting Value	Bit		Description
H'FFFF 5814	H'0000	15	—	0: Reserved bit
		14 to 12	PG7MD	PG7: Mode bits 000: PG7 I/O (port)
		11	—	0: Reserved bit
		10 to 8	PG6MD	PG6: Mode bits 000: PG6 I/O (port)
		7	—	0: Reserved bit
		6 to 4	PG5MD	PG5: Mode bits 000: PG5 I/O (port)
		3	—	0: Reserved bit
		2 to 0	PG4MD	PG4: Mode bits 000: PG4 I/O (port)

(6) TM Start Register (TMSTR): Selects TMnCNT (n = 0 to 2) counter operating/stopped state

Address	Setting Value	Bit		Description
H'FFFF D004	H'00 or H'01	7 to 3	—	0: Reserved bits
		2	STR2	TM2 counter start bit 0: Stops the TM2CNT counter count operation.
		1	STR1	TM1 counter start bit 0: Stops the TM1CNT counter count operation.
		0	STR0	TM0 counter start bit 0: Stops the TM0CNT counter count operation. 1: Starts the TM0CNT counter count operation.

(7) TM0 Constant Register (TM0COR): Holds the TM0CNT counter reload value

Address	Setting Value	Bit		Description
H'FFFF D008	H'0000 9896	31 to 0	TM0COR	The 32-bit value loaded into the TM0CNT counter when the TM0CNT counter underflows.

(8) TM0 Counter (TM0CNT): The TM0 counter value

Address	Setting Value	Bit		Description
H'FFFF D00C	H'0000 9896	31 to 0	TM0CNT	This registers counts down according to the input clock selected by the TM0CR register TPSC bit.

(9) TM0 Control Register (TM0CR): Selects the counter clock and controls interrupts when an underflow occurs

Address	Setting Value	Bit		Description
H'FFFF D010	H'0024	15 to 9	—	0: Reserved bits
		8	UNF	Underflow flag Cleared: When 0 is written to the UNF bit. Set: When the TM0CNT counter underflows.
		7, 6	—	0: Reserved bits
		5	UNIE	Underflow interrupt control bit 1: The underflow interrupt (TUNI) is enabled.
		4, 3	—	0: Reserved bits
		2 to 0	TPSC	Selects the TM0CNT counter count clock. 100: TM0CNT is incremented at Pck/1024.

(10) User Interrupt Mask Level Register (USERIMASK): Masks interrupts with a level less than the UIMASK set value.

Address	Setting Value	Bit		Description
H'FFFF F300	H'A500 0000	31 to 24	USERIMASKKEY	USERIMASK register write key code bits H'A5: Allows the UIMASK bits to be set.
		23 to 8	—	0: Reserved bits
		7 to 4	UIMASK	User interrupt mask level bits Interrupts with a level less than UIMASK set value are masked.
		3 to 0	—	0: Reserved bits

(11) Interrupt Priority Setting Register 0 (INT2PRI0)

Sets the priority (a level from 31 to 0) for the built-in peripheral module interrupts. The larger the setting value, the higher the priority. Each interrupt factor is assigned 5 bits to which 32 values for 30 levels (the setting values H'00 and H'01 result in the same state as the interrupt being masked) are allocated.

Address	Setting Value	Bit		Description
H'FFFF F400	H'0200 0000	31 to 29	—	0: Reserved bits
		28 to 24	TUNI0	Sets the priority (a level from 31 to 0) for the TUNI0 (TMU) interrupt.
		23 to 21	—	0: Reserved bits
		20 to 16	TUNI1	Sets the priority (a level from 31 to 0) for the TUNI1 (TMU) interrupt.
		15 to 13	—	0: Reserved bits
		12 to 8	TUNI2	Sets the priority (a level from 31 to 0) for the TUNI2 (TMU) interrupt.
		7 to 5	—	0: Reserved bits
		4 to 0	Reserved	—

(12) Interrupt Mask Clear Register 0 (INT2MSKCR)

The INT2MSKCR register can clear the mask set with interrupt mask register 0 (INT2MSKR). Setting a bit in this register to 1 clears the mask for the interrupt factor corresponding to that bit. This register always returns 0 when read.

Address	Setting Value	Bit		Description
H'FFFF F43C	H'0000 0001	31, 30	—	0: Reserved bits
		29	CMIG5	Timer G5 interrupt mask clear setting bit
		28	CMIG4	Timer G4 interrupt mask clear setting bit
		27	CMIG3	Timer G3 interrupt mask clear setting bit
		26	CMIG2	Timer G2 interrupt mask clear setting bit
		25	CMIG1	Timer G1 interrupt mask clear setting bit
		24	CMIG0	Timer G0 interrupt mask clear setting bit
		23	TF	Timer F interrupt mask clear setting bit
		22	TA	Timer A interrupt mask clear setting bit
		21	ADC	ADC interrupt mask clear setting bit
		20	IICI	IIC3 interrupt mask clear setting bit
		19	DRO	DRO interrupt mask clear setting bit
		18	DRI2	DRI2 interrupt mask clear setting bit
		17	DRI1	DRI1 interrupt mask clear setting bit
		16	DRI0	DRI0 interrupt mask clear setting bit
		15	HUDI	H-UDI interrupt mask clear setting bit
		14	RSPI2	RSPI2 interrupt mask clear setting bit
		13	RSPI1	RSPI1 interrupt mask clear setting bit
		12	RSPI0	RSPI0 interrupt mask clear setting bit
		11	SCIF3	SCIF3 interrupt mask clear setting bit
		10	SCIF2	SCIF2 interrupt mask clear setting bit
		9	SCIF1	SCIF1 interrupt mask clear setting bit
		8	SCIF0	SCIF0 interrupt mask clear setting bit
		7	DMAC6T11	DMA6 to DMA11 interrupt mask clear setting bit
		6	DMAC4T5	DMA4 and DMA5 interrupt mask clear setting bit
		5	DMAC0T3	DMA0 to DMA3 interrupt mask clear setting bit
		4	WDT	WDT interrupt mask clear setting bit
		3	—	0: Reserved bit
		2	TUNI2	TMU2 interrupt mask clear setting bit
		1	TUNI1	TMU1 interrupt mask clear setting bit
		0	TUNI0	TMU0 interrupt mask clear setting bit

5. Software

5.1 Operation Overview

This sample code uses the SH7450's built-in TM0 unit to continuously measure 1-second intervals and increments the port G output data in the TM0 interrupt handler. This operation turns the LEDs on the SH7450 evaluation board on and off. Figure 5.1 outlines the overall process.

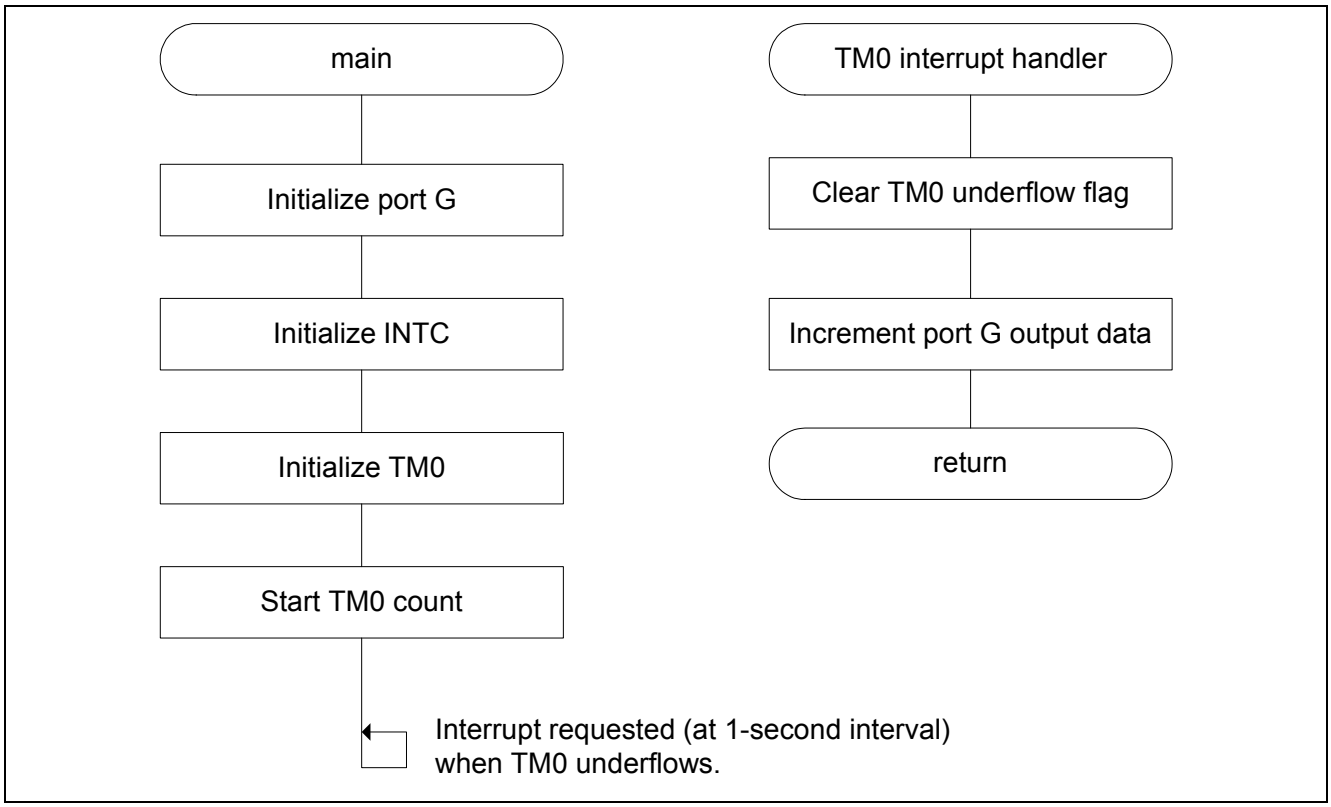


Figure 5.1 Outline of the Overall Process

Table 5.1 lists the section information for the sample code.

Table 5.1 Section Information

Address		Description
H'0000 0800	INTHandler	Exception handler
	VECTTBL	Reset vector table
	INTTBL	Interrupt vector table
	IntPRG	Interrupt functions program
H'0000 1800	PResetPRG	Reset program
H'0000 2000	P	Program area
	C	Constant area
	C$BSEC	Section B initialization table
	C$DSEC	Section D initialization table
	D	Initialization data area (ROM)
H'A000 0000	RSTHandler	Reset handler
H'E500 E000	B	Uninitialized data area
	R	Initialized data area (RAM)
H'E501 1C00	S	Stack address area

5.2 File Composition

Table 5.2 lists the files containing the sample code.

Table 5.2 Files Used in the Sample Code

File Name	Outline	Remarks
dbsct.c	Section B and D initialization address tables	
env.inc	Register address definitions related to exception handling	
intprg.src	Interrupt functions program	
iodefine.h	Register definition for the SH7450 and SH7451 Group microcontrollers	
main.c	Function main program	
resetprg.c	Reset program	
stacksct.h	Stack size definition file	
typedefine.h	Data type definitions	
vect.inc	Vector definition file	
vecttbl.src	Vector table	
vhandler.src	Reset and interrupt handler programs	

5.3 Constants

No constants are used in the sample code.

5.4 Structure/Union List

No structures or unions are used other than those defined in the SH7450 and SH7451 Group microcontroller register definition file (iodefine.h). See the application note referenced in chapter 3 for information about the structures and unions in iodefine.h.

5.5 Variables

No global variables are used in the sample code.

5.6 Functions

Table 5.3 lists the functions.

Table 5.3 Functions

Function	Outline
main	The main C language function. It initializes port G and TM0, and starts TM0.
INT_TMU0_TUNI0	TM0 underflow interrupt function

5.7 Function Specifications

This section presents the specifications of the functions in the sample code.

main

Outline	The main C language function. It initializes port G and TM0, and starts TM0.
Header	None
Declaration	void main(void)
Description	• Initializes port G and TM0, and starts the TM0 counter count operation. • After starting the TM0 count operation, this function waits (in an infinite loop) for the TM0 underflow interrupt (TUNI0).
Argument	None
Returned value	None
Remarks	None

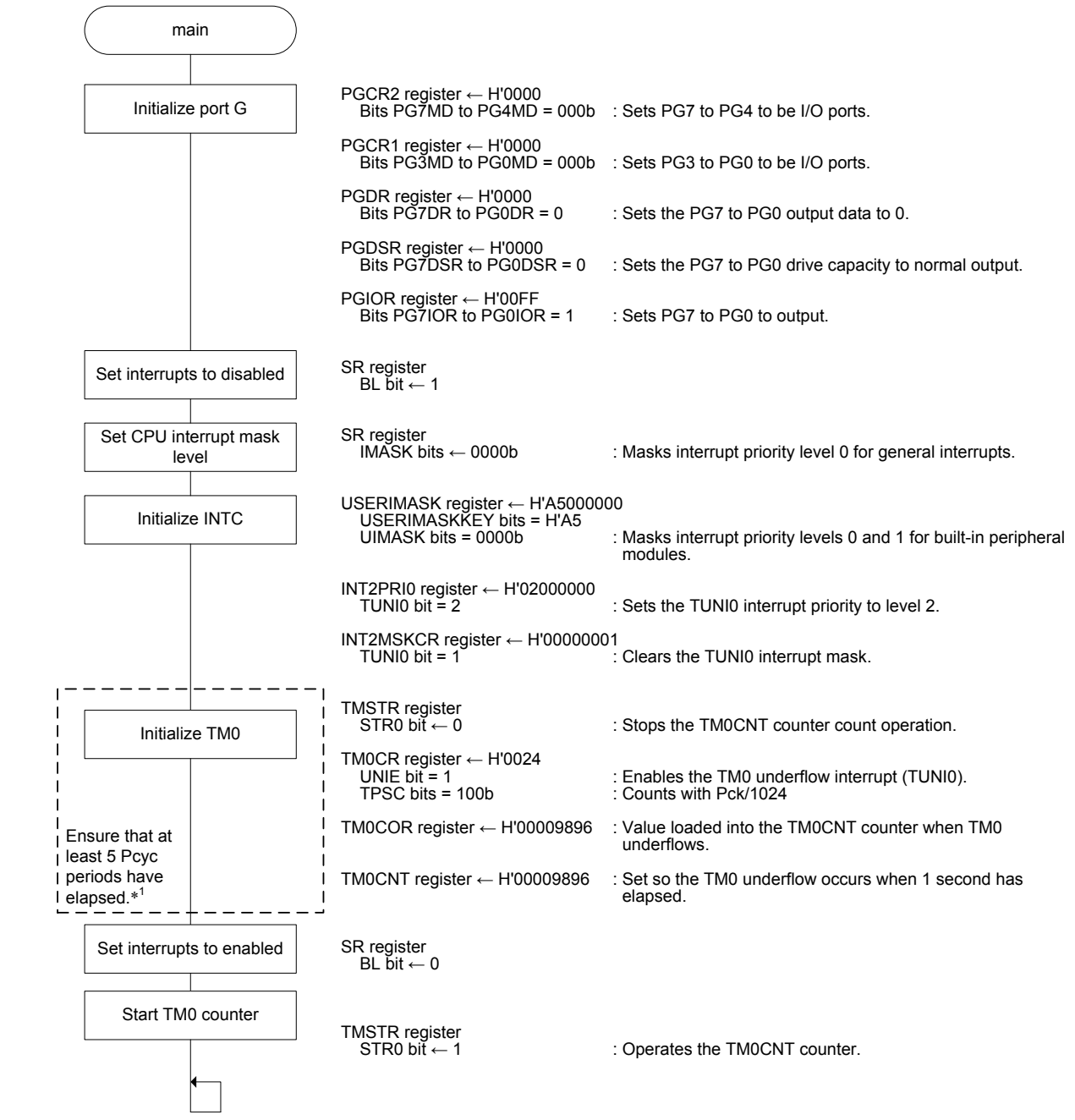
INT_TMU0_TUNI0

Outline	TM0 underflow interrupt function (TUNI0)
Header	None
Declaration	void INT_TMU0_TUNI0 (void)
Description	• Clears the TM0 underflow flag. • Increments the port G output data to update the LED pattern.
Argument	None
Returned value	None
Remarks	None

5.8 Flowcharts

5.8.1 Function main

Figure 5.2 shows the flowchart of the function main.



Note: *1. Applications must ensure a delay of at least the peripheral module priority determination time, which is 5 Pcyc (where Pcyc is the period of 1 cycle of the peripheral clock (Pck)) as the time until the changes to the INTC setting have been reflected in the priority judgment circuit before interrupts are enabled. Applications should ensure this 5 Pcyc period by performing dummy reads to the dummy access area (H'FFFF 5020 to H'FFFF 5023). In this sample code, the 5 Pcyc period is ensured by performing TM0 initialization after INTC initialization.

Figure 5.2 Function main

5.8.2 Function INT_TMU0_TUNI0

Figure 5.3 shows the flowchart of the function INT_TMU0_TUNI0.

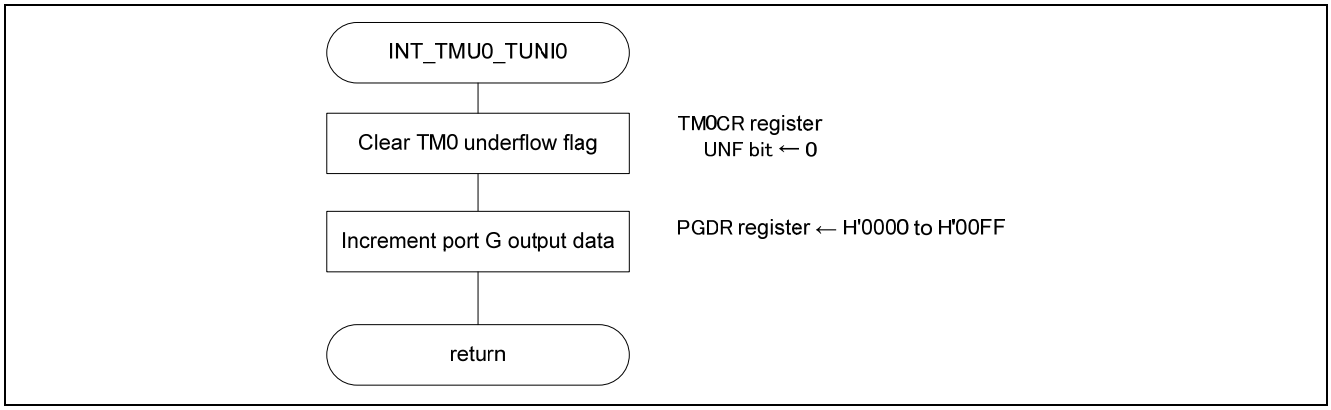


Figure 5.3 Function INT_TMU0_TUNI0

6. Sample Code

The sample code is available for download from the Renesas Electronics website.

7. Reference Documents

- SH7450 Group, SH7451 Group User's Manual: Hardware, Rev.1.10
  The latest version can be downloaded from the Renesas Electronics website.
- Technical Updates/Technical News
  The latest information can be downloaded from the Renesas Electronics website.
- C Compiler Manual
  SuperH Family C/C++ Compiler Package V.9.04 Release 00
  SuperH C/C++ Compiler Package V.9.04 User's Manual
  The latest version can be downloaded from the Renesas Electronics website.

Website and Support

Renesas Electronics Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry

Appendix A. Notes on Adding an Interrupt Handler

Table A.1 lists notes when adding an interrupt handler for a built-in peripheral module in the HEW program development environment.

Table A.1 Notes on Adding an Interrupt Handler for a Built-in Peripheral Module in the HEW Program Development Environment

Item	Details
Sets IMASK and UIMASK	• IMASK is set with the status register (SR) and UIMASK is set with the user interrupt mask level register (USERIMASK). • Interrupts with a level lower than that of the IMASK and UIMASK values are masked.
Sets the interrupt priority	• The interrupt priority levels are set with interrupt priority setting registers 0 to 12 (INT2PRI0 to INT2PRI12). • When interrupts are enabled, the priority level is set to 2 or higher (priority levels 0 and 1 correspond to the interrupt masked state). • Since the CPU interrupt is determined by 4 bits (levels 0 to 15), the LSB of the 5 bits of priority (levels 0 to 31) set with INT2PRIO0 to INT2PRI12 is discarded, converting the data to the 4 bits reported.
Clears the INTC interrupt mask setting	• The INTC interrupt mask used for built-in peripheral modules is cleared ith interrupt mask clear register 0 (INT2MSKCR) and interrupt mask clear register 1 (INT2MSKCR1).
Enables an interrupt for a built-in peripheral module	• The interrupts for each of the built-in peripheral modules are enabled individually. Example: For the TMU module, the TMU TM0 interrupt is enabled by setting the underflow interrupt control bit (UNIE) in the TM0 control register (TM0CR) to 1 (underflow interrupt (TUNI) enabled).
Registers an interrupt handler in the interrupt vector table of HEW	• Interrupt handlers for each factor are already registered in the interrupt vector table in the vecttbl.src file generated by HEW. • If an interrupt handler name is to be used without change, change the content of the interrupt handler registered in the vector table. • If a new interrupt handler name is to be used, after declaring the new interrupt handler as an external reference (import), register the new function in the vector table.
Clears an interrupt request	• The interrupt request is cleared in the interrupt handler. Example: For TM0, the TM0 interrupt request is cleared by setting the underflow flag (UNF) in the TM0 control register (TM0CR) to 0 (TM0CNT counter has not underflowed) in the interrupt handler.
Registers interrupt mask values in the interrupt mask table of HEW	• Interrupt priority levels are set as mask values in the interrupt mask table in the vecttbl.src file generated by HEW. • If the mask value in the interrupt mask table is not set to a value equal to or higher than the interrupt priority level, at the point multiple interrupts are enabled (BL bit in the SR register is set to 0), the CPU will accept the same interrupt request again and interrupt handler processing will be performed. If the interrupt request is cleared in the interrupt handler, it will not be possible to branch to the interrupt handler, since interrupt handler processing continues. Therefore, the interrupt handler processing will loop infinitely without clearing the interrupt flag.

Sample code to add an interrupt handler of a built-in peripheral module, TM0, for the HEW program development environment is shown below. In the sample code, comments are in green, and parts where modification is necessary are in red.

- Setting IMASK and UIMASK, setting interrupt priorities, clearing the INTC interrupt mask setting, and clearing the interrupt mask settings for each built-in peripheral module

main.c

```
#include <machine.h>
#include "iodefine.h"

void main(void)
{
     ── <Omitted> ──
     set_cr((int)((unsigned int)get_cr()|0x10000000U)); ← BL = 1 (Interrupt requests
                                                          are masked)
     set_cr((int)((unsigned int)get_cr()&0xFFFFFF0FU)); ← IMASK setting
     INTC.USERIMASK.LONG = 0xA5000000UL; ← UIMASK setting
     INTC.INT2PRI0.LONG = 0x02000000UL; ← Interrupt priority setting
     INTC.INT2MSKCR.LONG = 0x00000001UL; ← Clears the INTC interrupt mask setting
     ── <Omitted> ──
     TMU.TM0CR.WORD = 0x0024U; ← Clears the interrupt mask settings for the built-in
                                 peripheral modules
     ── <Omitted> ──
     set_cr((int)((unsigned int)get_cr()&0xEFFFFFFFU)); ← BL = 0 (Interrupt requests
                                                          are not masked)
     ...
}
```

- Registering interrupt mask value in the HEW interrupt mask table

vecttbl.src

```
_RESET_Vectors:
;<<VECTOR DATA START (POWER ON RESET)>>
        ;H'000 Power On Reset (H-UDI RESET)
        .data.l     _PowerON_Reset
;<<VECTOR DATA END (POWER ON RESET)>>
; Reserved
        .datab.l        8,H'00000000

        .section    INTTBL,data
        .export     _INT_Vectors

         ── <Omitted> ──
        .export     _INT_MASK
_INT_MASK:
        ── <Omitted> ──
        ;H'580 TMU0
        .data. b        H'10 ← Sets the interrupt priority level to the mask value
        ...                    in the mask table. (The upper 4 bits of the mask
                               value (The upper 4 bits of the mask value are set to
                               a value converted to 4 bits by discarding the LSB of
                               the TM0 interrupt priority (5 bits) set in the
                               INT2PRI0 register. During TM0 interrupt handler
                               processing, IMASK is set to upper 4 bits of the mask
                               value set here. In the example of changes to main.c
                               above, the TM0 mask value is set to H'10 since the
                               TM0 interrupt priority is set to H'02 with the
                               INT2PRI0 register.)
```

- Registering interrupt functions in the HEW interrupt vector table (when the interrupt function name is used as is)

vecttbl.src

```
_RESET_Vectors:
;<<VECTOR DATA START (POWER ON RESET)>>
        ;H'000 Power On Reset (H-UDI RESET)
        .data.l     _PowerON_Reset
;<<VECTOR DATA END (POWER ON RESET)>>
; Reserved
        .datab.l        8,H'00000000


        .section    INTTBL,data
        .export     _INT_Vectors

_INT_Vectors:
        — <Omitted> —
        ;H'580 TMU0
        .data.l_INT_TMU0_TUNI0 ← TM0 interrupt function
...
```

- Coding the interrupt function in assembly language (when the interrupt function name is used as is)

intprg.src

```
    .include    "vect.inc"
    .section    IntPRG, code

    ;H'040 Data TLB miss exception(read)
_INT_TLB_MISS_READ_EXP
     — <Omitted> —
    ;H'580 TMU0
_INT_TMU0_TUNI0
    — <Omitted> — ← In assembly language, code the clearing of the interrupt
                    request and any other processing to be performed.
     RTS
    — <Omitted> —
    ;H'5A0 TMU1
...
```

- Coding the interrupt function in C or C++ (when the interrupt function name is used as is)

intprg.src

```
    .include    "vect.inc"
    .section    IntPRG, code

    ;H'040 Data TLB miss exception(read)
_INT_TLB_MISS_READ_EXP:
...
    — <Omitted> —
    ;H'580 TMU0
;_INT_TMU0_TUNI0 ← Either comment out with semicolons or delete the TM0 interrupt
                  function.
...
```

main.c

```
void INT_TMU0_TUNI0(void) ← Create a TM0 interrupt function.
{
    — <Omitted> — ← In C or C++, code the clearing of the interrupt request and any
                    other processing to be performed.
}
```

- Registering interrupt functions in the HEW interrupt vector table (when a new interrupt function name is used)

vecttbl.src

```
_RESET_Vectors:
;<<VECTOR DATA START (POWER ON RESET)>>
        ;H'000 Power On Reset (H-UDI RESET)
        .data.l     _PowerON_Reset
;<<VECTOR DATA END (POWER ON RESET)>>
; Reserved
        .datab.l        8,H'00000000


        .section    INTTBL,data
        .export     _INT_Vectors
        .import     _NEW_INT_TMU0_TUNI0 ← External reference declaration for the new
                                          TM0 interrupt function

_INT_Vectors:
     — <Omitted> —
        ;H'580 TMU0
;        .data.l    _INT_TMU0_TUNI0 ← Either comment out with semicolons or delete
                                      the existing TM0 interrupt function.
         .data.l    _NEW_INT_TMU0_TUNI0 ← Register the new TM0 interrupt function.
...
```

- Coding the interrupt function in assembly language (when a new interrupt function name is used)

intprg.src

```
    .include    "vect.inc"
    .section    IntPRG, code

    ;H'040 Data TLB miss exception(read)
_INT_TLB_MISS_READ_EXP:

    — <Omitted> —
    ;H'580 TMU0
;_INT_TMU0_TUNI0 ← Either comment out with semicolons or delete the existing TM0
                   interrupt function.
_NEW_INT_TMU0_TUNI0 ← Create the new TM0 interrupt function.
    — <Omitted> — ← In assembly language, code the clearing of the interrupt
                    request and any other processing to be performed.
    RTS
    — <Omitted> —
    ;H'5A0 TMU1
...
```

- Coding the interrupt function in C or C++ (when a new interrupt function name is used)

main.c

```
void NEW_INT_TMU0_TUNI0 (void) ← Create the new TM0 interrupt function.
{
     — <Omitted> — ← In C or C++ language, code the clearing of the interrupt request and any other
                     processing be performed.
}
```

Appendix B. Interrupt Request Mask Operation

Table B.1 and figure B.1 present an example of mask operation for the TM0 underflow interrupt when UIMASK is 1 and IMASK is 2. The numbers (1) to (5) correspond in table B.1 and figure B.1. Note that bits in registers are noted as <register name>.<bit name> in table B.1 and figure B.1.

Table B.1 Example of Interrupt Mask Register Settings

	TMU	INTC			CPU
	TM0CR.UNIE	INT2MSKR.TUNI0	INT2PRI0.TUNI0	USERIMASK.UIMASK	SR.IMASK
(1)	0	0	00000b to 11111b	0001b	0010b
		1	00000b to 11111b		
(2)	1	0	00000b to 00011b		
(3)			00100b to 00101b		
(4)			00110b to 11111b		
(5)		1	00000b to 11111b		

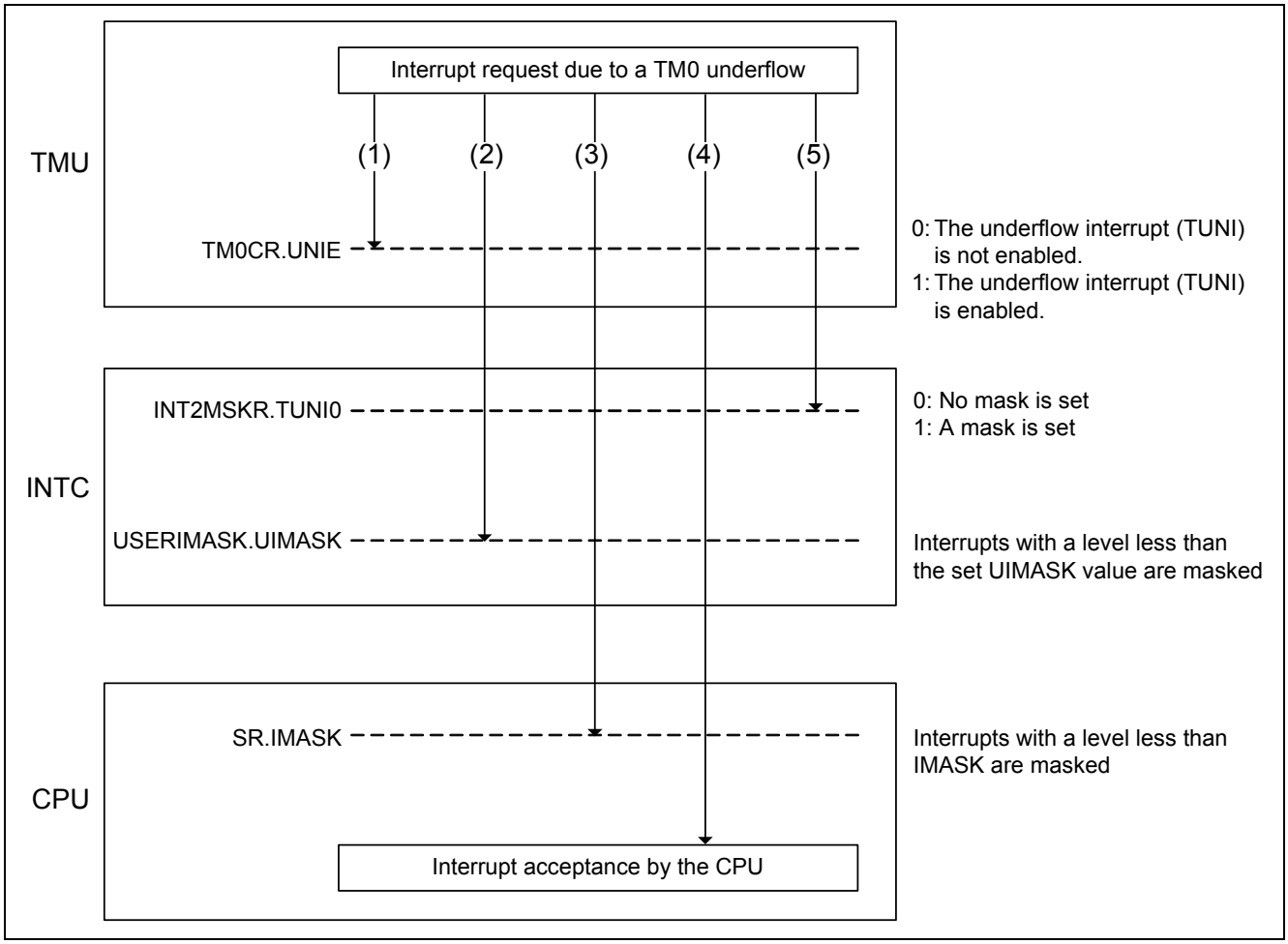


Figure B.1 Interrupt Mask Operation Example

(1) Since the TM0CR register UNIE bit is 0 (underflow interrupts (TUNI) not enabled), TMU internal interrupt requests are masked and interrupt requests are not output to the INTC. Therefore the INTC and the CPU will not detect TMU interrupt requests.

(2) Interrupt requests are output from TMU to the INTC. However, since the TM0 interrupt priority*[1] is less than the USERIMASK register UIMASK value, the INTC will mask the TMU interrupt requests and interrupt requests will not be output to the CPU. Therefore the CPU will not detect TMU interrupt requests.

(3) Since the TM0 interrupt priority is greater than UIMASK, interrupt requests will be output from TMU through the INTC to the CPU. However, since the TM0 interrupt priority*[1] is less than the SR register IMASK value, the CPU will mask TMU interrupt requests and TMU interrupt requests will not be detected.

(4) Since the TM0 interrupt priority is greater than the UIMASK value and greater than the IMASK value, interrupt requests will be output from TMU through the INTC to the CPU. The CPU will detect and accept the interrupt request.

(5) Interrupt requests are output from TMU to the INTC. However, since the INT2MSKR register TUNI0 bit is 1 (TM0 interrupt mask set), the INTC will mask the TMU interrupt requests and interrupt requests will not be output to the CPU. Therefore the CPU will not detect TMU interrupt requests.

Note: *1. Since the IMASK and UIMASK values are 4 bits, the LSB of the built-in peripheral module interrupt priorities (which are 5 bits) is discarded to create 4-bit values.

Appendix C. Multiple Interrupt Operation

This appendix describes the operation of multiple interrupts in the HEW program development environment using TMU channel 0 (TM0) and TMU channel 1 (TM1) as an example. This is described as an example where IMASK, UIMASK, and BL are set to 0 in normal processing.

Multiple Interrupts Example 1

This example shows how the higher priority interrupt TM1 occurs during TM0 interrupt function handling.

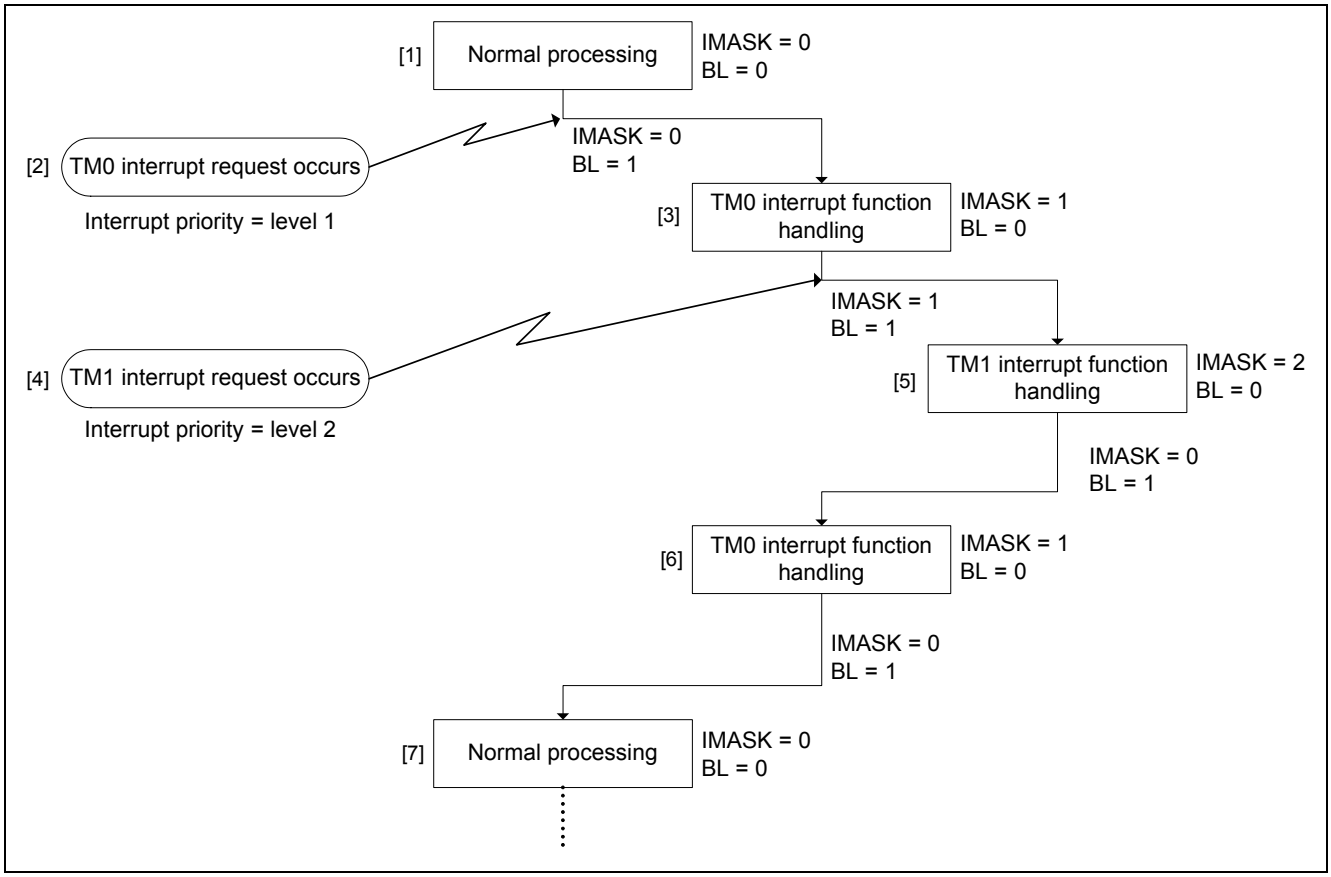


Figure C.1 Processing Flowchart of Multiple Interrupts (Example 1)

[1] During normal processing, it is possible to accept interrupts with an interrupt priority level other than 0 since IMASK is 0 (level 0 masked) and BL is 0 (interrupt requests not masked).

[2] Since a TM0 interrupt request (level 1) occurred, normal processing is interrupted and the CPU transitions to interrupt handling. First, in hardware processing, BL is set to 1 (interrupt requests masked). Then in software processing, the registers used in normal processing are saved on the stack. IMASK is set to 1, which is the TM0 interrupt priority, and BL is set to 0, and the software branches to TM0 interrupt function handling.

[3] During TM0 interrupt function handling, since IMASK is set to 1 (levels 1 or lower masked) and BL is set to 0, interrupts with an interrupt priority with a level of 2 or higher can be accepted.

[4] Since a TM1 interrupt request (level 2) has occurred, TM0 interrupt function handling is interrupted and the CPU transitions to TM1 interrupt handling. First, in hardware processing, BL is set to 1. Then in software processing, the registers used in TM0 interrupt function handling are saved on the stack. IMASK is set to 2, which is the TM1 interrupt priority, BL is set to 0, and the software branches to TM1 interrupt function handling.

[5] During TM1 interrupt function handling, since IMASK is set to 2 (levels 2 or lower masked) and BL is set to 0, interrupts with an interrupt priority with a level of 3 or higher can be accepted.
When TM1 interrupt function handling is complete, after IMASK is set to 0 and BL to 1 in software processing, the state prior to the interrupt (the registers used by the TM0 interrupt function hander) is restored and the interrupted processing is restarted.

[6] TM0 interrupt function handling restarts. IMASK and BL are in their states prior to TM1 interrupt handling, that is, the same states as they were in at step [3]. When TM0 interrupt function handling is complete, after IMASK is set to 0 and BL to 1 in software processing, the state prior to the interrupt (the registers used in normal processing) is restored and the interrupted processing restarts.

[7] Normal processing restarts. IMASK and BL are in their states prior to the occurrence of the TM0 interrupt, that is, the same states as they were in at step [1].

Multiple Interrupts Example 2

This example shows how the lower priority interrupt TM0 occurs during TM1 interrupt function handling.

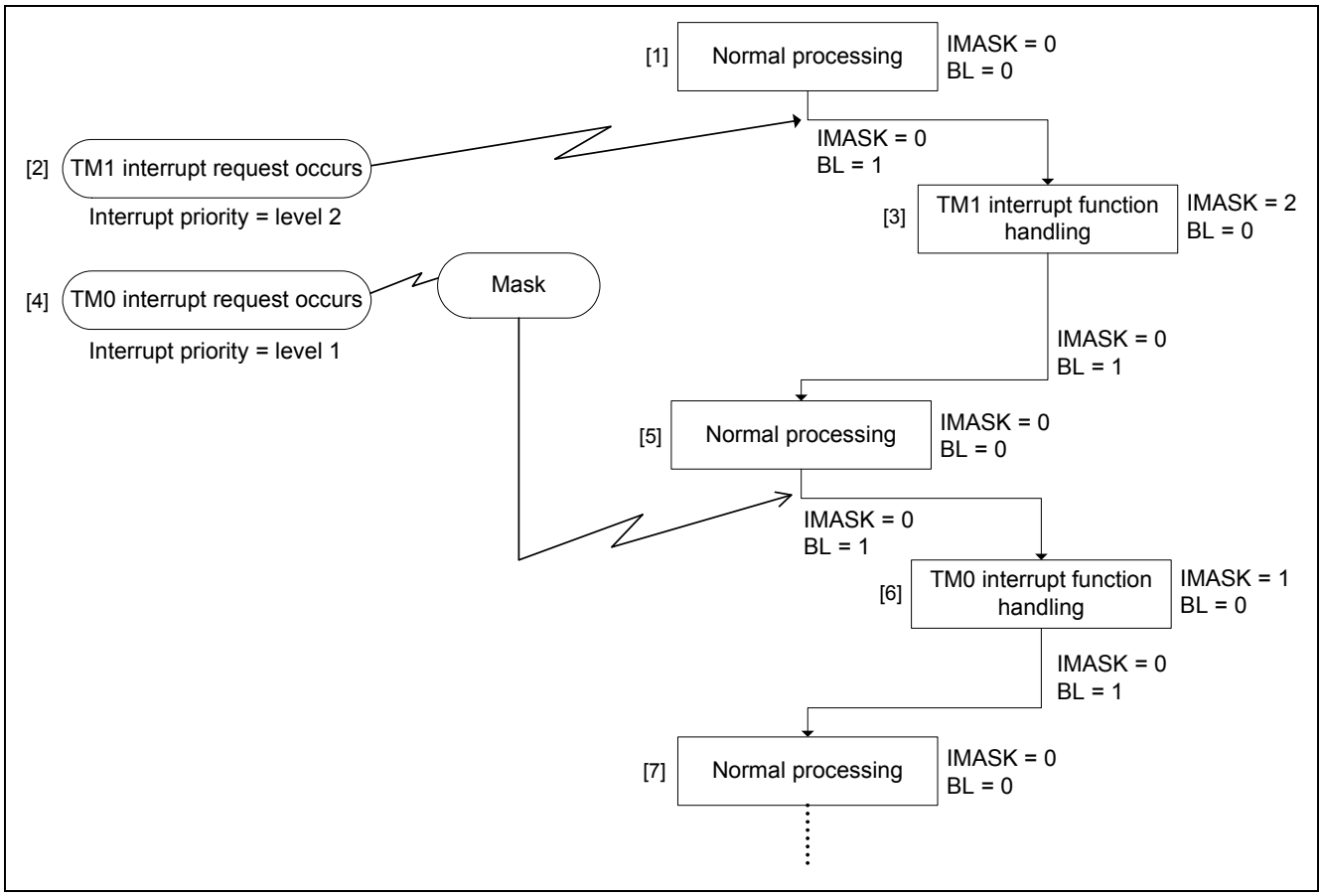


Figure C.2 Processing Flowchart for Multiple Interrupts (Example 2)

[1] During normal processing, it is possible to accept interrupts with an interrupt priority level other than 0 since IMASK is 0 (level 0 masked) and BL is 0 (interrupt requests not masked).

[2] Since a TM1 interrupt request (level 2) occurred, normal processing is interrupted and the CPU transitions to interrupt handling. First, in hardware processing, BL is set to 1 (interrupt requests masked). Then in software processing, the registers used in normal processing are saved on the stack. IMASK is set to 2, which is the TM1 interrupt priority, and BL is set to 0, and the software branches to TM1 interrupt function handling.

[3] During TM1 interrupt function handling, since IMASK is set to 2 (levels 2 or lower masked) and BL is set to 0, interrupts with an interrupt priority with a level of 3 or higher can be accepted.

[4] If a TM0 interrupt request (priority level 1) occurs, it will be masked because its interrupt priority level is lower than the IMASK value.
When TM1 interrupt function handling is complete, after IMASK is set to 0 and BL to 1 in software processing, the state prior to the interrupt (the registers used by the TM0 interrupt handler) is restored and the interrupted processing is restarted.

[5] Normal processing restarts. IMASK and BL are in their states prior to the occurrence of the TM1 interrupt, that is, the same states as they were in at step [1]. Since the interrupt priority level of the TM0 interrupt that was masked has become higher than the value of IMASK, the TM0 interrupt request will be accepted. Normal processing is interrupted and, in hardware, BL is set to 1. After that, in software, the registers used in normal processing are saved on the stack. IMASK is set to 1, which is the TM0 interrupt priority, and BL is set to 0, and the software branches to TM0 interrupt function handling.

[6] During TM0 interrupt function handling, since IMASK is set to 1 (levels 1 or lower masked) and BL is set to 0, interrupts with an interrupt priority with a level of 2 or higher can be accepted.
When TM0 interrupt function handling completes, after IMASK is set to 0 and BL to 1 in software processing, the state prior to the interrupt (the registers used in normal processing) is restored and the interrupted processing is restarted.

[7] Normal processing restarts. IMASK and BL are in their states prior to the occurrence of the TM0 interrupt, that is, the same states as they were in at step [1].

Multiple Interrupts Example 3

This example shows how a lower interrupt priority TM0 interrupt and a higher interrupt priority TM1 interrupt occur in contention (at the same time).

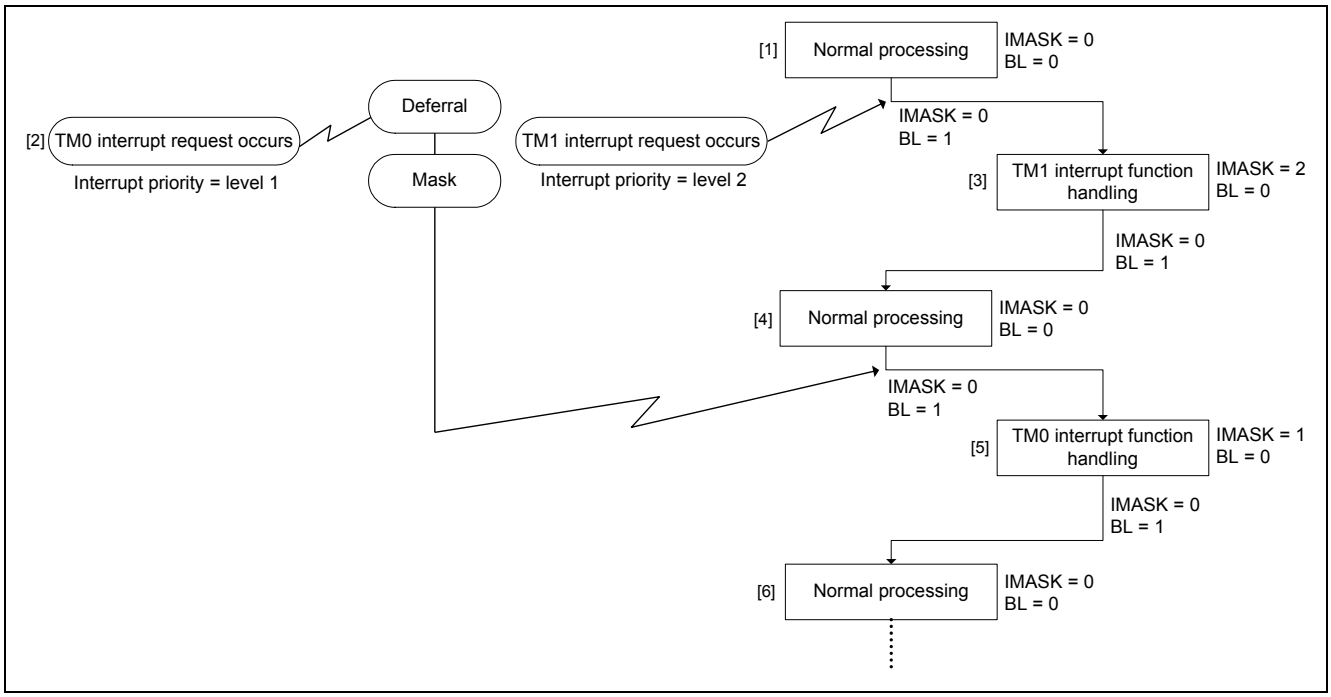


Figure C.3 Processing Flowchart for Multiple Interrupts (Example 3)

[1] During normal processing, it is possible to accept interrupts with an interrupt priority level other than 0 since IMASK is 0 (level 0 masked) and BL is 0 (interrupt requests not masked).

[2] Here, a TM0 interrupt request (level 1) and a TM1 interrupt request (level 2) occur at the same time and thus are in contention. The interrupt priorities are compared by the INTC and the TM1 interrupt request, which is the higher priority interrupt, is output. The TM0 interrupt request is deferred.
Normal processing is interrupted by the occurrence of the TM1 interrupt request. BL is set to 1 (interrupt requests masked) in hardware processing. Then in software processing, the registers used in normal processing are saved on the stack. IMASK is set to 2, which is the TM1 interrupt priority, and BL is set to 0, and the software branches to TM1 interrupt function handling.

[3] During TM1 interrupt function handling, since IMASK is set to 2 (levels 2 or lower masked) and BL is set to 0, interrupts with an interrupt priority with a level of 3 or higher can be accepted.
Since the interrupt priority level of the TM0 interrupt request is lower than the IMASK value, it is masked.

When TM1 interrupt function handling is complete, after IMASK is set to 0 and BL to 1 in software processing, the state prior to the interrupt (the registers used in normal processing) is restored and the interrupted processing restarts.

[4] Normal processing restarts. IMASK and BL are in their states prior to the occurrence of the TM1 interrupt, that is, the same states as they were in at step [1]. Since the interrupt priority level of the TM0 interrupt that was masked has become higher than the value of IMASK, the TM0 interrupt request will be accepted. Normal processing is interrupted and, in hardware processing, BL is set to 1. Then in software processing, the registers used in normal processing are saved on the stack. IMASK is set to 1, which is the TM0 interrupt priority, and BL is set to 0, and the software branches to TM0 interrupt function handling.

[5] During TM0 interrupt function handling, since IMASK is set to 1 (levels 1 or lower masked) and BL is set to 0, interrupts with an interrupt priority with a level of 2 or higher can be accepted.
When TM0 interrupt function handling is complete, after IMASK is set to 0 and BL to 1 in software processing, the state prior to the interrupt (the registers used in normal processing) is restored and the interrupted processing restarts.

[6] Normal processing restarts. IMASK and BL are in their states prior to the occurrence of the TM0 interrupt, that is, the same states as they were in at step [1].

Sample code for the multiple interrupt cases shown above for the HEW program development environment is shown below. In the sample code, comments are in green, and parts where a user needs to modify are in red.

- Setting IMASK and UIMASK, setting interrupt priorities, clearing the INTC interrupt mask setting, and clearing the interrupt mask settings for each built-in peripheral module

main.c

```
#include <machine.h>
#include "iodefine.h"

void main(void)
{
     ── <Omitted> ──
     set_cr((int)((unsigned int)get_cr()|0x10000000U)); ← BL = 1 (Interrupt requests
                                                          are masked)
     set_cr((int)((unsigned int)get_cr()&0xFFFFFF0FU)); ← IMASK setting
     INTC.USERIMASK.LONG = 0xA5000000UL; ← UIMASK setting

     INTC.INT2PRI0.LONG = 0x02040000UL; ← Sets the TM0 interrupt priority.
                                          Sets the TM1 interrupt priority.

     INTC.INT2MSKCR.LONG = 0x00000003UL; ← Clears the TM0 interrupt mask setting
                                           Clears the TM1 interrupt mask setting
     ── <Omitted> ──
     TMU.TM0CR.BIT.UNIE = 1U; ← TM0 underflow interrupt is enabled.
     TMU.TM1CR.BIT.UNIE = 1U; ← TM1 underflow interrupt is enabled.
     ── <Omitted> ──
     set_cr((int)((unsigned int)get_cr()&0xEFFFFFFFU)); ← BL = 0 (Interrupt requests are
                                                          not masked)
...
}
```

- Registering interrupt mask value in the HEW interrupt mask table

vecttbl.src

```
_RESET_Vectors:
;<<VECTOR DATA START (POWER ON RESET)>>
        ;H'000 Power On Reset (H-UDI RESET)
        .data.l      _PowerON_Reset
;<<VECTOR DATA END (POWER ON RESET)>>
; Reserved
        .datab.l        8,H'00000000


        .section    INTTBL,data
        .export     _INT_Vectors

        ── <Omitted> ──
        .export     _INT_MASK
_INT_MASK:
        ── <Omitted> ──
        ;H'580 TMU0
        .data.b      H'10 ← During TM0 interrupt function execution, interrupts with
                            an interrupt priority level of 1 or lower are masked.
        ;H'5A0 TMU1
        .data.b      H'20 ← During TM1 interrupt function execution, interrupts with
                            an interrupt priority level of 2 or lower are masked.
...
```

- Registering interrupt handlers in the HEW interrupt vector table

vecttbl.src

```
_RESET_Vectors:
;<<VECTOR DATA START (POWER ON RESET)>>
        ;H'000 Power On Reset (H-UDI RESET)
        .data.l      _PowerON_Reset
;<<VECTOR DATA END (POWER ON RESET)>>
; Reserved
        .datab.l        8,H'00000000


        .section    INTTBL,data
        .export     _INT_Vectors

_INT_Vectors:
     ── <Omitted> ──
        ;H'580 TMU0
        .data.l     _INT_TMU0_TUNI0 ← TM0 interrupt function
        ;H'5A0 TMU1
        .data.l     _INT_TMU1_TUNI1 ← TM1 interrupt function
...
```

- Coding the interrupt handler in C or C++

intprg.src

```
    .include    "vect.inc"
    .section    IntPRG, code

    ;H'040 Data TLB miss exception(read)
_INT_TLB_MISS_READ_EXP:
:
    — <Omitted> —
    ;H'580 TMU0
;_INT_TMU0_TUNI0 ← Either comment out with semicolons or delete the TM0 interrupt function.
    ;H'5A0 TMU1
;_INT_TMU1_TUNI1 ← Either comment out with semicolons or delete the TM1 interrupt function.
...
```

main.c

```
void INT_TMU0_TUNI0(void) ← Creates the TM0 interrupt function.
{
    — <Omitted> — ← In C or C++ language, code the clearing of the interrupt request and any other
                    processing to be performed.
}


void INT_TMU1_TUNI1(void) ← Create a TM1 interrupt function.
{
    — <Omitted> — ← In C or C++ language, code the clearing of the interrupt request and any other
                    processing to be performed.
}
```

Revision History	SH7450 Group, SH7451 Group Timer Unit (TMU) Interrupt Sample Code

Rev.	Date	Description	
		Page	Summary
1.00	Mar. 16, 2012	—	First edition issued

General Precautions in the Handling of MPU/MCU Products

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

1. Handling of Unused Pins
   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.
   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.
2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.
3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.
4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.
5. Differences between Products
   Before changing from one product to another, i.e. to one with a different type number, confirm that the change will not lead to problems.
   — The characteristics of MPU/MCU in the same group but having different type numbers may differ because of the differences in internal memory capacity and layout pattern. When changing to products of different type numbers, implement a system-evaluation test for each of the products.

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.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

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

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141

Colophon 1.1