RH850/U2A-EVA Group

Startup Application Note

Introduction

This application note describes the Startup processing on RH850/U2A (hereinafter referred to as U2A) series in the automotive single chip microcomputer by Renesas Electronics.

Aim of this document and software is to provide supplemental information for the function on RH850/U2A. It is not intended to implement in the design for mass production.

There is no guarantee to update in this document and software to reflect the latest manual, errata, technical update and development environment. You are fully responsible for the incorporation or any other use of the information of this document in the design of your product or system, and please refer to latest manual, errata, technical update and development environment.

Target device

- RH850/U2A-EVA Group
  - ➢ RH850/U2A16
  - ➢ RH850/U2A8
  - ➢ RH850/U2A6

Target integrated development environment

CS+ (by Renesas Electronics Corporation)

Version	: V.8.07.00
Device file	: R7F702300.DVF : R7F702301.DVF : R7F702302.DVF

MULTI (by Green Hills Software)

Product	: IDE for V800
Version	: 2021.1.5(v 7.1.6)
Target	: V800/RH850
Device file	: R7F702300.DVF : R7F702301.DVF : R7F702302.DVF
EXEC file	: ExecG3G4_ V10500

In the case of MULTI, some directory names depend on the version, etc. of MULTI. If the version etc. is different, please replace it accordingly.

Reference Document

RH850/U2A-EVA Group User's Manual: Hardware

The Hardware User's Manual provides information about functional and electrical behavior of the device. At the release time of this application note the following manual version available:RH850/U2A-EVA User's Manual(Rev.1.20): R01UH0864EJ0120

Contents

1. Overview

Startup processing is the processing from Power-on till calling user applications.

This application note is for the startup processing in the integrated development environment, CS+ [Note 1] by Renesas Electronics Corporation and MULTI by Green Hills Software.

1.1 Note

RH850/U2A series contains multi cores and number of CPU depend on the products.

RH850/U2A16 implement 4 CPUs and RH850/U2A8 and U2A6 implement 2 CPUs. CPU2 (PE2), CPU3 (PE3) are not implemented in RH850/U2A8 and U2A6.

This application note is for the startup processing on RH850/U2A16. In the case of RH850/U2A8 or U2A6, CPU2 (PE2), CPU3 (PE3) processing is not applicable.

[Note 1] Former known as CubeSuite+.

2. CS+

This chapter describes each Startup processing in using CS+.

2.1 Startup Related Files

Table 2.1 indicates the list of relevant files to Startup.

Table 2.1 Startup Related File List in CS+

#	File	Directory	Description
1	boot.asm	Project root /	Startup routine for boot loader
2	cstart0～3.asm	Project root /PE0～3/	Startup routine for user applications（each PE）
3	vecttbl0～3.asm	Project root /PE0～3/	Vector table（each PE）
4	main0～3.c	Project root/PE0～3/	Main processing（each PE）

In this project, PE0 and PE1 refer to vecttbl0.asm, and PE2 and PE3 refer to vecttbl2.asm. Vecttbl1.asm and vecttbl3.asm are unused. In the case of dividing the vector table referred by each PE, change the reference address of the Reset Vector PEx register.

2.2 Setting Sections

Table 2.2 and Table 2.3 indicates the examples of main sections related to Startup.

Table 2.2 Startup Related File List in CS+(Part.1)

#	Project	Section	Allocate data
1	Common Part (Main Project)	RESET_PE0～3	RESET vector
2		EIINTTBL_PE0～3	EI level interrupt vector table for table reference method
3		.text	Program code (boot.asm)
4	PE0 （Sub Project）	.const	Read only data
5		.INIT_DSEC.const	Initialization table for sections with initial value
6		.INIT_BSEC.const	Initialization table for sections without initial value
7		.text.cmn	Passing data between boot.asm and cstat0.asm
8		.text	Program code (cstart0.asm/main0.c)
9		.data	Data with initial value (ROM)
10		.data.R	Data with initial value (RAM)
11		.bss	Data without initial value
12		.stack.bss	Stack
13	PE1 （Sub Project）	.const	Read only data
14		.INIT_DSEC.const	Initialization table for sections with initial value
15		.INIT_BSEC.const	Initialization table for sections without initial value
16		.text.cmn	Passing data between boot.asm and cstat1.asm
17		.text	Program code (cstart1.asm/main1.c)
18		.data	Data with initial value (ROM)
19		.data.R	Data with initial value (RAM)
20		.bss	Data without initial value
21		.stack.bss	Stack

Table 2.3 Startup Related Sections in CS+ (Part 2)

#	Project	Section	Allocate data
22	PE2	.const	Read only data
23	（Sub Project）	.INIT_DSEC.const	Initialization table for sections with initial value
24		.INIT_BSEC.const	Initialization table for sections without initial value
25		.text.cmn	Passing data between boot.asm and cstat2.asm.
26		.text	Program code (cstart2.asm/main2.c)
27		.data	Data with initial value (ROM)
28		.data.R	Data with initial value (RAM)
29		.bss	Data without initial value
30		.stack.bss	Stack
31	PE3	.const	Read only data
32	（Sub Project）	.INIT_DSEC.const	Initialization table for sections with initial value
33		.INIT_BSEC.const	Initialization table for sections without initial value
34		.text.cmn	Passing data between boot.asm and cstat3.asm.
35		.text	Program code (cstart3.asm/main3.c)
36		.data	Data with initial value (ROM)
37		.data.R	Data with initial value (RAM)
38		.bss	Data without initial value
39		.stack.bss	Stack

Depending on each device or project, the name may be different, some sections may not be needed, or other sections may be needed. For details of the sections, refer to the user manuals of CS+ and devices.

2.2.1 Section Setting Method

The following describes a method of specifying or adding a section in a program in the assembly and C languages.

・Assembly language

.section SectionName SectionType (These spaces are essential.)

Section Name: Specify the name of section.

Section Type: Specify a relocation attribute.

･C language

.pragma section [Section Type] [Section Name]

Section Type: Specify the relocation attribute.

Section Name: Specify the name of section.

Main relocation attributes are indicated in Table 2.4 below. For the other relocation attributes, refer to the user's manual for CS+.

Table 2.4 Main Relocation Attributes

#	Relocation attributes	Default section	Notes
1	text	.text	Program code is located in .text section.
2	r0_disp32	.data	The data with initial value is located in .data section.
3	const	.const	The read only data is located in .const section.
4	default	-	All the settings are cleared to return to the default section.

The created section can be referred with "xxx.relocation attributes".

For example, if the section name is newsection and relocation attribute is text, it will be newsection.text.

2.3 Startup Processing

2.3.1 Definition of the conditional assembly control instructions

Table 2.5 and Table 2.6 indicates the definition of the instructions for conditional assembly controls. These instructions for conditional assembly controls are defined in boot.asm

Table 2.5 Definition List of the Conditional Assembly Control Instructions (Part.1)

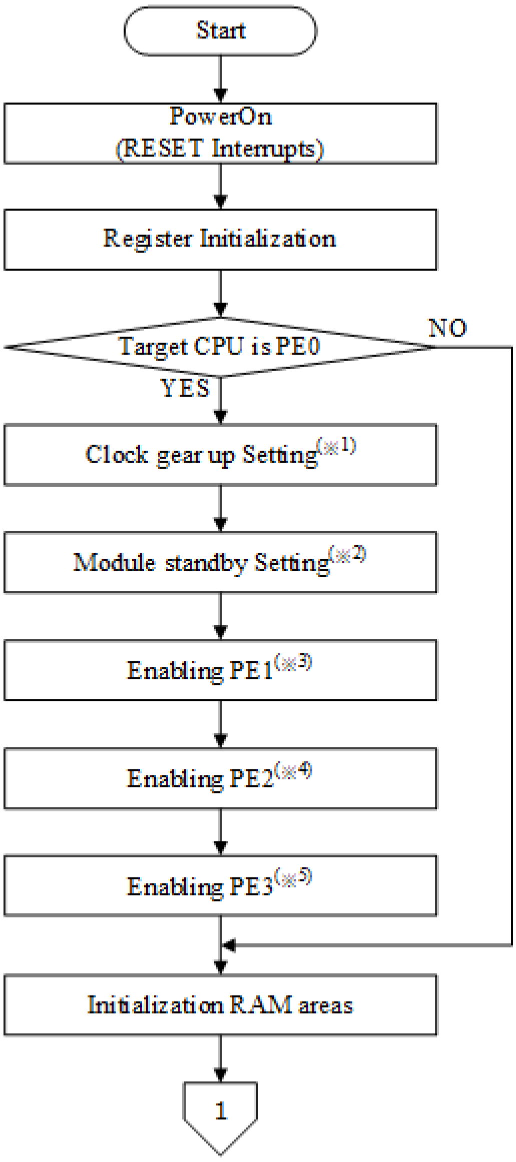
#	Definition name	Value	Description
1	ENABLE_PE1_BY_PE0		Define PE1 as enable/disable. Enabling PE is to be executed in PE0. The default value is 1(Enable PE1).
		0	Disables PE1
		1	Enables PE1
2	ENABLE_PE2_BY_PE0		Define PE2 as enable/disable. Enabling PE is to be executed in PE0. The default value is 1(Enable PE2).
		0	Disables PE2
		1	Enables PE2
3	ENABLE_PE3_BY_PE0		Define PE3 as enable/disable. Enabling PE is to be executed in PE0. The default value is 1(Enable PE3).
		0	Disables PE3
		1	Enables PE3
4	USE_TABLE_REFERENCE_METHOD		As an interrupt vector method, define table reference method as used/unused. The default value is 1(Use a table reference method).
		0	Does not use a table reference method as an interrupt vectoring method.
		1	Uses a table reference method as an interrupt vectoring method.
5	ENABLE_CLOCK_GEARUP		Define clock gearup as executed/unexecuted. Clock gearup is to be executed in PE0. The default value is 1(Clock gearup is executed).
		0	Clock Gearup is not executed
		1	Clock Gearup is executed

Table 2.6 Definition List of the Conditional Assembly Control Instructions (Part.2)

#	Definition name	Value	Description
6	ENABLE_MODULE_STANDBY_SET		Define module standby settings as executed/unexecuted (Enable/disable clock supply to each function to be used) The default value is 0(Module standby setting is not executed).
		0	Module standby setting is not executed.
		1	Module standby setting is executed.

2.3.2 Overall Flow

Figure 2.1 and Figure 2.2 indicates the overall flow for the startup process in CS+[Note1].



※1：Only if 1 is set to ENABLE_CLOCK_GEARUP.

※2：Only if 1 is set to ENABLE_MODULE_STANDBY.

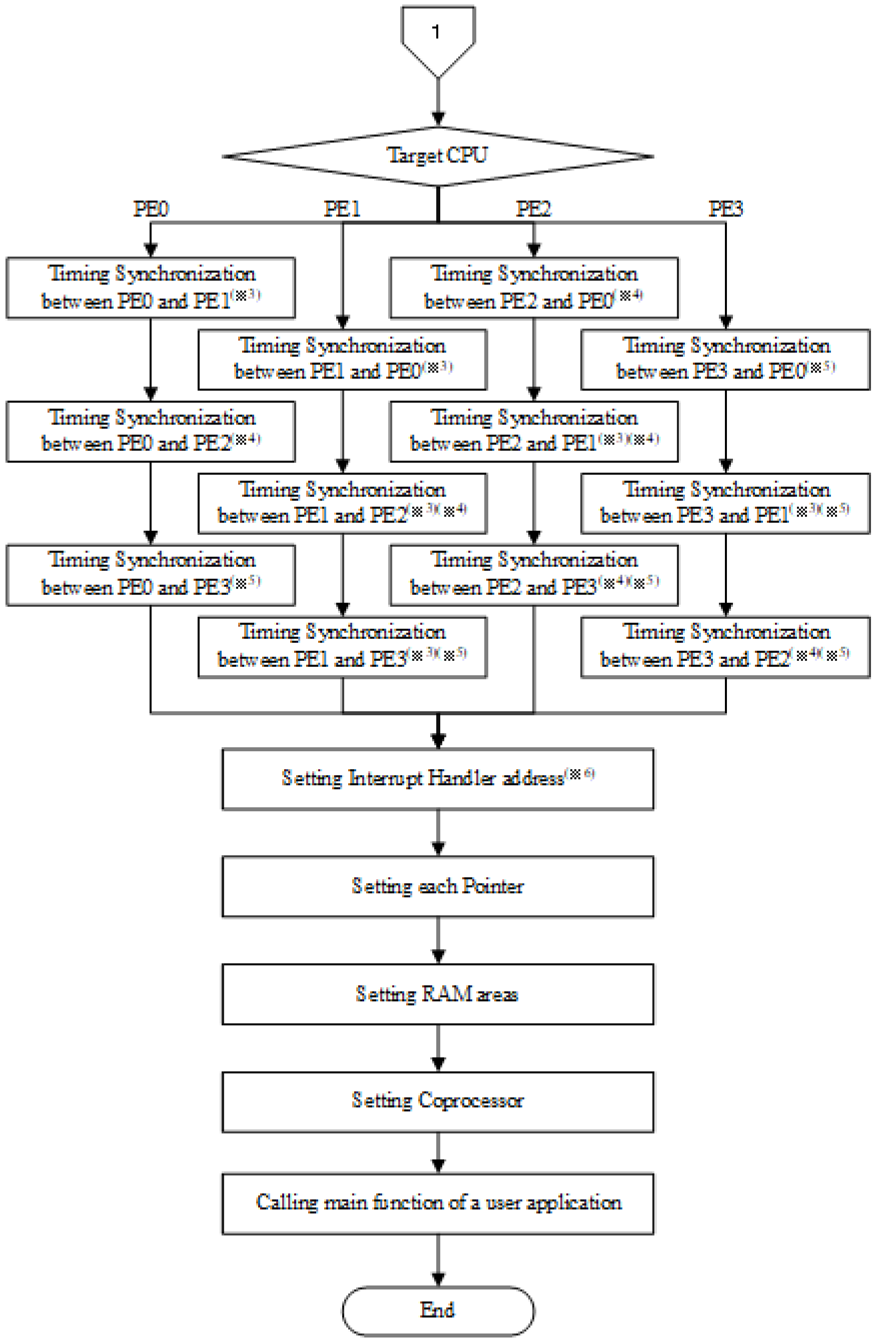
※3：Only if 1 is set to ENABLE_PE1_BY_PE0.

※4：Only if 1 is set to ENABLE_PE2_BY_PE0.

※5：Only if 1 is set to ENABLE_PE3_BY_PE0.

Figure 2.1 Overall Flow of Startup in CS+ (Part.1)

[Note1] Initial state (initially stopped or active state) of initially stopped core is selected by debug tool. For the details of debugging for initially stopped core, please refer to User’s manual and Additional documents for emulator, and User’s manual and help for the emulator debugger.



※3 : Only if 1 is set to ENABLE_PE1_BY_PE0.

※4 : Only if 1 is set to ENABLE_PE2_BY_PE0.

※5 : Only if 1 is set to ENABLE_PE3_BY_PE0.

※6 : Only if 1 is set to USE_TABLE_REFERENCE_METHOD.

Figure 2.2 Overall Flow of Startup in CS+ (Part.2)

2.3.3 Process Routines

Table 2.7, Table 2.8 and Table 2.9 indicates the routines where each process has been implemented.

Table 2.7 Process Implemented-Routine List in CS+ (Part.1)

#	Process Name	Routine Name	Implementation files	Notes
1	Power on (Reset interrupts)	-	vecttbl0.asm vecttbl1.asm vecttbl2.asm vecttbl3.asm	Implemented in the interrupt vector table.
2	Register initialization	__start	boot.asm	
3	Setting clock gearup	_clock_gearup	boot.asm	To be processed only when the running PE is PE0. (*1)
4	Setting module standby set	_module_standby_set	boot.asm	To be processed only when the running PE is PE0 (※2)
5	Enabling PE1~3	__start_PE0	boot.asm	To be processed only when the running PE is PE0(※3) (※4) (※5)
6	Initializing RAM areas	_hdwinit_PE0 _hdwinit_PE1 _hdwinit_PE2 _hdwinit_PE3	boot.asm	To be processed with "__ hdwinit_PE0" when the running PE is PE0, with "__ hdwinit_PE1" when the running PE is PE1, with "__ hdwinit_PE2" when the running PE is PE2, and with "__ hdwinit_PE3" when the running PE is PE3. (※3) (※4) (※5)
7	Timing synchronization (PE0 ~PE3)	_hdwinit_PE0 _hdwinit_PE1 _hdwinit_PE2 _hdwinit_PE3	boot.asm	To be processed with "__ hdwinit_PE0" when the running PE is PE0, with "__ hdwinit_PE1" when the running PE is PE1, with "__ hdwinit_PE2" when the running PE is PE2, and with "__ hdwinit_PE3" when the running PE is PE3. (※3) (※4) (※5)

※1 : The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_CLOCK_GEARUP is 0, setting clock gearup will not be executed.

※2 : The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_MODULE_STANDBY_SET is 0, Setting module standby set will not be processed.

※3 : The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE1_BY_PE0 is 0, PE1 will not be processed.

※4 : The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE2_BY_PE0 is 0, PE2 will not be processed.

※5 : The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE3_BY_PE0 is 0, PE3 will not be processed.

Table 2.8 Process Implemented-Routine List in CS+ (Part.2)

#	Process Name	Routine Name	Implementation files	Notes
8	Setting interrupt handler address	_init_eiint	boot.asm	
9	Setting each pointer	__start_pm0 __start_pm1 __start_pm2 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	To be processed with "__start_pm0" when the running PE is PE0, with "__start_pm1" when the running PE is PE1, with "__start_pm2" when the running PE is PE2, and with "__start_pm3" when the running PE is PE3.(※3) (※4) (※5)
10	Initializing RAM areas	__start_pm0 __start_pm1 __start_pm2 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	To be processed with "__start_pm0" when the running PE is PE0, with "__start_pm1" when the running PE is PE1, with "__start_pm2" when the running PE is PE2, and with "__start_pm3" when the running PE is PE3.(※3) (※4) (※5)
11	Setting coprocessor	__start_pm0 __start_pm1 __start_pm2 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	To be processed with "__start_pm0" when the running PE is PE0, with "__start_pm1" when the running PE is PE1, with "__start_pm2" when the running PE is PE2, and with "__start_pm3" when the running PE is PE3.(※3) (※4) (※5)

※3：The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE1_BY_PE0 is 0, PE1 will not be processed.

※4：The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE2_BY_PE0 is 0, PE2 will not be processed.

※5：The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE3_BY_PE0 is 0, PE3 will not be processed.

Table 2.9 Process Implemented-Routine List in CS+(Part.3)

#	Process Name	Routine Name	Implementation files	Notes
12	Calling a main function of user application	__start_pm0 __start_pm1 __start_pm2 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	To be processed with "__start_pm0" when the running PE is PE0, with "__start_pm1" when the running PE is PE1, with "__start_pm2" when the running PE is PE2, and with "__start_pm3" when the running PE is PE3.(※3) (※4) (※5)

※3：The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE1_BY_PE0 is 0, PE1 will not be processed.

※4：The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE2_BY_PE0 is 0, PE2 will not be processed.

※5：The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE3_BY_PE0 is 0, PE3 will not be processed.

2.3.4 Details of Each Process

This section describes the details of each process.

If nothing is specified here, the same process will be executed in either of PE0～PE3[Note 1]

2.3.4.1 Power-On (RESET Interrupts)

On powering on, Program Counter becomes RESET Vector address (RESET section).

It jumps from the RESET Vector to __start routine.

When device is powered on, PE0 starts. In processing of PE0 (ref. Section 3.3.4.5), if ENABLE_PE1_BY_PE0 is 1, PE1 is activated, ENABLE_PE2_BY_PE0 is 1, PE2 is activated, and ENABLE_PE3_BY_PE0 is 1, PE3 is activated.

Figure 2.3 indicates program code examples.

```
.section  "RESET", text          ; RESET Vector
.align    512
jr32      __start
```

Figure 2.3 Example Program Code for Power-On: RESET Interrupts in CS+

[Note 1] To enable PE1, set 1 to ENABLE_PE1_BY_PE0.
To enable PE2, set 1 to ENABLE_PE2_BY_PE0.
To enable PE3, set 1 to ENABLE_PE3_BY_PE0.

2.3.4.2 Initializing Registers

Table 2.10 and Table 2.11 indicates registers for initialization. Since the setting value is an example, please initialize with an optimum value according to the system.

Table 2.10 Initialization Register List in CS+ (Part.1)

#	Resister type	Resister name	Setting value example	Description
1	Program registers	r1～r31	0	
2	Basic system registers	EIPC	0	
3		FEPC	0	
4		CTPC	0	
5		EIWR	0	
6		FEWR	0	
7		EBASE	0	
8		INTBP	0	
9		MEA	0	
10		MEI	0	
11		RBIP	0	
12		PSW	0x00010020	ID=1: Prohibit receiving EI level exceptions. CU0=1: Enable FPU
13	FPU system registers	FPSR	0x00220000	Value after Reset
14		FPEPC	0	
15		FPST	0	
16		FPCC	0	
17	MPU function system registers	MCA	0	
18		MCS	0	
19		MCR	0	
20		MPLA[Note1]	0	
21		MPUA[Note1]	0	
22		MPAT [Note1]	0	
23		MPID0	0	
24		MPID1	0	
25		MPID2	0	
26		MPID3	0	
27		MPID4	0	
28		MPID5	0	

[Note1] The registers of all the 32 MPU entries should be initialized. Set the index register MPIDX from 0 to 31, and initialize the corresponding MPLA, MPUA, MPAT registers.

Table 2.11 Initialization Register List in CS+ (Part.2)

#	Resister type	Resister name	Setting value example	Description
29	MPU function system registers	MPID6	0	
30		MPID7	0	
31		MCI	0	
32	Cache operation function registers	ICTAGL	0	
33		ICTAGH	0	
34		ICDATL	0	
35		ICDATH	0	
36		ICERR	0	
37	Virtualization support function system register	HVSB	0	
38	Guest Context Register	GMEIPC	0	
39		GMFEPC	0	
40		GMEBASE	0	
41		GMINTBP	0	
42		GMEIWR	0	
43		GMFEWR	0	
44		GMMEA	0	
45		GMMEI	0	

Figure 2.4 indicates a program code example.

```
    $nowarning
    mov     r0, r1
    $warning
    mov     r0, r2
(Omitting the middle part)
    ldsr    r0, 0, 0            ; SR0,0   EIPC
    ldsr    r0, 2, 0            ; SR2,0   FEPC
    ldsr    r0, 16, 0           ; SR16,0 CTPC
(Omitting the middle part)
    mov     0x00010020, r10
    ldsr    r10, 5, 0           ; SR5,0   PSW
(Omitting the rest)
```

Figure 2.4 Example Program Code for Register Initialization in CS+

2.3.4.3 Clock Gearup Settings

After starting PE0, change the system clock to PLL and execute clock gearup.

This process is to be executed only when all the following conditions are satisfied.

・ENABLE_CLOCK_GEARUP is 1.

・The running PE is PE0(PEID bit0:2(PEID)=0

・Main OSC and PLL are enabled (PLLS=0x00000003)

Figure 2.5 and Figure 2.6 indicate the program code examples for clock gearup as a reference.

```
.L.clock_gearup.0:
      ld.w     0xff980004[r0], r2       ; get PLLS
      andi     0x3, r2, r2
      cmp      0x3, r2
      bnz      .L.clock_gearup.0

      mov      0xA5A5A501, r2           ; set 0xA5A5A501 in CLKKCPROT1 for ・・・
      st.w     r2, 0xff980700[r0]       ; set CLKKCPROT1

      ld.w     0xff980120[r0], r2       ; get CLKD_PLLC
      ori      0x2, r2, r2              ; set 2 in CLKD_PLLC.PLLCLKDCSID ・・・
      mov      -0x6, r6
      and      r6, r2
      st.w     r2, 0xff980120[r0]       ; set CLKD_PLLC

.L.clock_gearup.1:
      ld.w     0xff980128[r0], r2       ; get CLKD_PLLS
      andi     0x2, r2, r0              ; confirm that the value of CLKD_ ・・・
      bz       .L.clock_gearup.1        ; if the CLKD_PLLS.PLLCLKD ・・・
(Omitting the middle part)
      ld.w     0xff980100[r0], r2       ; get CKSC_CPUC
      mov      -0x2, r6                 ; set 0 in CKSC_CPUC.CPUCLKSCSID ・・・
      and      r6, r2
      st.w     r2, 0xff980100[r0]       ; set CKSC_CPUC

.L.clock_gearup.4:
      ld.w     0xff980108[r0], r2       ; get CKSC_CPUS
```

Figure 2.5 Example Program Code for Clock Gearup Settings(CS+)(Part.1)

```
      andi     0x1, r2, r0              ; confirm that the value of CKSC_CPUS・・・
      bnz      L.clock_gearup.4         ; if the CKSC_CPUS.CPUCLKSACT is・・・
(Omitting the middle part)
      ld.w     0xff980120[r0], r2       ; get CLKD_PLLC
      ori      0x1, r2, r2              ; set 1 in PLLC.PLLCLKD. ・・・
      mov      -0x7, r6
      and      r6, r2
      st.w     r2, 0xff980120[r0]       ; set CLKD_PLLC

.L.clock_gearup.7:
      ld.w     0xff980128[r0], r2       ; get CLKD_PLLS
      andi     0x2, r2, r0              ; confirm that the value of CLKD_ ・・・
      bz       .L.clock_gearup.7        ; if the CLKD_PLLS.PLLCLKD ・・・
(Omitting the middle part)
      mov      0xA5A5A500, r2           ; set 0xA5A5A500 in CLKKCPROT1 for・・・
      st.w     r2, 0xff980700[r0]       ; set CLKKCPROT1
```

Figure 2.6 Example Program Code for Clock Gearup Settings(CS+)(Part.2)

2.3.4.4 Module Standby Settings

Set the module standby registers of the function to be used.

This process is to be executed only when all the following conditions are satisfied.

・ENABLE_MODULE_STANDBY_SET is 1

・The running PE is PE0(PEID bit0:2(PEID)=0

Figure 2.7 indicates the program code examples for module standby register settings (enabling clock supply) as a reference.

```
      mov      0xA5A5A501, r2           ; set 0xA5A5A501 in MSRKCPROT for ・・・
      st.w     r2, 0xFF981710[r0]       ; set MSRKCPROT

      ; RS-CANFD
      st.w     r0, 0xFF981000[r0]       ; set MSR_RSCFD (RS-CANFD8-15 is ・・・

      mov      0xA5A5A500, r2           ; set 0xA5A5A500 in MSRKCPROT for ・・・
      st.w     r2, 0xFF981710[r0]       ; set MSRKCPROT
```

Figure 2.7 Example Program Code for RS-CANFD module standby register settings(CS+)

2.3.4.5 Enabling PE1～3

To enable PE1,PE2 or PE3, set corresponding PEx bit of BOOTCTRL (PE1 bit1(BC1), PE2 bit2(BC2), PE3 bit3(BC3)) to 1.

・To enable PE1, set 1 to ENABLE_PE1_BY_PE0.

・To enable PE2, set 1 to ENABLE_PE2_BY_PE0.

・To enable PE3, set 1 to ENABLE_PE3_BY_PE0.

・The process PE is PE0(PEID bit0:2(PEID)=0

Figure 2.8 indicates a program code example.

```
        ld.w      0xfffb2000[r0], r10      ; get BOOTCTRL
        ori       2, r10, r11              ; set 1 in BOOTCTRL.BC1 for enabled PE1
        st.w      r11, 0xfffb2000[r0]      ; set BOOTCTRL
(Omitting the middle part)
        ld.w      0xfffb2000[r0], r10      ; get BOOTCTRL
        ori       4, r10, r11              ; set 1 in BOOTCTRL.BC2 for enabled PE2
        st.w      r11, 0xfffb2000[r0]      ; set BOOTCTRL
(Omitting the middle part)
        ld.w      0xfffb2000[r0], r10      ; get BOOTCTRL
        ori       8, r10, r11              ; set 1 in BOOTCTRL.BC3 for enabled PE3
        st.w      r11, 0xfffb2000[r0]      ; set BOOTCTRL
```

Figure 2.8 Example Program Code for PE1～3 to be enabled in CS+

2.3.4.6 Initializing RAM Areas

Initialize Local RAM and Cluster RAM.

In this project, to shorten the startup time, each PE executes initialization in the specified RAM address areas.

The RAM initializing in PE0 is as follows (U2A16, U2A8).

- Local RAM (CPU0) : 0xFDC00000～0xFDC0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE000000～0xFE03FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE400000～0xFE47FFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE800000～0xFE80FFFF (64KB)

The RAM initializing in PE0 is as follows (U2A6).

- Local RAM (CPU0) : 0xFDC00000～0xFDC0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE000000～0xFE03FFFF (256KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE800000～0xFE80FFFF (64KB)

The RAM initializing in PE1 is as follows (U2A16, U2A8).

- Local RAM (CPU1) : 0xFDA00000～0xFDA0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE040000～0xFE07FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE480000～0xFE4FFFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE810000～0xFE81FFFF (64KB)

The RAM initializing in PE1 is as follows (U2A6).

- Local RAM (CPU1) : 0xFDA00000～0xFDA0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE040000～0xFE07FFFF (256KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE810000～0xFE81FFFF (64KB)

The RAM initializing in PE2 is as follows (U2A16).

- Local RAM (CPU2) : 0xFD800000～0xFD80FFFF (64KB)
- Cluster RAM (Cluster1) : 0xFE100000～0xFE13FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE500000～0xFE57FFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE820000～0xFE82FFFF (64KB)

The RAM initializing in PE3 is as follows (U2A16).

- Local RAM (CPU3) : 0xFD600000～0xFD60FFFF (64KB)
- Cluster RAM (Cluster1) : 0xFE140000～0xFE17FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE580000～0xFE5FFFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE830000～0xFE83FFFF (64KB)

When some PEs are not to be started, adjust the specified address in order to execute RAM initialization in each starting PE equally.

Figure 2.9 indicates a program code example.

```
    ; local ram address
    LOCAL_RAM_CPU0_ADDR    .set     0xFDC00000
    LOCAL_RAM_CPU0_END     .set     0xFDC0FFFF
  (Omitting the middle part)
    ; cluster ram address
    CLUSTER_RAM0_ADDR0     .set     0xFE000000
    CLUSTER_RAM0_END0      .set     0xFE03FFFF
  (Omitting the middle part)
    ; clear Cluster RAM0
    mov    CLUSTER_RAM0_ADDR0, r6
    mov    CLUSTER_RAM0_END0, r7
    jarl   _zeroclr4, lp

    ; clear Cluster RAM2
    mov    CLUSTER_RAM2_ADDR0, r6
    mov    CLUSTER_RAM2_END0, r7
    jarl   _zeroclr4, lp

    ; clear Cluster RAM3
    mov    CLUSTER_RAM3_ADDR0, r6
    mov    CLUSTER_RAM3_END0, r7
    jarl   _zeroclr4, lp

    ; clear Local RAM(CPU0)
    mov    LOCAL_RAM_CPU0_ADDR, r6
    mov    LOCAL_RAM_CPU0_END, r7
      jarl   _zeroclr4, lp
  (Omitting the middle part)
_zeroclr4:
    br      .L.zeroclr4.2
.L.zeroclr4.1:
    st.w    r0, [r6]
    add     4, r6
.L.zeroclr4.2:
    cmp     r6, r7
    bh      .L.zeroclr4.1
```

Figure 2.9 RAM Example Program Code for Initializing the .data and .bss Sections in CS+(PE0)

2.3.4.7 Timing Synchronization (PE0~PE3)

Proceeding PE waits so that other PEs can be processed further simultaneously.

This process is to be executed only when all the following conditions are satisfied

- ENABLE_PE1_BY_PE0 is 1
- ENABLE_PE2_BY_PE0 is 1
- ENABLE_PE3_BY_PE0 is 1

Figure 2.10 indicates program code examples for the wait process of PE0.

```
    CLUSTER_RAM2_ADDR       .set      0xFE400000
    CRAM_ADDR               .set      CLUSTER_RAM2_ADDR
  (Omitting the middle part)
    mov     CRAM_ADDR, r10
    set1    0, [r10]                ; Bit0 indicate PE0 wait for PEx

    ; wait for PE1
  .L.hdwinit_PE0.1:
    tst1    1, [r10]                ; Bit1 indicate PE1 wait for PE0
    bnz     .L.hdwinit_PE0.2
    snooze
    br      .L.hdwinit_PE0.1

  .L.hdwinit_PE0.2:
    ; wait for PE2
  .L.hdwinit_PE0.3:
    tst1    2, [r10]                ; Bit2 indicate PE2 wait for PE0
    bnz     .L.hdwinit_PE0.4
    snooze
    br      .L.hdwinit_PE0.3

  .L.hdwinit_PE0.4:
    ; wait for PE3
  .L.hdwinit_PE0.5:
    tst1    3, [r10]                ; Bit3 indicate PE3 wait for PE0
    bnz     .L.hdwinit_PE0.6
    snooze
    br      .L.hdwinit_PE0.5

.L.hdwinit_PE0.6:
```

Figure 2.10 Example Program Code for PE0 Side in the Timing Synchronization (CS+).

2.3.4.8 Setting Interrupt Handler Address

Set a base pointer address in the table reference method to INTBP.

The base pointer address to set is a starting address of EIINTTBL section.

Set the initial value of RBASE: (PE0,PE1: 0x00000000, PE2,PE3: 0x00800000)for a base address of the Direct Vector Method, since "0" was set to PSW: bit15 (EBV) through the register initialization (Section 3.3.4.2).

If "1" was set to PSW: bit15 (EBV), "0" which has been set to EBASE through the register initialization is to be used.

This process is to be executed only when all the following conditions are satisfied.

- ・USE_TABLE_REFERENCE_METHOD is 1.

Figure 2.11 indicates a program code example.

```
mov     #__sEIINTTBL_PE0, r10
ldsr    r10, 4, 1                  ; set INTBP
```

Figure 2.11 Example Program Code for the Interrupt Handler Address Setting in CS+(PE0)

2.3.4.9 Setting Each Pointer

Set Stack pointer, Global pointer, and Element pointer.

The value to be set to each pointer is as follows:

- Stack pointer

  The address of _stacktop_pm0, if the running PE is PE0(PEID bit0:2(PEID)=0)

  The address of _stacktop_pm1, if the running PE is PE1(PEID bit0:2(PEID)=1)

  The address of _stacktop_pm2, if the running PE is PE2(PEID bit0:2(PEID)=2)

  The address of _stacktop_pm3, if the running PE is PE3(PEID bit0:2(PEID)=3)

- Global pointer.[Note1]

  The starting address of __gp_data [Note2]

- Element pointer [Note3]

  The starting address of __ep_data [Note4]

Figure 2.12 indicates a program code example in PE0.

```
STACKSIZE .set 0x200
       .section  ".stack.bss", bss
       .align    4
       .ds   (STACKSIZE)
       .align    4
_stacktop_pm0:
   (Omitting the middle part)
       mov      #_stacktop_pm0, sp        ; set sp register
       mov      #__gp_data, gp            ; set gp register
       mov      #__ep_data, ep            ; set ep register
```

Figure 2.12 Example Program Code for each Pointer Setting in PE0

[Note1] To be processed in all PEs(PE0~PE3).

[Note2] __gp_data is label that be automatically generated by linker

[Note3] To be processed in all PEs(PE0~PE3).

[Note4] __ep_data is label that be automatically generated by linker

2.3.4.10 Setting RAM Areas

__INITSCT_RH routine Note 1 initializes the .data section (RAM section with initial value) and the .bss section (RAM section without initial value).

Copying data from ROM to the .data section and clearing the .bss section to zero are executed by calling the __INITSCT_RH routine with setting the starting and end addresses of initialization table for RAM section with initial value to parameter register (r6 and r7), and the starting and end addresses of initialization table for RAM section without initial value to parameter register (r8 and r9).

The initialization table for RAM section with initial value needs to be located in the .INIT_DSEC.const section. And the starting and end addresses of copy source ROM section and the starting address of copy destination RAM section need to be set to the table.

The initialization table for RAM section without initial value needs to be located in the .INIT_BSEC.const section. And the starting and end addresses of RAM section for clearing need to be set to the table.

Figure 2.13 indicates a program code example.

```
    ; when the section has the initial value
    .section  ".INIT_DSEC.const", const
    .align    4
    .dw       #__s.data, #__e.data, #__s.data.R

    ; when the section without initial value
    .section  ".INIT_BSEC.const", const
    .align    4
    .dw       #__s.bss, #__e.bss
(Omitting the middle part)
    mov       #__s.INIT_DSEC.const, r6     ; INIT_DSEC section begin address
    mov       #__e.INIT_DSEC.const, r7     ; INIT_DSEC section end address
    mov       #__s.INIT_BSEC.const, r8     ; INIT_BSEC begin address
    mov       #__e.INIT_BSEC.const, r9     ; INIT_BSEC end address
    jarl32    __INITSCT_RH, lp             ; initialize RAM area
```

Figure 2.13 Example Program Code for Initializing the .data and .bss Sections in CS+

Some RAMs such as DTSRAM and MMCA RAM are initialized by RAM Initialization function of hardware. For details of the RAM Initialization function, refer to the device's user's manual.

Note 1 __INITSCT_RH routine is provided by a compiler.

2.3.4.11 Setting Coprocessor

Setting 1 to FEPSW bit16 (CU0) enables FPU.

If FPU is not needed, set 0 to PSW bit16 (CU0).

Figure 2.14 indicates a program code example.

```
    stsr    5, r10, 0           ; get PSW
    movhi   0x0001, r0, r11     ; set 1 in PSW.CU0 for enabling FPU
    or      r11, r10
    ldsr    r10, 3, 0           ; set PSW via FEPSW
(Omitting the middle part)
    feret                       ; apply PSW and PC
```

Figure 2.14 Example Program Code for Coprocessor Setting in CS+

2.3.4.12 Calling a Main Function of User Application

Program counter transitions to the user application's main functions:

_main_pm0, if the running PE is PE0(PEID bit0:2(PEID)=0).

_main_pm1, if the running PE is PE1(PEID bit0:2(PEID)=1).

_main_pm2, if the running PE is PE2(PEID bit0:2(PEID)=2).

_main_pm3, if the running PE is PE3(PEID bit0:2(PEID)=3).

Figure 2.15 indicates a program code example for calling a main function in the user application for PE0.

```
    mov     #_main_pm0, r10
    ldsr    r10, 2, 0           ; set _main_pm0 address to PC via FEPC

    feret                       ; apply PSW and PC
```

Figure 2.15 Example Program Code for Calling a Main Function of User Application in CS+(PE0)

3. MULTI

This chapter describes the Startup process in using MULTI as the integrated development environment.

3.1 Startup Related Files

Table 3.1 indicates the files related to startup.

Table 3.1 Startup Related File List in MULTI

#	File	Directory	Description
1	startup.850(※1)	Project root/src/	PE0/PE1 Startup Routine Call, Vector Table
2	startup_PE0.850(※1)	Project root/src/	PE0 Startup Routine, Vector Table
3	startup_PE1.850(※1)	Project root/src/	PE1 Startup Routine, Vector Table
4	startup2.850(※1)	Project root/src/	PE2/PE3 Startup Routine Call, Vector Table
5	startup_PE2.850(※1)	Project root/src/	PE2 Startup Routine, Vector Table
6	startup_PE3.850(※1)	Project root/src/	PE3 Startup Routine, Vector Table
7	main.c	Project root/src/	Main Routine
8	main_pe0.c(※1)	Project root/src/	PE0 Main Routine
9	main_pe1.c(※1)	Project root/src/	PE1 Main Routine
10	main_pe2.c(※1)	Project root/src/	PE2 Main Routine
11	main_pe3.c(※1)	Project root/src/	PE3 Main Routine
12	section.ld(※1)	Project root/	Section Settings

*1: You can set a file name optionally.

When you change a file name, modify the project files as well.

3.2 Setting Sections

Table 3.2 indicates examples of startup related sections.

Table 3.2 Startup Related Sections in MULTI

#	Section	Allocated Data
1	.cintvect	First of interrupt vector table of PE0/PE1
2	.coldboot	Boot controller of PE0/PE1
3	.intvect_PE0	Interrupt vector table of PE0
4	.intvect_PE1	Interrupt vector table of PE0
5	.rozdata	Fixed data
6	.robase	
7	.rosdata	
8	.rodata	
9	.text	
10	.ascet_const	
11	.mytext0	
12	.fixaddr	
13	.fixtype	
14	.secinfo	
15	.syscall	
16	.romdata	Data with initial value (ROM)
17	.romsldata	
18	.cintvect2	Reset vector table of PE2/PE3
19	.coldboot2	Boot controller of PE2/PE3
20	.intvect_PE2	Interrupt vector table of PE2
21	.intvect_PE3	Interrupt vector table of PE3
22	.romdata	Data with initial value (ROM)
23	.data	Data with initial value (RAM)
24	.bss	Data without initial value (RAM)
25	.sdabase	SDA (Small Data Area) base register
26	.stack	Stack area
27	.heapbase	Base address of heap area
28	.heap	Heap area

Depending on each device or project, the name may be different, some sections may not be needed, or other sections may be needed. For details of the sections, refer to the user manuals of MULTI and devices.

3.2.1 Section Setting Method

The following describes a method of specifying or adding a section in a program in the assembly and C languages.

If a section is added, the section.Id is needed to be modified in addition to the following specifications.

・Assembly Language

.section Section Name [,"attribute"] [> location address]

Section Name: Specify the name of section.

Attribute: Specify an attribute.

Table 3.3 indicates specifiable attributes.

If you specify multiple attribute, specify them enclosing with double quotations.

e.g. "ab"

Location address: Specify the address where a section is to be located.

Table 3.3 List of Attributes Specifiable with .section

#	Attribute	Description
1	a	The section has the allocated memory which is not used for debugging or symbol information.
2	b	The section can have BSS semantics. Although normal data directives such as .word and .byte are allowed in .bss section, all the values specified in these directives are to be discarded by an assembler. Only the section size is recorded in ELF output file, and the content of the section is omitted. When the section is downloaded to target, an area is allocated for the section, but data is NOT downloaded to the section. Instead, the startup code initializes all bytes in the section to zero.
3	w	The section is writable.
4	z	The section contains executable code.

・C Language

#pragma ghs section [*Section Type*=”*Section Name*”]

Section Type：Specify a type of section to change the allocation.

Section Name：Specify a name of section.

Table 3.4 indicates the specifiable section type.

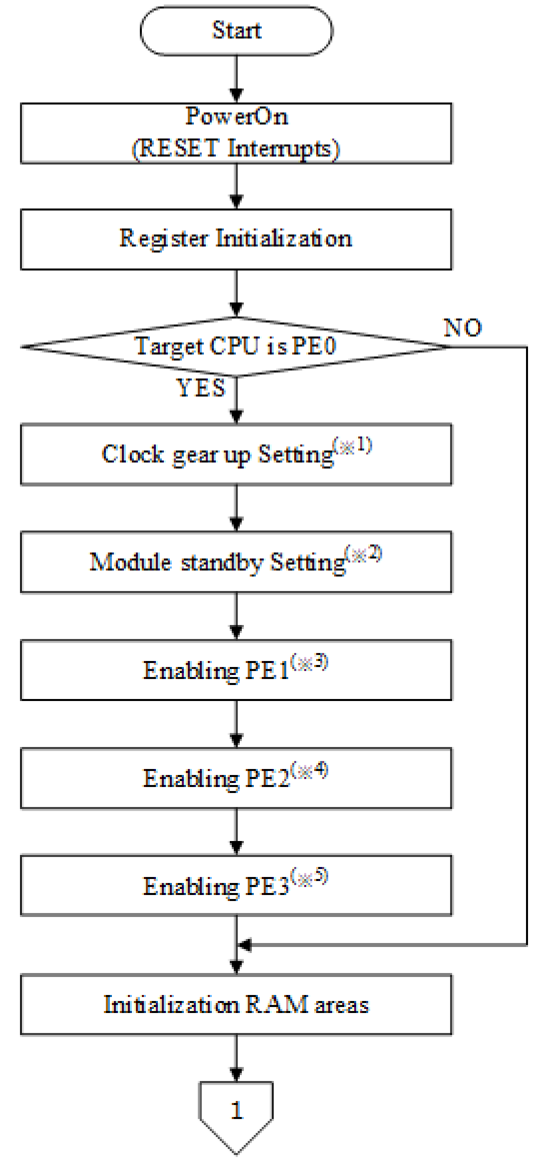
Table 3.4 List of Specifiable Section Type in the #pragma ghs Section

#	Section Type	Program Section by Default	Note
1	bss	.bss	
2	Data	.data	
3	text	.text	
4	rodata	.rodata	
5	sbss	.sbss	
6	sdata	.sdata	
7	rosdata	.rosdata	
8	zbss	.zbss	
9	zdata	.zdata	
10	rozdata	.rozdata	

3.3 Startup Processing

3.3.1 Overall Flow

Figure 3.1 and Figure 3.2 indicate the overall flow of the startup processing in MULTI [Note1].



※1：Only if 1 is set to ENABLE_CLOCK_GEARUP.

※2：Only if 1 is set to ENABLE_MODULE_STANDBY_SET.

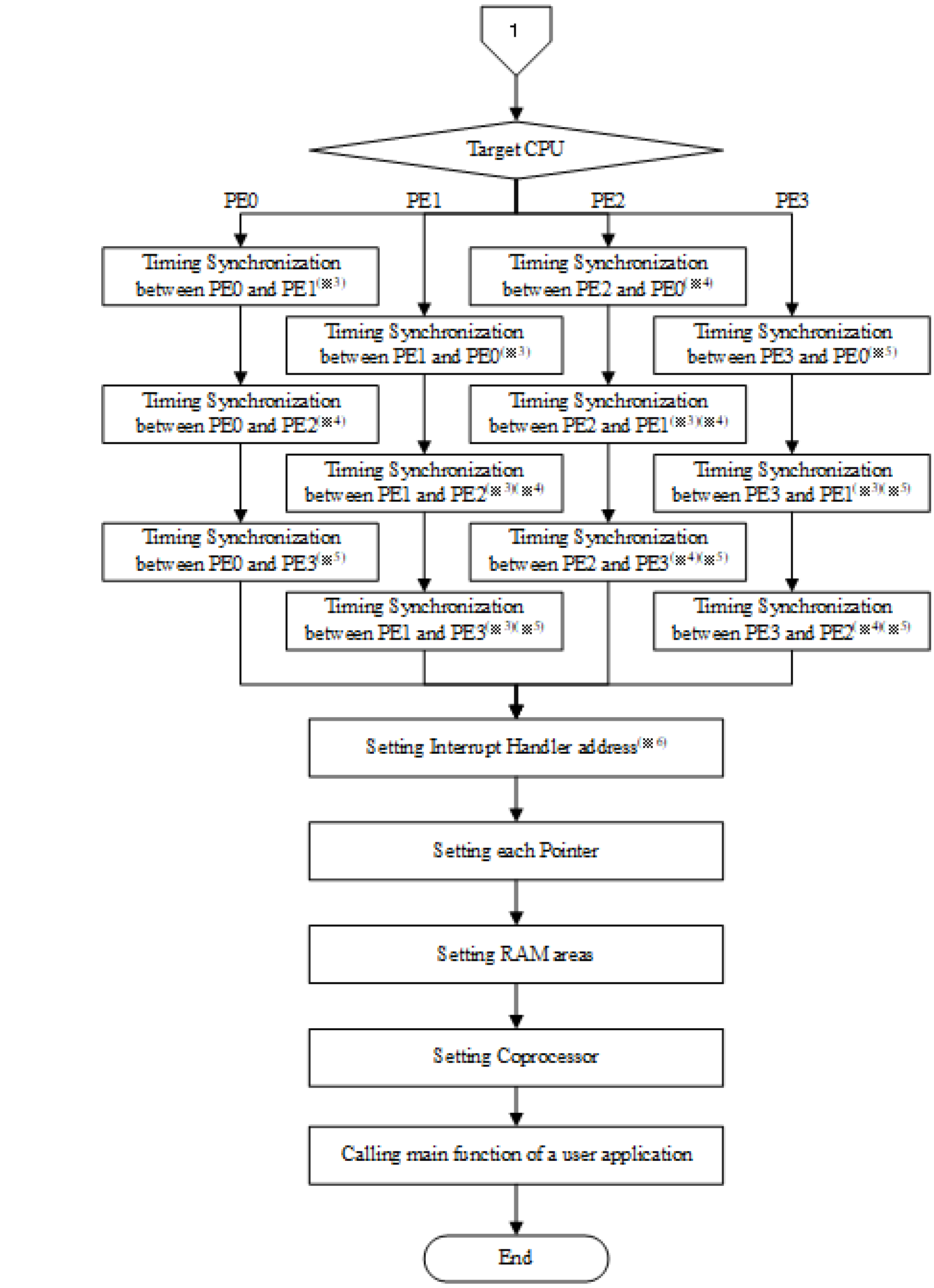
※3：Only if 1 is set to ENABLE_PE1_BY_PE0.

※4：Only if 1 is set to ENABLE_PE2_BY_PE0.

※5：Only if 1 is set to ENABLE_PE3_BY_PE0.

Figure 3.1 Overall Flow of Startup in MULTI (Part.1)

[Note1] Initial state (initially stopped or active state) of initially stopped core is selected by debug tool. For the details of debugging for initially stopped core, please refer to User's manual and Additional documents for emulator, and User's manual and help for the emulator debugger.



※3：Only if 1 is set to ENABLE_PE1_BY_PE0.

※4：Only if 1 is set to ENABLE_PE2_BY_PE0.

※5：Only if 1 is set to ENABLE_PE3_BY_PE0.

※6：Only if 1 is set to USE_TABLE_REFERENCE_METHOD.

Figure 3.2 Overall Flow of Startup in MULTI (Part.2)

3.3.2 Process Routines

Table 3.5, Table 3.6 and Table 3.7 indicates the routines where each process has been implemented.

Table 3.5 Process Implemented-Routine List in MULTI(Part.1)

#	Process Name	Routine Name	Implementation Files	Notes
1	Power on (Reset interrupts)	-	startup.850 startup2.850	The process of PE0 and PE1 are implemented in vector table of startup.850, the process of PE2 and PE3 are implemented in vector table of startup2.850.
2	Initializing registers	_RESET _RESET2	startup.850 startup2.850	The process of PE0 and PE1 are implemented in startup.850, the process of PE2 and PE3 are implemented in startup2.850.
3	Clock gearup settings	_clock_gearup	startup.850	To be processed only when running PE is PE0. (※1)
4	Module standby settings	_module_standby_set	startup.850	To be processed only when running PE is PE0. (※2)
5	Enabling PE1～3	__start_PE0	startup.850	To be processed only when running PE is PE0. (※3) (※4) (※5)
6	Initializing RAM areas	__start_PE0 __start_PE1 __start_PE2 __start_PE3	startup.850 startup2.850	To be processed with "_hdwinit_PE0" when the running PE is PE0, with "_hdwinit_PE1" when the running PE is PE1, with "_hdwinit_PE2" when the running PE is PE2, and with "_hdwinit_PE3" when the running PE is PE3. (※3) (※4) (※5)

*1 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_CLOCK_GEARUP is 0, PE will not be processed.

*2 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_ MODULE_STANDBY_SET is 0, PE will not be processed.

*3 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_PE1_BY_PE0 is 0, PE1 will not be processed.

*4 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_PE2_BY_PE0 is 0, PE2 will not be processed.

*5 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_PE3_BY_PE0 is 0, PE3 will not be processed.

Table 3.6 Process Implemented-Routine List in MULTI(Part.2)

#	Process Name	Routine Name	Implementation Files	Notes
7	Timing synchronization	__start_PE0 __start_PE1 __start_PE2 __start_PE3	startup.850 startup2.850	To be processed with "_hdwinit_PE0" when the running PE is PE0, with "_hdwinit_PE1" when the running PE is PE1, with "_hdwinit_PE2" when the running PE is PE2, and with "_hdwinit_PE3" when the running PE is PE3. (※3) (※4) (※5)
8	Interrupt handler address settings	_init_eiint _init_eiint2	startup.850 startup2.850	The process of PE0 and PE1 are implemented in startup.850, and the process of PE2 and PE3 are implemented in startup2.850.
9	Initializing each pointer	_RESET_PE0 _RESET_PE1 _RESET_PE2 _RESET_PE3	startup_PE0.850 startup_PE1.850 startup_PE2.850 startup_PE3.850	To be processed with "_RESET_PE0" when the running PE is PE0, with "_RESET_PE1" when the running PE is PE1, with "_RESET_PE2" when the running PE is PE2, and with "_RESET_PE3" when the running PE is PE3. (※3) (※4) (※5)
10	Setting Coprocessor	_RESET_PE0 _RESET_PE1 _RESET_PE2 _RESET_PE3	startup_PE0.850 startup_PE1.850 startup_PE2.850 startup_PE3.850	To be processed with "_RESET_PE0" when the running PE is PE0, with "_RESET_PE1" when the running PE is PE1, with "_RESET_PE2" when the running PE is PE2, and with "_RESET_PE3" when the running PE is PE3. (※3) (※4) (※5)

*3 The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE1_BY_PE0 is 0, PE1 will not be processed.

*4 The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE2_BY_PE0 is 0, PE2 will not be processed.

*5 The running PE is judged by PEID bit0:2(PEID).
However, in the case that ENABLE_PE3_BY_PE0 is 0, PE3 will not be processed.

Table 3.7 Process Implemented-Routine List in MULTI(Part.3)

#	Process Name	Routine Name	Implementation Files	Notes
11	Calling a main function of user application	_RESET_PE0 _RESET_PE1 _RESET_PE2 _RESET_PE3	startup_PE0.850 startup_PE1.850 startup_PE2.850 startup_PE3.850	To be processed with "_RESET_PE0" when the running PE is PE0, with "_RESET_PE1" when the running PE is PE1, with "_RESET_PE2" when the running PE is PE2, and with "_RESET_PE3" when the running PE is PE3. (※3) (※4) (※5)

*3 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_PE1_BY_PE0 is 0, PE1 will not be processed.

*4 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_PE2_BY_PE0 is 0, PE2 will not be processed.

*5 The running PE is judged by PEID bit0:2(PEID).

However, in the case that ENABLE_PE3_BY_PE0 is 0, PE3 will not be processed.

3.3.3 Details of Each Process

This section describes the details of each process.

3.3.3.1 Power-On (RESET Interrupts)

On powering on, the program counter becomes RESET vector address(RESET section).

According to the setting of RESET vector, program counter transits to _RESET section for PE0 / PE1 and transits to _RESET2 section for PE2 / PE3.

When the device power on itself, PE0 startup. On processing(reference) PE0, enable PE1 when ENABLE_PE1_BY_PE0 is 1, enable PE2 when ENABLE_PE2_BY_PE0 is 1, and enable PE3 when ENABLE_PE3_BY_PE0 is 1.

Figure 3.3 Example Program Code for Power-On: RESET Interrupts of PE0/PE1 in MULTI indicates program code examples.

```
    .global _RESETVECT
    .global _RESET
    .offset 0x0000
    #if (RESET_ENABLE > 0x00000000)
      .extern _RESET
_RESETVECT:
      jr _RESET                -- jump to _RESET(startup routine)
    #else
      jr __unused_isr
    #endif
```

Figure 3.3 Example Program Code for Power-On: RESET Interrupts of PE0/PE1 in MULTI

3.3.3.2 Initializing Registers

Table 3.8 and Table 3.9 indicates registers for initialization. Since the setting value is an example, please initialize with optimum value depending on the system.

Table 3.8 Initialization Register List in MULTI(Part.1)

#	Register Type	Register Name	Setting Value Examples	Description
1	Program registers	r1～r31	0	
2	Basic system registers	EIPC	0	
3		FEPC	0	
4		CTPC	0	
5		EIWR	0	
6		FEWR	0	
7		EBASE	0	
8		INTBP	0	
9		MEA	0	
10		MEI	0	
11		RBIP	0	
12		PSW	0x00010020	ID=1: Prohibit receiving EI level exceptions. CU0=1: Enable FPU
13	FPU system registers	FPSR	0x00220000	Value after Reset
14		FPEPC	0	
15	MPU function system registers	FPST	0	
16		FPCC	0	
17		MCA	0	
18		MCS	0	
19		MCR	0	
20		MPLA[Note1]	0	
21		MPUA[Note1]	0	
22		MPAT [Note1]	0	
23		MPID0	0	
24		MPID1	0	
25		MPID2	0	
26		MPID3	0	
27		MPID4	0	
28		MPID5	0	

[Note1] The registers of all the 32 MPU entries should be initialized. Set the index register MPIDX from 0 to 31, and initialize the corresponding MPLA, MPUA, MPAT registers.

Table 3.9 Initializing registers list in MULTI(Part.2)

#	Register Type	Register Name	Setting Value Example	Description
29	MPU function system registers	MPID6	0	
30		MPID7	0	
31		MCI	0	
32	Cache operation function registers	ICTAGL	0	
33		ICTAGH	0	
34		ICDATL	0	
35		ICDATH	0	
36		ICERR	0	
37	Virtualization support function system register	HVSB	0	
38	Guest Context Register	GMEIPC	0	
39		GMFEPC	0	
40		GMEBASE	0	
41		GMINTBP	0	
42		GMEIWR	0	
43		GMFEWR	0	
44		GMMEA	0	
45		GMMEI	0	

Figure 3.4 indicates program code examples of initializing registers.

```
   mov    r0, r1
   mov    r0, r2
(Omitting the middle part)
   ldsr   r0,  0, 0          -- SR0,0   EIPC
   ldsr   r0,  2, 0          -- SR2,0   FEPC
   ldsr   r0, 16, 0          -- SR16,0 CTPC
(Omitting the middle part)
   mov    0x00010020, r10
   ldsr   r10, 5, 0          -- SR5,0   PSW
(Omitting the rest)
```

Figure 3.4 Example Program Code for Initializing Registers in MULTI

3.3.3.3 Clock Gearup Settings

After starting PE0, change the system clock as “PLL”, and execute clock gearup.

This process is to be executed only when all the following conditions are satisfied.

・ENABLE_CLOCK_GEARUP is 1.

・The running PE is PE0(PEID bit0:2(PEID)=0)

・Main OSC and PLL are enabled (PLLS=0x00000003)

Figure 3.5 and Figure 3.6 indicate setting method examples for clock gearup as a reference.

```
.L.clock_gearup.0:
  ld.w      0xff980004[r0], r2      -- get PLLS
  andi      0x3, r2, r2
  cmp       0x3, r2
  bnz       .L.clock_gearup.0

  mov       0xA5A5A501 , r2          -- set 0xA5A5A501 in CLKKCPROT1
for ・・・
  st.w      r2, 0xff980700[r0]      -- set CLKKCPROT1

  ld.w      0xff980120[r0], r2      -- get CLKD_PLLC
  ori       0x2, r2, r2             -- set 2 in CLKD_PLLC.PLLCLKDCSID・・・
  mov       -0x6, r6
  and       r6, r2
  st.w      r2, 0xff980120[r0]      -- set CLKD_PLLC

.L.clock_gearup.1:
  ld.w      0xff980128[r0], r2      -- get CLKD_PLLS
  andi      0x2, r2, r0              -- confirm that the value of CLKD_ ・・・
  bz        .L.clock_gearup.1        -- if the CLKD_PLLS.PLLCLKD ・・・
(Omitting the middle part)
  ld.w      0xff980100[r0], r2      -- get CKSC_CPUC
  mov       -0x2, r6                -- set 0 in CKSC_CPUC.CPUCLKSC・・・
  and       r6, r2
  st.w      r2, 0xff980100[r0]      -- set CKSC_CPUC

.L.clock_gearup.4:
  ld.w      0xff980108[r0], r2        -- get CKSC_CPUS
```

Figure 3.5 Example Program Code for Clock Gearup Settings in MULTI(Part.1)

```
   andi      0x1, r2, r0              -- confirm that the value of CKSC_CPUS・・・
   bnz       .L.clock_gearup.4        -- if the CKSC_CPUS.CPUCLKSACT
is ・・・
(Omitting the middle part)
   ld.w      0xff980120[r0], r2       -- get CLKD_PLLC
   ori       0x1, r2, r2              -- set 1 in CLKD_PLLC.PLLCLKD. ・・・
   mov       -0x7, r6
   and       r6, r2
   st.w      r2, 0xff980120[r0]       -- set CLKD_PLLC

.L.clock_gearup.7:
   ld.w      0xff980128[r0], r2       -- get CLKD_PLLS
   andi      0x2, r2, r0              -- confirm that the value of CLKD_ ・・・
   bz        .L.clock_gearup.7        -- if the CLKD_PLLS.PLLCLKD ・・・
(Omitting the middle part)
   mov       0xA5A5A500, r2           -- set 0xA5A5A500 in CLKKCPROT1・・・
   st.w      r2, 0xff980700[r0]       -- set CLKKCPROT1
```

Figure 3.6 Example Program Code for Clock Gearup Settings in MULTI(Part.2)

3.3.3.4 Module Standby Settings

Set the module standby registers depending on a function to be used. (Enable/disable clock supply to each function to be used)

This process is to be executed only when all the following conditions are satisfied.

・ENABLE_MODULE_STANDBY_SET is 1

・The running PE is PE0(PEID bit0:2(PEID)=0

Figure 3.7 indicates the example of setting module standby registers for RS-CANFD as a reference.

```
   mov       0xA5A5A501, r2           -- set 0xA5A5A501 in MSRKCPROT ・・・
   st.w      r2, 0xFF981710[r0]       -- set MSRKCPROT

   -- RS-CANFD
   st.w      r0, 0xFF981000[r0]       -- set MSR_RSCFD (RS-CANFD8-15 ・・・

   mov       0xA5A5A500, r2           -- set 0xA5A5A500 in MSRKCPROT ・・・
   st.w      r2, 0xFF981710[r0]       -- set MSRKCPROT
```

Figure 3.7 Example Program Code for RS--CANFD Module Standby Register Settings in MULTI

3.3.3.5 Enabling PE1～3

To enable PE1,PE2 or PE3, set corresponding PEx bit of BOOTCTRL (PE1 bit1(BC1), PE2 bit2(BC2), PE3 bit3(BC3)) to 1.

This process is to be executed only when all the following conditions are satisfied.

・In the case of enabling PE1, ENABLE_PE1_BY_PE0 is 1.

・In the case of enabling PE2, ENABLE_PE2_BY_PE0 is 1.

・In the case of enabling PE3, ENABLE_PE3_BY_PE0 is 1.

・The running PE is PE0(PEID bit0:2(PEID)=0)

Figure 3.8 indicates program code examples for PE to be enabled in MULTI.

```
   ld.w     0xfffb2000[r0], r10        -- get BOOTCTRL
   ori      2, r10, r11                -- set 1 in BOOTCTRL.BC1 for  ・・・
   st.w     r11, 0xfffb2000[r0]        -- set BOOTCTRL
(Omitting the middle part)
   ld.w     0xfffb2000[r0], r10        -- get BOOTCTRL
   ori      4, r10, r11                -- set 1 in BOOTCTRL.BC2 for  ・・・
   st.w     r11, 0xfffb2000[r0]        -- set BOOTCTRL
(Omitting the middle part)
   ld.w     0xfffb2000[r0], r10        -- get BOOTCTRL
   ori      4, r10, r11                -- set 1 in BOOTCTRL.BC2 for  ・・・
   st.w     r11, 0xfffb2000[r0]        -- set BOOTCTRL
```

Figure 3.8 Example Program Code for PE1~3 to be enabled in MULTI

3.3.3.6 Initializing RAM Areas

Initialize Local RAM and Cluster RAM.

In this project, to shorten the startup time, each PE executes initialization in the specified RAM address areas

The RAM initializing in PE0 is as follows (U2A16, U2A8).

- Local RAM (CPU0) : 0xFDC00000～0xFDC0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE000000～0xFE03FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE400000～0xFE47FFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE800000～0xFE80FFFF (64KB)

The RAM initializing in PE0 is as follows (U2A6).

- Local RAM (CPU0) : 0xFDC00000～0xFDC0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE000000～0xFE03FFFF (256KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE800000～0xFE80FFFF (64KB)

The RAM initializing in PE1 is as follows (U2A16, U2A8).

- Local RAM (CPU1) : 0xFDA00000～0xFDA0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE040000～0xFE07FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE480000～0xFE4FFFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE810000～0xFE81FFFF (64KB)

The RAM initializing in PE1 is as follows (U2A6).

- Local RAM (CPU1) : 0xFDA00000～0xFDA0FFFF (64KB)
- Cluster RAM (Cluster0) : 0xFE040000～0xFE07FFFF (256KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE810000～0xFE81FFFF (64KB)

The RAM initializing in PE2 is as follows (U2A16).

- Local RAM (CPU2) : 0xFD800000～0xFD80FFFF (64KB)
- Cluster RAM (Cluster1) : 0xFE100000～0xFE13FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE500000～0xFE57FFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE820000～0xFE82FFFF (64KB)

The RAM initializing in PE3 is as follows (U2A16).

- Local RAM (CPU3) : 0xFD600000～0xFD60FFFF (64KB)
- Cluster RAM (Cluster1) : 0xFE140000～0xFE17FFFF (256KB)
- Cluster RAM (Cluster2) : 0xFE580000～0xFE5FFFFF (512KB)
- Cluster RAM (Cluster3)(Retention RAM) : 0xFE830000～0xFE83FFFF (64KB)

When some PEs are not to be started, adjust the specified address in order to execute RAM initialization in each starting PE equally.

Figure 3.9 indicates program code examples for RAM initialization in MULTI.

```
  -- clear Cluster RAM0
  mov          0xFE000000, r6
  mov          0xFE03FFFF, r7
  jarl        _zeroclr, lp

  -- clear Cluster RAM2
  mov          0xFE400000, r6
  mov          0xFE47FFFF, r7
  jarl        _zeroclr, lp

  -- clear Cluster RAM3
  mov          0xFE800000, r6
  mov          0xFE80FFFF, r7
  jarl        _zeroclr, lp

  -- clear Local RAM(CPU0)
  mov          0xFDC00000, r6
  mov          0xFDC0FFFF, r7
  jarl        _zeroclr, lp
 (Omitting the middle part)
_zeroclr4:
    br        .L.zeroclr4.2
.L.zeroclr4.1:
    st.w      r0, [r6]
    add       4, r6
.L.zeroclr4.2:
    cmp       r6, r7
    bh        .L.zeroclr4.1
```

Figure 3.9 Program Code Examples for RAM initialization (for PE0) in MULTI

3.3.3.7 Timing Synchronization (PE0～3)

In order to synchronize each PE, before user application calling, each PE waits until other PEs have arrived the same process.

After all PEs reach this process, each PE starts the processes simultaneously.

This process is to be executed only when all the following conditions are satisfied.

・In the case of enabling PE1, ENABLE_PE1_BY_PE0 is 1.

・In the case of enabling PE2, ENABLE_PE2_BY_PE0 is 1.

・In the case of enabling PE3, ENABLE_PE3_BY_PE0 is 1.

Figure 3.10 indicates program code examples for the wait process of PE0.

```
.L.hdwinit_PE0.0:
    mov         0xFE400000, r10
    set1        0, 0[r10]             -- Bit0 indicate PE0 wait for PEx

    -- wait for PE1
.L.hdwinit_PE0.1:
    tst1        1, 0[r10]             -- Bit1 indicate PE1 wait for PE0
    bnz         .L.hdwinit_PE0.2
    snooze
    br          .L.hdwinit_PE0.1

.L.hdwinit_PE0.2:

    -- wait for PE2
.L.hdwinit_PE0.3:
    tst1        2, 0[r10]             -- Bit2 indicate PE2 wait for PE0
    bnz         .L.hdwinit_PE0.4
    snooze
    br          .L.hdwinit_PE0.3

.L.hdwinit_PE0.4:

    -- wait for PE3
.L.hdwinit_PE0.5:
    tst1        3, 0[r10]             -- Bit3 indicate PE3 wait for PE0
    bnz         .L.hdwinit_PE0.6
    snooze
    br          .L.hdwinit_PE0.5

.L.hdwinit_PE0.6:
```

Figure 3.10 Example Program Code for Timing Synchronization(PE0~3) in MULTI.

3.3.3.8 Setting Interrupt Handler Address

Set the base pointer address in table reference method to INTBP.

The base pointer address to be set is the first address of EIINTTBL section.

The base address in direct vector method is set PSW:bit15(EBV)=0 at initializing register, thus, the initial value of RBASE (0x00000000 for PE0 and PE1, 0x00800000 for PE2 and PE3) is to be used.

In the case that 1 is set to PSW:bit15(EBV), 0 set in EBASE at initializing register is to be used.

This process is to be executed only when all the following conditions are satisfied.

・USE_TABLE_REFERENCE_METHOD is 1.

Figure 3.11 and Figure 3.12 indicate program code example for interrupt handler address settings in MULTI

```
#define IRQ_TABLE_START_PE0                    0x00000000u
#define IRQ_TABLE_START_PE1                    0x00000000u
#define IRQ_TABLE_START_PE2                    0x00800000u
#define IRQ_TABLE_START_PE3                    0x00800000u

    stsr      0, r10, 2              -- get PEID.PEID

    cmp       0, r10
    bnz       .L.init_eiint.1        -- if PEID.PEID is not 0
    mov       IRQ_TABLE_START_PE0, r10
    ldsr      r10, 4, 1              -- set INTBP
    br        .L.init_eiint.4

.L.init_eiint.1:
    cmp       1, r10
    bnz       .L.init_eiint.2        -- if PEID.PEID is not 1
    mov       IRQ_TABLE_START_PE1, r10
    ldsr      r10, 4, 1              -- set INTBP
    br        .L.init_eiint.4
```

Figure 3.11 Example Program Code for Interrupt Handler Address Settings in MULTI

```
.L.init_eiint.2:
   cmp        2, r10
   bnz        .L.init_eiint.3         -- if PEID.PEID is not 2
   mov        IRQ_TABLE_START_PE2, r10
   ldsr       r10, 4, 1               -- set INTBP
   br         .L.init_eiint.4

.L.init_eiint.3:
   cmp        3, r10
   bnz        .L.init_eiint.5         -- if PEID.PEID is not 3
   mov        IRQ_TABLE_START_PE3, r10
   ldsr       r10, 4, 1               -- set INTBP
   br         .L.init_eiint.4

.L.init_eiint.4:
```

Figure 3.12 Example Program Code for Interrupt Handler Address Settings in MULTI

3.3.3.9 Initializing Each Pointers

Set the global pointer, text pointer and stack pointers

The value set in each pointer are shown as follows.

・Global Pointer

The first address in.sdabase section.

・Text Pointer

The first address of.robase section.

・Stack Pointer

The end address of stack section.

Figure 3.13 indicates program code examples of initializing each pointer.

```
-- set global pointer
movhi       hi(___ghsbegin_sdabase),zero,gp
movea       lo(___ghsbegin_sdabase),gp,gp

-- set text pointer
movhi       hi(___ghsbegin_robase),zero,tp
movea       lo(___ghsbegin_robase),tp,tp

-- set stack pointer
movhi       hi(___ghsend_stack-4),zero,sp
movea       lo(___ghsend_stack-4),sp,sp
mov         -4,r1
and         r1,sp
```

Figure 3.13 Example Program Code for Initializing Each Pointer in MULTI

3.3.3.10 Setting Coprocessor

Set 1 to FEPSW bit16(CU0) to enable FPU.

If FPU is not needed, set 0 to PSW bit16(CU0).

Figure 3.14 indicates a program code example for coprocessor settings.

```
-- enable FPU
stsr     5, r10, 0            -- get PSW
movhi    0x0001, r0, r11      -- set 1 in PSW.CU0 for enable FPU
or       r11, r10
ldsr     r10, 3, 0            -- set PSW via FEPSW
```

Figure 3.14 Example Program Code for Coprocessor Settings in MULTI

3.3.3.11 Calling a Main Function of User Application

Program counter transitions to main in user application.

Figure 3.15 indicates an example program code for calling a main function of user application.

```
    jr       __start
```

Figure 3.15 Example Program Code for Calling a Main Function of User Application in MULTI

Revision History

Rev.	Date	Description	
		Page	Summary
0.50	2019.03.08	All sections	Issued the 1st version.
0.60	2019.11.06	16, 17	Modified Table 3.10 and Table 3.11. (Added SCBP register and modified the order of each register)
		18, 19	Added the following conditions for Clock gear up Setting. ・Main OSC and PLL are enabled (OPBT11.STARTUPPL = 0) Modified Figure 3.5 and Figure 3.6. (Modified Example Program Code for Register Initialization in CS+)
		20	Modified Figure 3.8. (Modified Example Program Code for PE1～3 to be enabled in CS+)
		22	Modified Figure 3.9. (Modified Example Program Code for RAM Initializing the .data and .bss Sections in CS+(PE0))
		26	Modified Figure 3.13. (Modified Example Program Code for Initializing the .data and .bss Sections in CS+)
		27	Modified Figure 3.14. (Modified Example Program Code for Coprocessor Setting in CS+)
		38, 39	Modified Table 4.8 and Table 4.9. (Added SCBP register and modified the order of each register)
		40, 41	Added the following conditions for Clock gear up Setting. ・Main OSC and PLL are enabled (OPBT11.STARTUPPL = 0) Modified Figure 4.5 and Figure 4.6. (Modified Example Program Code for Clock Gearup Settings in MULTI)
		42	Modified Figure 4.8. (Modified Example Program Code for PE1~3 to be enabled in MULTI)
		50	Modified Figure 4.14. (Modified Example Program Code for Calling a Main Function of User Application in MULTI)
0.70	2020.03.20	16, 38	Modified Table 3.10 and Table 4.8. (Removed CTBP register ,SCBP register and MPCFG register)
		16, 38	Added notes to Table 3.10 and Table 4.8. (MPLA, MPUA, MPAT registers)
1.00	2020.09.30	All section	Renumbered tables and figures to corresponding one.
		17.39	Modified Table 2.11 and Table 3.9. (Removed GMFEPSW, GMEIIC and GMFEIC registers) (Added GMMEA and GMMEI registers)
		18,40	2.3.4.3 / 3.3.4.3 Clock Gearup Settings Modified the conditions under which the clock gearup is performed.
1.01	2021.03.18	1,3	Added RH850/U2A8 as a target device. Added a Section 1.1 Note.
	2021.03.18	10,32	Added note for initially stopped core.

1.10	2022.01.31	1,3	Added RH850/U2A6 as a target device.
		18,19,40,41	Modified Example Program Code for Clock Gearup Settings. (Figure2.5, Figure 2.6, Figure 3.6 and Figure 3.7.)
		19,41	Modified Example Program Code for RS-CANFD module standby register settings. (Figure 2.7 and Figure 3.7)
		21,43	Added RAM initialize area for U2A6 operation. The initialization range of Cluster RAM (Cluster2, Cluster3) in U2A16 PE3 is modified.

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.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
8. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

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/.