RH850/U2A-EVA Group

Startup アプリケーションノート

要旨

本アプリケーションノートでは、ルネサスエレクトロニクスの自動車向けシングルチップマイクロコンピュータの RH850/U2A シリーズ(以降、U2A と称す)における Startup 処理について説明します。

本資料およびプログラムは、RH850/U2A 搭載機能の理解促進を意図するものであり、量産設計を対象とするものではありません。

また、最新のマニュアル、正誤表、テクニカルアップデートや、開発環境の更新を反映しておりません。該当機能を使用される場合には、本プログラムは参考として扱い、最新のドキュメントや開発環境にて、お客様の責任において行ってください。

対象デバイス

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

対象統合開発環境

CS+(ルネサスエレクトロニクス社製)

バージョン	: V8.07.00
デバイスファイル	: R7F702300.DVF
	: R7F702301.DVF
	: R7F702302.DVF

MULTI(Green Hills Software 社製)

プロダクト	: IDE for V800
バージョン	: 2021.1.5(v 7.1.6)
ターゲット	: V800/RH850
デバイスファイル	: R7F702300.DVF
	: R7F702301.DVF
	: R7F702302.DVF
EXEC ファイル	: ExecG3G4_V10500

MULTI の場合、MULTI 自体のバージョン等に依存するディレクトリ名等が存在します。バージョン等が異なる場合は適宜読み替えてください。

参照文書

RH850/U2A-EVA ユーザーズマニュアル ハードウェア編

デバイスの機能詳細及び電気的特性に関してはユーザーズマニュアル ハードウェア編に記載します。本アプリケーションノートは以下のマニュアルを参照し作成しております。

- RH850/U2A-EVA User's Manual (Rev.1.20): R01UH0864EJ0120

目次

1. 概要

Startup 処理は、PowerOn からユーザアプリケーションを呼び出すまでの処理です。

本アプリケーションノートは、ルネサスエレクトロニクス社製の統合開発環境である CS+(注1)、および Green Hills Software 社製の統合開発環境である MULTI を対象とした Startup 処理を示します。

1.1 ご使用上の注意

RH850/U2A シリーズは製品により、CPU 搭載数が異なります。
RH850/U2A16 は CPU を 4 つ搭載しており、RH850/U2A8, U2A6 は CPU を 2 つ搭載しています。

本アプリケーションノートでは、RH850/U2A16 の Startup 処理について記載しています。RH850/U2A8, U2A6 を使用する場合は、CPU2(PE2), CPU3(PE3)の処理は実行されません。

(注1) 旧名 CubeSuite+

2. CS+

本章では、統合開発環境として CS+を使用した場合の Startup 処理について説明します。

2.1 Startup 関連ファイル

Startup に関連するファイル一覧を表 2.1 Startup 関連ファイル一覧(CS+)に示します。

表 2.1 Startup 関連ファイル一覧(CS+)

#	ファイル	ディレクトリ	説明
1	boot.asm	プロジェクトルート/	ブート処理スタートアップ処理
2	cstart0～3.asm	プロジェクトルート/PE0～3/	ユーザアプリケーション用スタートアップ処理（PE 毎に有り）
3	vecttbl0～3.asm	プロジェクトルート/PE0～3/	ベクタテーブル（PE 毎に有り）
4	main0～3.c	プロジェクトルート/PE0～3/	メイン処理（PE 毎に有り）

本プロジェクトでは、PE0 および PE1 は vecttbl0.asm を、PE2 および PE3 は vecttbl2.asm を参照しており、vecttbl1.asm および vecttbl3.asm は未使用となっています。各 PE で参照するベクタテーブルを分ける場合、Reset Vector PEx レジスタの参照アドレスを変更して使用してください。

2.2 セクション設定

Startup に関連する主なセクションの例を表 2.2、表 2.3 に示します。

表 2.2 Startup 関連セクション(CS+) (その 1)

#	セクション	セクション	割り当てデータ
1	共通部（メイン	RESET_PE0～3	RESET ベクタ
2	プロジェクト）	EIINTTBL_PE0～3	テーブル参照方式用 EI レベル割込みベクタテーブル
3		.text	プログラムコード(boot.asm)
4	PE0（サブ	.const	読み取り専用データ
5	プロジェクト）	.INIT_DSEC.const	初期値ありセクションの初期化テーブル
6		.INIT_BSEC.const	初期値なしセクションの初期化テーブル
7		.text.cmn	boot.asm/cstat0.asm 間受け渡し情報
8		.text	プログラムコード(cstart0.asm/main0.c)
9		.data	初期値ありデータ(ROM)
10		.data.R	初期値ありデータ(RAM)
11		.bss	初期値なしデータ
12		.stack.bss	スタック
13	PE1（サブ	.const	読み取り専用データ
14	プロジェクト）	.INIT_DSEC.const	初期値ありセクションの初期化テーブル
15		.INIT_BSEC.const	初期値なしセクションの初期化テーブル
16		.text.cmn	boot.asm/cstat1.asm 間受け渡し情報
17		.text	プログラムコード(cstart1.asm/main1.c)
18		.data	初期値ありデータ(ROM)
19		.data.R	初期値ありデータ(RAM)
20		.bss	初期値なしデータ
21		.stack.bss	スタック

表 2.3 Startup 関連セクション(CS+) (その 2)

#	プロジェクト	セクション	割り当てデータ
22	PE2（サブ	.const	読み取り専用データ
23	プロジェクト）	.INIT_DSEC.const	初期値ありセクションの初期化テーブル
24		.INIT_BSEC.const	初期値なしセクションの初期化テーブル
25		.text.cmn	boot.asm/cstat2.asm 間受け渡し情報
26		.text	プログラムコード(cstart2.asm/main2.c)
27		.data	初期値ありデータ(ROM)
28		.data.R	初期値ありデータ(RAM)
29		.bss	初期値なしデータ
30		.stack.bss	スタック
31	PE3（サブ	.const	読み取り専用データ
32	プロジェクト）	.INIT_DSEC.const	初期値ありセクションの初期化テーブル
33		.INIT_BSEC.const	初期値なしセクションの初期化テーブル
34		.text.cmn	boot.asm/cstat3.asm 間受け渡し情報
35		.text	プログラムコード(cstart3.asm/main3.c)
36		.data	初期値ありデータ(ROM)
37		.data.R	初期値ありデータ(RAM)
38		.bss	初期値なしデータ
39		.stack.bss	スタック

デバイスやプロジェクトによっては名称が異なる、一部セクションが必要ない、または他のセクションが必要である場合もあります。セクションの詳細については、CS+やデバイスのユーザーズマニュアルを参照ください。

2.2.1 セクション指定方法

アセンブラや C 言語で、プログラム中でセクションの指定や追加等を行う際のセクション指定方法を以下に示します。

・アセンブラ

.section *セクション名 セクション種別*

セクション名：セクションの名前を指定します。

セクション種別：再配置属性を指定します。

・C 言語

#pragma section [*セクション種別*] [*セクション名*]

セクション種別：再配置属性を指定します。

セクション名：セクションの名前を指定します。

主な再配置属性を表 2.4 に示します。その他の再配置属性については CS+のユーザーズマニュアルを参照ください。

表 2.4 主な再配置属性

#	再配置属性	デフォルトセクション	備考
1	text	.text	.text セクションにはプログラムコードが配置されます。
2	r0_disp32	.data	.data セクションには初期値ありデータが配置されます。
3	const	.const	.const セクションには読み取り専用データが配置されます。
4	default	-	全ての指定を解除して、デフォルトセクションに戻します。

作成したセクションは「xxx.再配置属性」で参照可能です。

例えば、セクション名が newsection、再配置属性が text の場合は、「newsection.text」となります。

2.3 Startup 処理

2.3.1 条件アセンブル制御命令用定義

条件アセンブル制御命令用の定義を表 2.5、表 2.6 に示します。条件アセンブル制御命令定義は boot.asm にて定義されています。

表 2.5 条件アセンブル制御命令用定義一覧 (その 1)

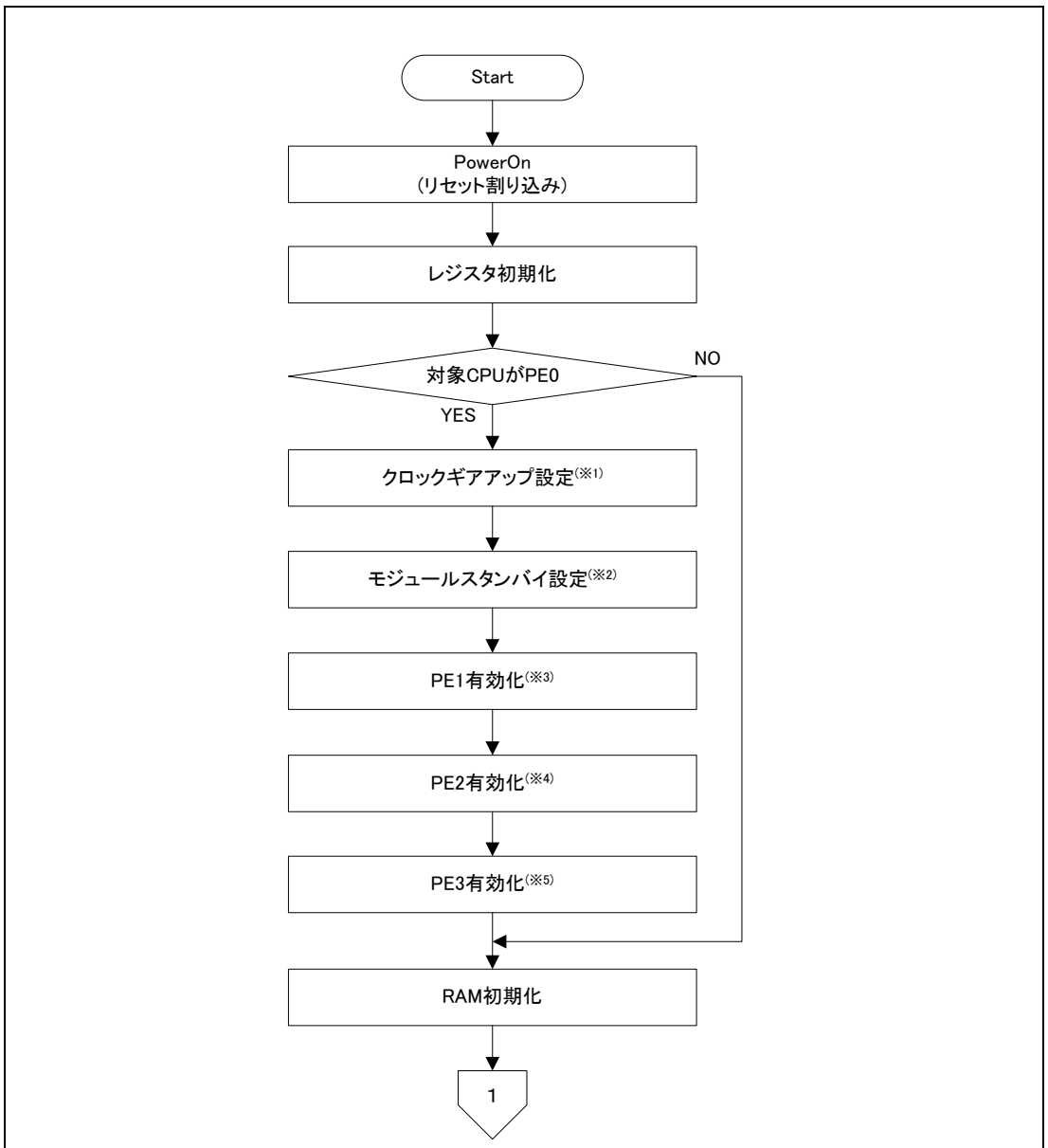
#	定義名	値	説明
1	ENABLE_PE1_BY_PE0		PE1 の有効化有無を定義します。PE の有効化は PE0 にて実施されます。 初期値は 1（PE1 を有効化する）です。
		0	PE1 を有効化しません。
		1	PE1 を有効化します。
2	ENABLE_PE2_BY_PE0		PE2 の有効化有無を定義します。PE の有効化は PE0 にて実施されます。 初期値は 1（PE2 を有効化する）です。
		0	PE2 を有効化しません。
		1	PE2 を有効化します。
3	ENABLE_PE3_BY_PE0		PE3 の有効化有無を定義します。PE の有効化は PE0 にて実施されます。 初期値は 1（PE3 を有効化する）です。
		0	PE3 を有効化しません。
		1	PE3 を有効化します。
4	USE_TABLE_REFERENCE_METHOD		割り込みベクタ方式としてテーブル参照方式の使用有無を定義します。 初期値は 1（テーブル参照方式を使用する）です。
		0	テーブル参照方式を使用しません。
		1	テーブル参照方式を使用します。
5	ENABLE_CLOCK_GEARUP		クロックギアアップの実施有無を定義します。クロックギアアップは PE0 にて実施されます。 初期値は 1（クロックギアアップを実施する）です。
		0	クロックギアアップを実施しません。
		1	クロックギアアップを実施します。

表 2.6 条件アセンブル制御命令用定義一覧 (その 2)

#	定義名	値	説明
6	ENABLE_MODULE_STANDBY_SET		モジュールスタンバイ設定（使用する各機能へのクロック供給有無の設定）の実施有無を定義します。 初期値は 0（モジュールスタンバイ設定を実施しない）です。
		0	モジュールスタンバイ設定を実施しません。
		1	モジュールスタンバイ設定を実施します。

2.3.2 全体の流れ

CS+を使用した場合の Startup 処理の全体の流れを図 2.1、図 2.2 に示します[注1]。



※1：ENABLE_CLOCK_GEARUP に 1 が設定されている場合のみ

※2：ENABLE_MODULE_STANDBY_SET に 1 が設定されている場合のみ

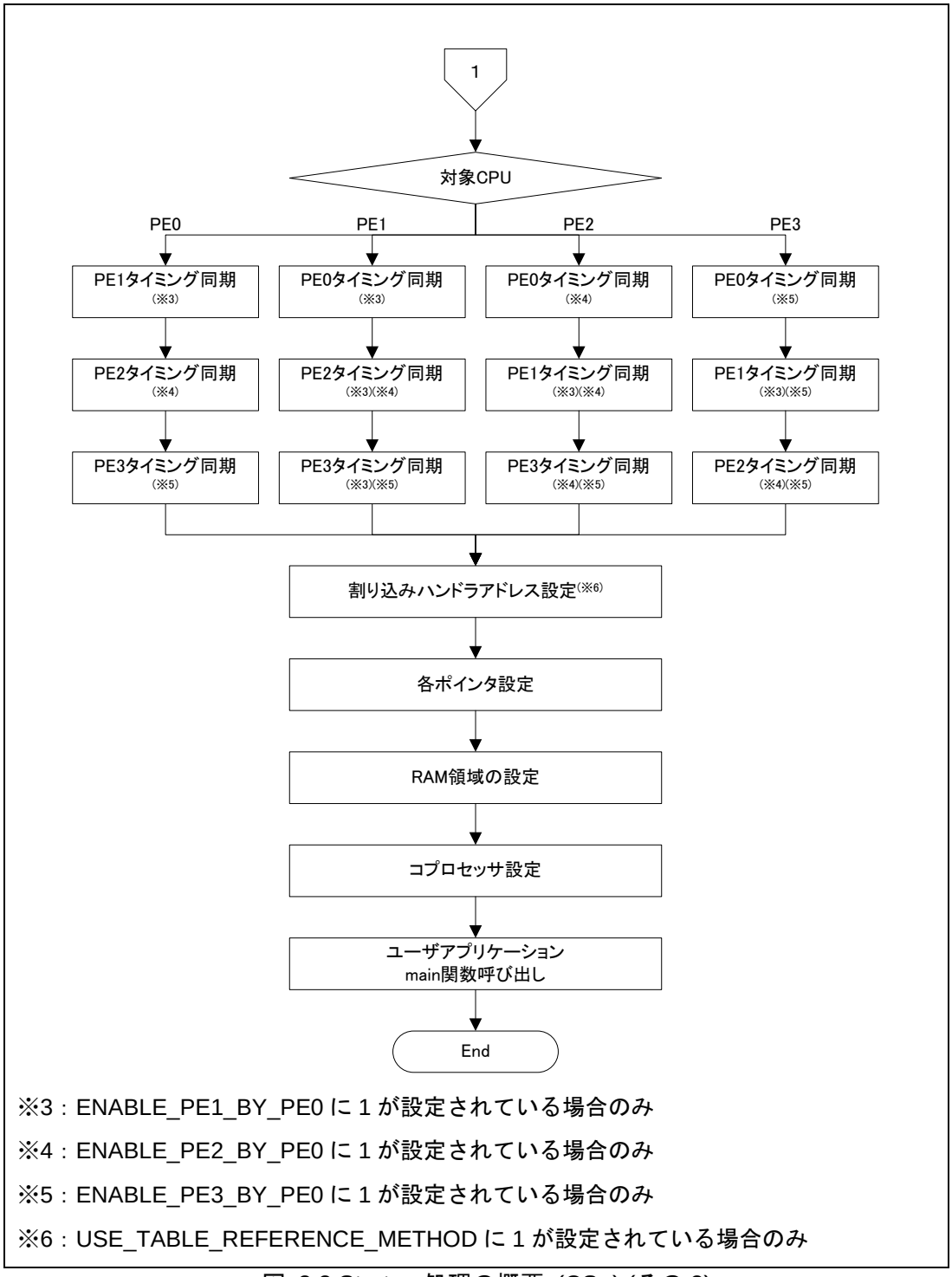
※3：ENABLE_PE1_BY_PE0 に 1 が設定されている場合のみ

※4：ENABLE_PE2_BY_PE0 に 1 が設定されている場合のみ

※5：ENABLE_PE3_BY_PE0 に 1 が設定されている場合のみ

図 2.1 Startup 処理の概要 (CS+) (その 1)

注1 初期停止コア(PE1,2,3)の、初期状態(初期停止状態、起動状態)はデバッグ・ツールにより設定されます。初期停止コアのデバッグについての詳細は、使用するエミュレーターのユーザーズマニュアル及び別冊、エミュレータデバッガのマニュアルおよびヘルプをご参照ください。



※3：ENABLE_PE1_BY_PE0 に 1 が設定されている場合のみ

※4：ENABLE_PE2_BY_PE0 に 1 が設定されている場合のみ

※5：ENABLE_PE3_BY_PE0 に 1 が設定されている場合のみ

※6：USE_TABLE_REFERENCE_METHOD に 1 が設定されている場合のみ

図 2.2 Startup 処理の概要 (CS+) (その 2)

2.3.3 処理

各処理の実装を表 2.7、表 2.8、表 2.9 に示します。

表 2.7 各処理の実装一覧(CS+) (その 1)

#	処理名	ラベル	実装ファイル	備考
1	PowerOn(リセット割り込み)	-	vecttbl0.asm vecttbl1.asm vecttbl2.asm vecttbl3.asm	割り込みベクタテーブルにて実装
2	レジスタ初期化	__start	boot.asm	
3	クロックギアアップ設定	_clock_gearup	boot.asm	処理 PE が PE 0 の場合のみ処理されます。(※1)
4	モジュールスタンバイ設定	_module_standby_set	boot.asm	処理 PE が PE 0 の場合のみ処理されます。(※2)
5	PE1～3 有効化	__start_PE0	boot.asm	処理 PE が PE 0 の場合のみ処理されます。(※3) (※4) (※5)
6	RAM 領域初期化	_hdwinit_PE0、 _hdwinit_PE1、 _hdwinit_PE2、 _hdwinit_PE3	boot.asm	処理 PE が PE0 の場合は _hdwinit_PE0、 処理 PE が PE1 の場合は _hdwinit_PE1、 処理 PE が PE2 の場合は _hdwinit_PE2、 処理 PE が PE3 の場合は _hdwinit_PE3 で処理されます。(※3) (※4) (※5)
7	PE0～3 タイミング同期	_hdwinit_PE0、 _hdwinit_PE1、 _hdwinit_PE2、 _hdwinit_PE3	boot.asm	処理 PE が PE0 の場合は _hdwinit_PE0、 処理 PE が PE1 の場合は _hdwinit_PE1、 処理 PE が PE2 の場合は _hdwinit_PE2、 処理 PE が PE3 の場合は _hdwinit_PE3 で処理されます。(※3) (※4) (※5)

※1：処理 PE は PEID bit0:2(PEID)で判断します。
ENABLE_CLOCK_GEARUP が 0 の場合、処理は行われません。

※2：処理 PE は PEID bit0:2(PEID)で判断します。
ENABLE_MODULE_STANDBY_SET が 0 の場合、処理は行われません。

※3：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE1_BY_PE0 が 0 の場合は PE1 の処理は行われません。

※4：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE2_BY_PE0 が 0 の場合は PE2 の処理は行われません。

※5：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE3_BY_PE0 が 0 の場合は PE3 の処理は行われません。

表 2.8 各処理の実装一覧(CS+) (その 2)

#	処理名	ラベル	実装ファイル	備考
8	割り込みハンドラアドレス設定	_init_eiint	boot.asm	
9	各ポインタ設定	__start_pm0、 __start_pm1、 __start_pm2、 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	処理 PE が PE0 の場合は __start_pm0、 処理 PE が PE1 の場合は __start_pm1、 処理 PE が PE2 の場合は __start_pm2、 処理 PE が PE3 の場合は __start_pm3 で処理されます。(※3) (※4) (※5)
10	RAM 領域の設定	__start_pm0、 __start_pm1、 __start_pm2、 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	処理 PE が PE0 の場合は __start_pm0、 処理 PE が PE1 の場合は __start_pm1、 処理 PE が PE2 の場合は __start_pm2、 処理 PE が PE3 の場合は __start_pm3 で処理されます。(※3) (※4) (※5)
11	コプロセッサ設定	__start_pm0、 __start_pm1、 __start_pm2、 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	処理 PE が PE0 の場合は __start_pm0、 処理 PE が PE1 の場合は __start_pm1、 処理 PE が PE2 の場合は __start_pm2、 処理 PE が PE3 の場合は __start_pm3 で処理されます。(※3) (※4) (※5)

※3：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE1_BY_PE0 が 0 の場合は PE1 の処理は行われません。

※4：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE2_BY_PE0 が 0 の場合は PE2 の処理は行われません。

※5：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE3_BY_PE0 が 0 の場合は PE3 の処理は行われません。

表 2.9 各処理の実装一覧(CS+) (その 3)

#	処理名	ラベル	実装ファイル	備考
12	ユーザアプリケーション main 関数呼び出し	__start_pm0、 __start_pm1、 __start_pm2、 __start_pm3	cstart0.asm cstart1.asm cstart2.asm cstart3.asm	処理 PE が PE0 の場合は __start_pm0、 処理 PE が PE1 の場合は __start_pm1、 処理 PE が PE2 の場合は __start_pm2、 処理 PE が PE3 の場合は __start_pm3 で処理されます。(※3) (※4) (※5)

※3：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE1_BY_PE0 が 0 の場合は PE1 の処理は行われません。

※4：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE2_BY_PE0 が 0 の場合は PE2 の処理は行われません。

※5：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE3_BY_PE0 が 0 の場合は PE3 の処理は行われません。

2.3.4 各処理詳細

各処理の詳細を説明します。

特に記載が無い場合は、PE0～PE3(注1)の全てについて同様の処理が行われます。

2.3.4.1 PowerOn(リセット割り込み)

PowerOn 時、RESET ベクタ(RESET セクション)のアドレスに遷移します。

RESET ベクタにより__start 処理に遷移します。

デバイス自体の PowerOn 時は PE0 が起動します。PE0 の処理(0 参照)にて ENABLE_PE1_BY_PE0 が 1 の場合には PE1 を、ENABLE_PE2_BY_PE0 が 1 の場合には PE2 を、ENABLE_PE3_BY_PE0 が 1 の場合には PE3 を有効化します。

プログラムコード例を図 2.3 に示します。

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

図 2.3 PowerOn(リセット割り込み)プログラムコード例(CS+)

(注1) PE1 は ENABLE_PE1_BY_PE0 が 1 の場合のみ有効になります。
PE2 は ENABLE_PE2_BY_PE0 が 1 の場合のみ有効になります。
PE3 は ENABLE_PE3_BY_PE0 が 1 の場合のみ有効になります。

2.3.4.2 レジスタ初期化

初期化するレジスタの一覧を表 2.10、表 2.11 に示します。設定値は例であり、システムによって最適な値で初期化してください。

表 2.10 初期化レジスター覧(CS+) (その 1)

#	レジスタ分類	レジスタ名	設定値例	説明
1	プログラムレジスタ	r1～r31	0	
2	基本システムレジスタ	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：E1 レベル例外受付禁止 CU2-0=1：FPU 有効化
13	FPU 機能レジスタ	FPSR	0x00220000	リセット後の値
14		FPEPC	0	
15		FPST	0	
16		FPCC	0	
17	MPU 機能レジスタ	MCA	0	
18		MCS	0	
19		MCR	0	
20		MPLA 注 1	0	
21		MPUA 注 1	0	
22		MPAT 注 1	0	
23		MPID0	0	
24		MPID1	0	
25		MPID2	0	
26		MPID3	0	
27		MPID4	0	
28		MPID5	0	

注 1 MPU の 32 保護領域の設定を、それぞれ初期化する必要があります。MPIDX レジスタ 0~31 に対応する MPLA, MPUA, MPAT レジスタをそれぞれ設定してください。

表 2.11 初期化レジスター覧(CS+) (その 2)

#	レジスタ分類	レジスタ名	設定値例	説明
29	MPU 機能レジスタ	MPID6	0	
30		MPID7	0	
31		MCI	0	
32	キャッシュ操作機能レジスタ	ICTAGL	0	
33		ICTAGH	0	
34		ICDATL	0	
35		ICDATH	0	
36		ICERR	0	
37	仮想サポート機能システムレジスタ	HVSB	0	
38	ゲストコンテキストレジスタ	GMEIPC	0	
39		GMFEPC	0	
40		GMEBASE	0	
41		GMINTBP	0	
42		GMEIWR	0	
43		GMFEWR	0	
44		GMMEA	0	
45		GMMEI	0	

プログラムコード例を図 2.4 に示します。

```
    $nowarning
    mov     r0, r1
    $warning
    mov     r0, r2
(中略)
    ldsr    r0, 0, 0            ; SR0,0   EIPC
    ldsr    r0, 2, 0            ; SR2,0   FEPC
    ldsr    r0, 16, 0           ; SR16,0 CTPC
(中略)
    mov     0x00010020, r10
    ldsr    r10, 5, 0           ; SR5,0   PSW
(後略)
```

図 2.4 レジスタ初期化プログラムコード例(CS+)

2.3.4.3 クロックギアアップ設定

PE0 の起動後、システムクロックを PLL に変更してクロックギアアップを実施します。

本処理は以下の条件を全て満たす場合のみ実行されます。

- ENABLE_CLOCK_GEARUP が 1
- 処理 PE が PE0(PEID bit0:2(PEID)=0)
- Main OSC と PLL が有効(PLLS=0x00000003)

参考としてクロックギアアップの設定方法例を図 2.5、図 2.6 に示します。

```
.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 ・・・
(中略)
      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
```

図 2.5 クロックギアアップ設定プログラムコード例(CS+) (その 1)

```
      andi      0x1, r2, r0             ; confirm that the value of CKSC_CPUS・・・
      bnz       L.clock_gearup.4        ; if the CKSC_CPUS.CPUCLKSACT is・・・
(中略)
      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 ・・・
(中略)
      mov       0xA5A5A500, r2          ; set 0xA5A5A500 in CLKKCPROT1 for・・・
      st.w      r2, 0xff980700[r0]      ; set CLKKCPROT1
```

図 2.6 クロックギアアップ設定プログラムコード例(CS+) (その 2)

2.3.4.4 モジュールスタンバイ設定

使用する機能に応じたモジュールスタンバイレジスタの設定（使用機能へのクロック供給の有無の設定）を行います。

本処理は以下の条件を満たす場合のみ実行されます。

・ENABLE_MODULE_STANDBY_SET が 1

・処理 PE が PE0(PEID bit0:2(PEID)=0)

参考として RS-CANFD のモジュールスタンバイレジスタの設定方法例を図 2.7 に示します。

```
      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
```

図 2.7 RS--CANFD モジュールスタンバイレジスタ設定プログラムコード例(CS+)

2.3.4.5 PE1～3 有効化

PE1～3 を有効化します。有効化する PE に応じた BOOTCTRL (PE1 bit1(BC1), PE2 bit2(BC2), PE3 bit3(BC3))を 1 に設定します。

本処理は以下の条件を全て満たす場合のみ実行されます。

- PE1 を有効化する場合、ENABLE_PE1_BY_PE0 が 1
- PE2 を有効化する場合、ENABLE_PE2_BY_PE0 が 1
- PE3 を有効化する場合、ENABLE_PE3_BY_PE0 が 1
- 処理 PE が PE0(PEID bit0:2(PEID)=0)

プログラムコード例を図 2.8 に示します。

```
    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
(中略)
    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
(中略)
    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
```

図 2.8 PE1～3 有効化プログラムコード例(CS+)

2.3.4.6 RAM 領域初期化

Local RAM および Cluster RAM の初期化を行います。

本プロジェクトでは、スタートアップ時間を短縮するため、PE0～3 の全ての PE が持ち回りで均等に、指定されたアドレスの RAM 領域の初期化処理を実施します。

PE0 にて初期化する RAM は以下のとおりです。(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)

PE0 にて初期化する RAM は以下のとおりです。(U2A6)

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

PE1 にて初期化する RAM は以下のとおりです。(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)

PE1 にて初期化する RAM は以下のとおりです。(U2A6)

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

PE2 にて初期化する RAM は以下のとおりです。(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)

PE3 にて初期化する RAM は以下のとおりです。(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)

一部の PE を起動しない場合、起動する PE 毎に均等に RAM 初期化が行われるように指定アドレスを調整してください。

プログラムコード例を図 2.9 に示します。

```
      ; local ram address
      LOCAL_RAM_CPU0_ADDR    .set      0xFDC00000
      LOCAL_RAM_CPU0_END     .set      0xFDC0FFFF
(中略)
      ; cluster ram address
      CLUSTER_RAM0_ADDR0     .set      0xFE000000
      CLUSTER_RAM0_END0      .set      0xFE03FFFF
(中略)
      ; 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
(中略)
_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
```

図 2.9 RAM 初期化プログラムコード例(PE0 用) (CS+)

2.3.4.7 PE0～3 タイミング同期

各 PE が一緒のタイミングで以降の処理に進めるように、処理が先行している PE が待機します。

各 PE が共に本処理まで到達したタイミングで以降の処理が実行されます。

本処理は以下の条件を満たす場合に実行されます。

- PE1 を有効化する場合、ENABLE_PE1_BY_PE0 が 1
- PE2 を有効化する場合、ENABLE_PE2_BY_PE0 が 1
- PE3 を有効化する場合、ENABLE_PE3_BY_PE0 が 1

PE0 における待機処理のプログラムコード例を図 2.10 に示します。

```
    CLUSTER_RAM2_ADDR       .set    0xFE400000
    CRAM_ADDR               .set    CLUSTER_RAM2_ADDR
(中略)
    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:
```

図 2.10 PE0～3 タイミング同期プログラムコード例(PE0 用) (CS+)

2.3.4.8 割り込みハンドラアドレス設定

テーブル参照方式のベースポインタアドレスを INTBP に設定します。

設定するベースポインタアドレスは、EIINTTBL セクションの先頭アドレスです。

直接ベクタ方式のベースアドレスは、レジスタ初期化(2.3.4.2)にて PSW:bit15(EBV)に 0 を設定しているため、RBASE の初期値（PE0,PE1 は 0x00000000、PE2,PE3 は 0x00800000）を使用します。

PSW:bit15(EBV)に 1 が設定された場合はレジスタ初期化にて EBASE に設定した 0 が使用されます。

本処理は以下の条件を全て満たす場合のみ実行されます。

・USE_TABLE_REFERENCE_METHOD が 1

プログラムコード例を図 2.11 に示します。

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

図 2.11 割り込みハンドラアドレス設定プログラムコード例(PE0 用) (CS+)

2.3.4.9 各ポインタ設定

スタックポインタ、グローバルポインタ、エレメントポインタを設定します。

以下に各ポインタに設定する値を示します。

・スタックポインタ

処理 PE が PE0(PEID bit0:2(PEID)=0)の場合は_stacktop_pm0 のアドレス

処理 PE が PE1(PEID bit0:2(PEID)=1)の場合は_stacktop_pm1 のアドレス

処理 PE が PE2(PEID bit0:2(PEID)=2)の場合は_stacktop_pm2 のアドレス

処理 PE が PE3(PEID bit0:2(PEID)=3)の場合は_stacktop_pm3 のアドレス

・グローバルポインタ(注1)

__gp_data(注2)の先頭アドレス

・エレメントポインタ(注3)

__ep_data(注4)の先頭アドレス

PE0 におけるプログラムコード例を図 2.12 に示します。

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

図 2.12 各ポインタ設定プログラムコード例(PE0 用) (CS+)

(注1) 全ての PE(PE0～3)にて処理されます。

(注2) __gp_data はリンカが自動的に作成するラベルです。

(注3) 全ての PE(PE0～3)にて処理されます。

(注4) __ep_data はリンカが自動的に作成するラベルです。

2.3.4.10 RAM 領域の設定

.data セクション(初期値あり RAM セクション)、.bss セクション(初期値なし RAM セクション)を __INITSCT_RH 処理(注1)にて初期化します。

__INITSCT_RH 処理は、パラメータレジスタ(r6、r7)に初期値あり RAM セクション初期化テーブルの先頭/末尾アドレス、パラメータレジスタ(r8、r9)に初期値なし RAM セクション初期化テーブルの先頭/末尾アドレスを設定した状態で呼び出すことにより、.data セクションには ROM からデータがコピーされ、.bss セクションは 0 クリアされます。

初期値あり RAM セクション初期化テーブルは、.INIT_DSEC.const セクションに配置する必要があり、テーブルにはコピー元 ROM セクション先頭アドレス、コピー元 ROM セクション末尾アドレス、コピー先 RAM セクション先頭アドレスを設定します。

初期値なし RAM セクション初期化テーブルは、.INIT_BSEC.const セクションに配置する必要があり、テーブルにはクリア対象 RAM セクション先頭アドレス、クリア対象 RAM セクション末尾アドレスを設定します。

プログラムコード例を図 2.13 に示します。

```
      ; 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
(中略)
      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
```

図 2.13 .data セクション、.bss セクション初期化プログラムコード例(CS+)

DTSRAM や MMCA RAM といった一部の RAM はハードウェアの RAM Initialization 機能により初期化されます。RAM Initialization 機能の詳細についてはデバイスのユーザーズマニュアルを参照ください。

(注1) __INITSCT_RH ルーチンはコンパイラから提供されるルーチンです。

2.3.4.11 コプロセッサ設定

FEPSW bit16(CU0)を 1 に設定し、FPU を有効化します。

FPU が必要ない場合は、PSW bit16(CU0)に 0 を設定してください。

プログラムコード例を図 2.14 に示します。

```
    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
(中略)
    feret                       ; apply PSW and PC
```

図 2.14 コプロセッサ設定プログラムコード例(CS+)

2.3.4.12 ユーザアプリケーション main 関数呼び出し

ユーザアプリケーションの main 関数として、以下の関数に遷移します。

処理 PE が PE0(PEID bit0:2(PEID)=0)の場合は_main_pm0

処理 PE が PE1(PEID bit0:2(PEID)=1)の場合は_main_pm1

処理 PE が PE2(PEID bit0:2(PEID)=2)の場合は_main_pm2

処理 PE が PE3(PEID bit0:2(PEID)=3)の場合は_main_pm3

PE0 におけるユーザアプリケーション main 関数呼び出しプログラムコード例を図 2.15 に示します。

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

    feret                       ; apply PSW and PC
```

図 2.15 ユーザアプリケーション main 関数呼び出しプログラムコード例(PE0 用) (CS+)

3. MULTI

本章では、統合開発環境として MULTI を使用した場合の Startup 処理について説明します。

3.1 Startup 関連ファイル

Startup に関連するファイルの一覧を表 3.1 に示します。

表 3.1 Startup 関連ファイル一覧(MULTI)

#	ファイル	ディレクトリ	説明
1	startup.850(※1)	プロジェクトルート/src/	PE0/PE1 スタートアップ処理呼び出し、ベクタテーブル
2	startup_PE0.850(※1)	プロジェクトルート/src/	PE0 スタートアップ処理、ベクタテーブル
3	startup_PE1.850(※1)	プロジェクトルート/src/	PE1 スタートアップ処理、ベクタテーブル
4	startup2.850(※1)	プロジェクトルート/src/	PE2/PE3 スタートアップ処理呼び出し、ベクタテーブル
5	startup_PE2.850(※1)	プロジェクトルート/src/	PE2 スタートアップ処理、ベクタテーブル
6	startup_PE3.850(※1)	プロジェクトルート/src/	PE3 スタートアップ処理、ベクタテーブル
7	main.c	プロジェクトルート/src/	メイン処理
8	main_pe0.c(※1)	プロジェクトルート/src/	PE0 メイン処理
9	main_pe1.c(※1)	プロジェクトルート/src/	PE1 メイン処理
10	main_pe2.c(※1)	プロジェクトルート/src/	PE2 メイン処理
11	main_pe3.c(※1)	プロジェクトルート/src/	PE3 メイン処理
12	section.ld(※1)	プロジェクトルート/	セクション設定

※1：ファイル名は任意に設定できます。
ファイル名を変更する場合は、プロジェクトファイルも修正してください。

3.2 セクション設定

Startup に関連する主なセクションの例を表 3.2 に示します。

表 3.2 Startup 関連セクション(MULTI)

#	セクション	割り当てデータ
1	.cintvect	PE0/PE1 リセットベクタテーブル
2	.coldboot	PE0/PE1 ブートコントローラ
3	.intvect_PE0	PE0 割り込みベクタテーブル
4	.intvect_PE1	PE1 割り込みベクタテーブル
5	.rozdata	定数データ
6	.robase	
7	.rosdata	
8	.rodata	
9	.text	
10	.ascet_const	
11	.mytext0	
12	.fixaddr	
13	.fixtype	
14	.secinfo	
15	.syscall	
16	.romdata	初期値ありデータ(ROM)
17	.romsldata	
18	.cintvect2	PE2/PE3 リセットベクタテーブル
19	.coldboot2	PE2/PE3 ブートコントローラ
20	.intvect_PE2	PE2 割り込みベクタテーブル
21	.intvect_PE3	PE3 割り込みベクタテーブル
22	.romdata	初期値ありデータ(ROM)
23	.data	初期値ありデータ(RAM)
24	.bss	初期値なしデータ(RAM)
25	.sdabase	SDA(スモールデータエリア)ベースレジスタ
26	.stack	スタック領域
27	.heapbase	ヒープ領域ベースアドレス
28	.heap	ヒープ領域

デバイスやプロジェクトによっては名称が異なる、一部セクションが必要ない、または他のセクションが必要である場合もあります。セクションの詳細については、MULTI やデバイスのユーザーズマニュアルを参照ください。

3.2.1 セクション指定方法

アセンブラや C 言語で、プログラム中でセクションの指定や追加等を行う際のセクション指定方法を以下に示します。

セクションを追加した場合、以下の指定の他に、section.ld も修正する必要があります。

・アセンブラ

.section *セクション名* [,"*属性*"] [> *配置アドレス*]

セクション名：セクションの名前を指定します。

属性：属性を指定します。

指定可能な属性を表 3.3 に示します。

複数指定する場合はダブルクォーテーション内で続けて指定します。

例："ab"

配置アドレス：セクションを配置するアドレスを指定します。

表 3.3 .section で指定可能な属性一覧

#	属性	説明
1	a	セクションがデバッグやシンボルの情報に使用されていない、割り当てられたメモリを持つことを意味します。
2	b	そのセクションが BSS セマンティックを持てることを意味します。.bss セクションでは、.word や.byte などの通常データ・擬似命令が許可されますが、これらの擬似命令で指定されたすべての値はアセンブラによって破棄されます。ELF 出力ファイルにセクションのサイズのみ記録され、セクションの内容は省略されます。セクションがターゲットにダウンロードされると、そのセクションに対して領域が割り当てられますが、そのセクションにデータはダウンロードされません。その代わりに、スタートアップ・コードによってそのセクション内のすべてのバイトがゼロに初期化されます。
3	w	セクションは書き込み可能です。
4	z	セクションに実行可能コードが含まれています。

・C 言語

#pragma ghs section [*セクション種別*=”*セクション名*”]

セクション種別：割り付けを変更するセクションの種別を指定します。

セクション名：セクションの名前を指定します。

指定可能なセクション種別を表 3.4 に示します。

表 3.4 #pragma ghs section で指定可能なセクション種別一覧

#	セクション種別	デフォルトのプログラムセクション	備考
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 処理

3.3.1 全体の流れ

MULTI を使用した場合の Startup 処理の全体の流れを図 3.1、図 3.2 に示します[注1]。

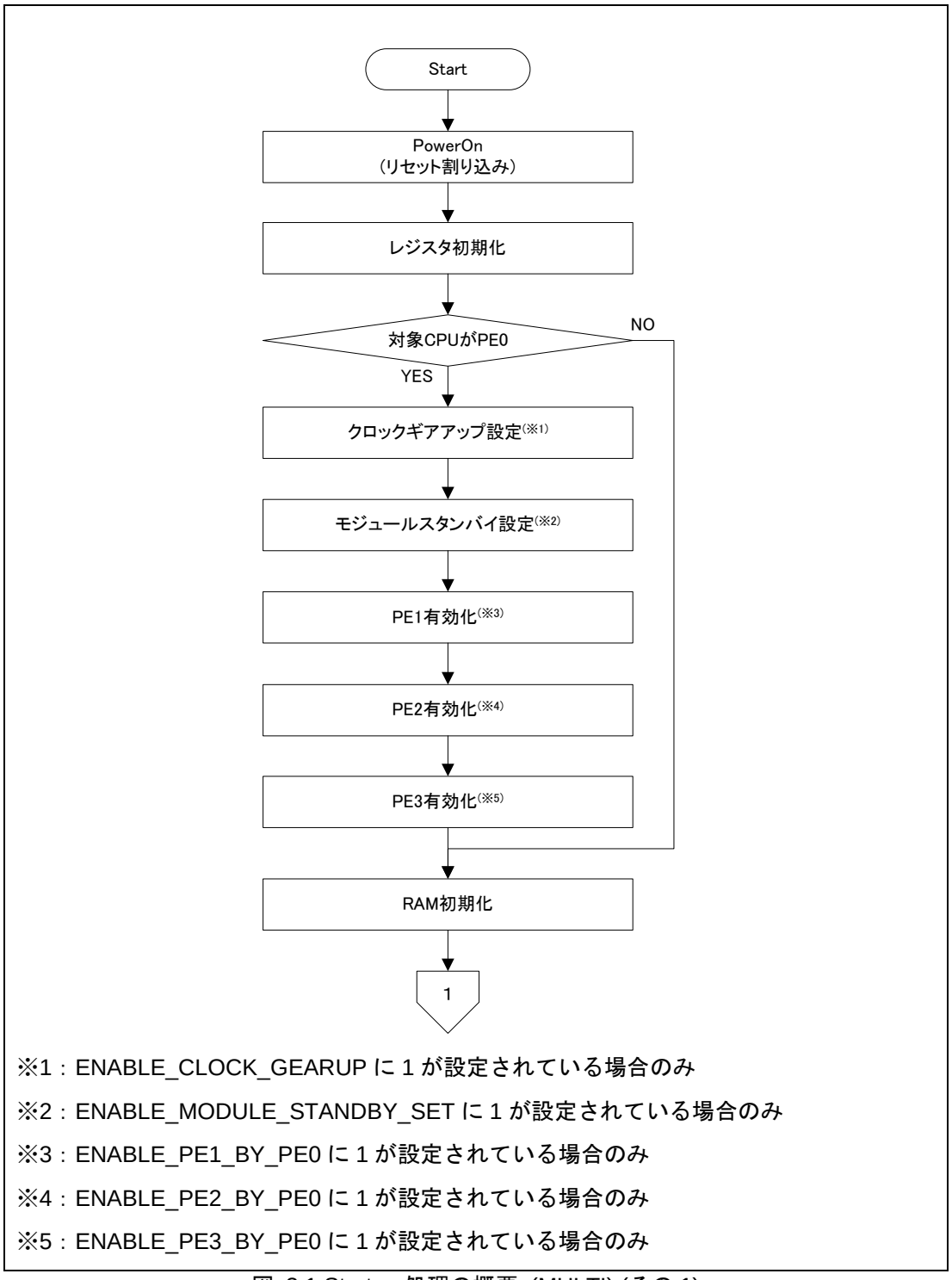
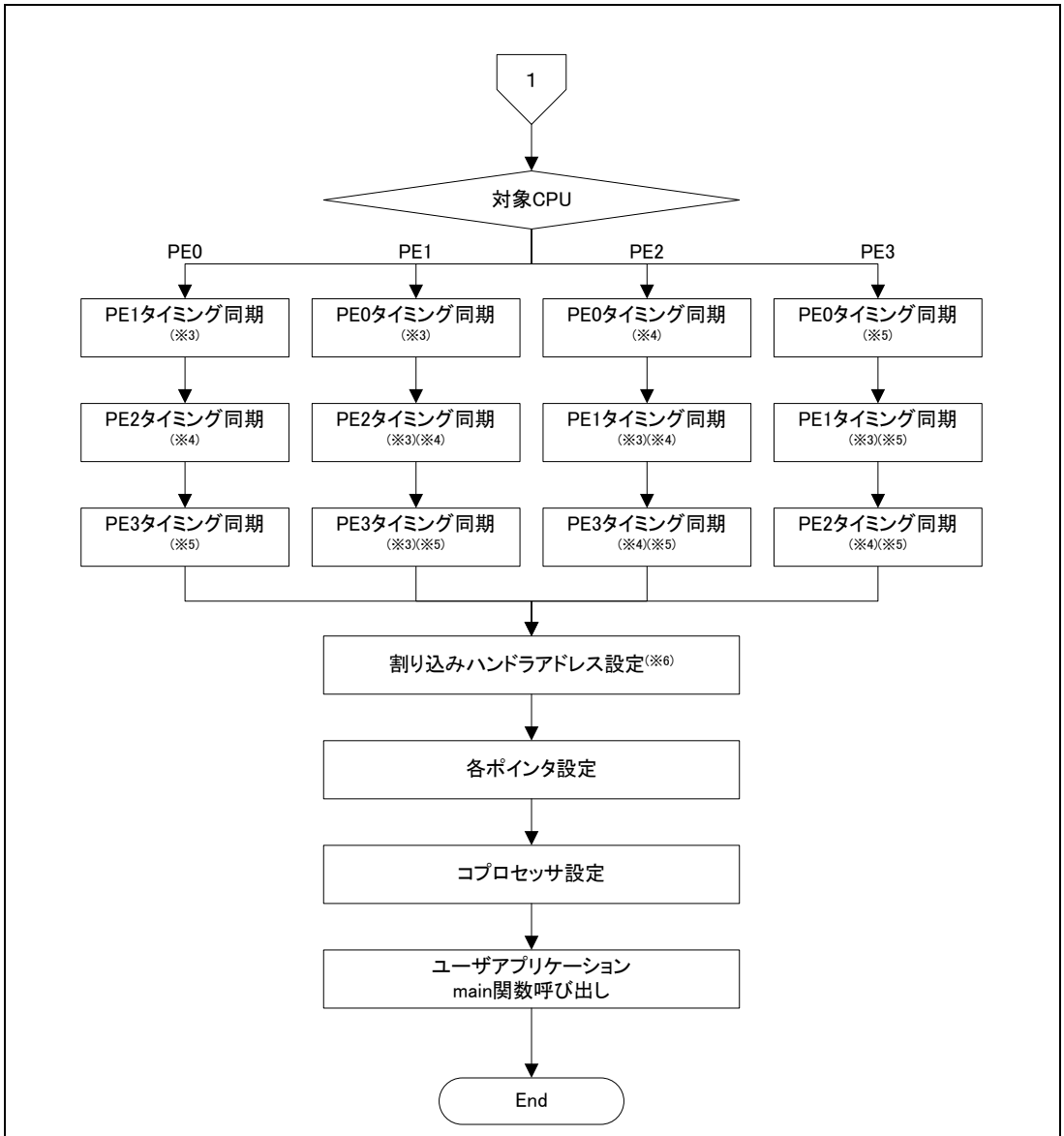


図 3.1 Startup 処理の概要 (MULTI) (その 1)

注1 初期停止コア(PE1,2,3)の、初期状態(初期停止状態、起動状態)はデバッグ・ツールにより設定されます。初期停止コアのデバッグについての詳細は、使用するエミュレーターのユーザーズマニュアル及び別冊、エミュレータデバッガのマニュアルおよびヘルプをご参照ください。



※3：ENABLE_PE1_BY_PE0 に 1 が設定されている場合のみ

※4：ENABLE_PE2_BY_PE0 に 1 が設定されている場合のみ

※5：ENABLE_PE3_BY_PE0 に 1 が設定されている場合のみ

※6：USE_TABLE_REFERENCE_METHOD に 1 が設定されている場合のみ

図 3.2 Startup 処理の概要 (MULTI) (その 2)

3.3.2 処理

各処理の実装を表 3.5 表 3.6 表 3.7 に示します。

表 3.5 各処理の実装一覧(MULTI) (その 1)

#	処理名	ラベル	実装ファイル	備考
1	PowerOn(リセット割り込み)	-	startup.850、 startup2.850	PE0/PE1 の処理は startup.850 のベクタテーブル、PE2/PE3 の処理は startup2.850 のベクタテーブルに実装されます。
2	レジスタ初期化	_RESET、 _RESET2	startup.850、 startup2.850	PE0/PE1 の処理は startup.850、PE2/PE3 の処理は startup2.850 に実装されます。
3	クロックギアアップ設定	_clock_gearup	startup.850	処理 PE が PE 0 の場合のみ処理されます。(※1)
4	モジュールスタンバイ設定	_module_standby_set	startup.850	処理 PE が PE 0 の場合のみ処理されます。(※2)
5	PE1～3 有効化	__start_PE0	startup.850	処理 PE が PE 0 の場合のみ処理されます。(※3) (※4) (※5)
6	RAM 領域初期化	__start_PE0、 __start_PE1、 __start_PE2、 __start_PE3	startup.850、 startup2.850	PE0/PE1 の処理は startup.850、PE2/PE3 の処理は startup2.850 に実装されます。 処理 PE が PE0 の場合は _hdwinit_PE0、 処理 PE が PE1 の場合は _hdwinit_PE1、 処理 PE が PE2 の場合は _hdwinit_PE2、 処理 PE が PE3 の場合は _hdwinit_PE3、 で処理されます。(※3) (※4) (※5)

※1：処理 PE は PEID bit0:2(PEID)で判断します。
ENABLE_CLOCK_GEARUP が 0 の場合、処理は行われません。

※2：処理 PE は PEID bit0:2(PEID)で判断します。
ENABLE_MODULE_STANDBY_SET が 0 の場合、処理は行われません。

※3：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE1_BY_PE0 が 0 の場合は PE1 の処理は行われません。

※4：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE2_BY_PE0 が 0 の場合は PE2 の処理は行われません。

※5：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE3_BY_PE0 が 0 の場合は PE3 の処理は行われません。

表 3.6 各処理の実装一覧(MULTI) (その 2)

#	処理名	ラベル	実装ファイル	備考
7	PE0～3 タイミング同期	__start_PE0、 __start_PE1、 __start_PE2、 __start_PE3	startup.850、 startup2.850	PE0/PE1 の処理は startup.850、PE2/PE3 の処理は startup2.850 に実装されます。 処理 PE が PE0 の場合は _hdwinit_PE0、 処理 PE が PE1 の場合は _hdwinit_PE1、 処理 PE が PE2 の場合は _hdwinit_PE2、 処理 PE が PE3 の場合は _hdwinit_PE3、 で処理されます。(※3) (※4) (※5)
8	割り込みハンドラアドレス設定	_init_eiint、 _init_eiint2	startup.850、 startup2.850	PE0、PE1 の処理は startup.850、に、PE2、PE3 の処理は startup2.850 に実装されます。
9	各ポインタ初期化	_RESET_PE0、 _RESET_PE1、 _RESET_PE2、 _RESET_PE3	startup_PE0.850、 startup_PE1.850、 startup_PE2.850、 startup_PE3.850	処理 PE が PE0 の場合は _RESET_PE0、 処理 PE が PE1 の場合は _RESET_PE1、 処理 PE が PE2 の場合は _RESET_PE2、 処理 PE が PE3 の場合は _RESET_PE3 で処理されます。(※3) (※4) (※5)
10	コプロセッサ設定	_RESET_PE0、 _RESET_PE1、 _RESET_PE2、 _RESET_PE3	startup_PE0.850、 startup_PE1.850、 startup_PE2.850、 startup_PE3.850	処理 PE が PE0 の場合は _RESET_PE0、 処理 PE が PE1 の場合は _RESET_PE1、 処理 PE が PE2 の場合は _RESET_PE2、 処理 PE が PE3 の場合は _RESET_PE3 で処理されます。(※3) (※4) (※5)

※3：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE1_BY_PE0 が 0 の場合は PE1 の処理は行われません。

※4：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE2_BY_PE0 が 0 の場合は PE2 の処理は行われません。

※5：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE3_BY_PE0 が 0 の場合は PE3 の処理は行われません。

表 3.7 各処理の実装一覧(MULTI) (その 3)

#	処理名	ラベル	実装ファイル	備考
11	ユーザアプリケーション main 関数呼び出し	_RESET_PE0、 _RESET_PE1、 _RESET_PE2、 _RESET_PE3	startup_PE0.850、 startup_PE1.850、 startup_PE2.850、 startup_PE3.850	処理 PE が PE0 の場合は _RESET_PE0、 処理 PE が PE1 の場合は _RESET_PE1、 処理 PE が PE2 の場合は _RESET_PE2、 処理 PE が PE3 の場合は _RESET_PE3 で処理されます。(※3) (※4) (※5)

※3：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE1_BY_PE0 が 0 の場合は PE1 の処理は行われません。

※4：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE2_BY_PE0 が 0 の場合は PE2 の処理は行われません。

※5：処理 PE は PEID bit0:2(PEID)で判断します。
ただし、ENABLE_PE3_BY_PE0 が 0 の場合は PE3 の処理は行われません。

3.3.3 各処理詳細

各処理の詳細を説明します。

3.3.3.1 PowerOn(リセット割り込み)

PowerOn 時、RESET ベクタ(RESET セクション)のアドレスに遷移します。

RESET ベクタにより、PE0/PE1 の場合は_RESET 処理に、PE2/PE3 の場合は_RESET2 処理に遷移します。

デバイス自体の PowerOn 時は PE0 が起動します。PE0 の処理(参照)にて ENABLE_PE1_BY_PE0 が 1 の場合には PE1 を、ENABLE_PE2_BY_PE0 が 1 の場合には PE2 を、ENABLE_PE3_BY_PE0 が 1 の場合には PE3 を有効化します。

プログラムコード例を図 3.3 に示します。

```
    .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
```

図 3.3 PowerOn(PE0/PE1 リセット割り込み)プログラムコード例(MULTI)

3.3.3.2 レジスタ初期化

初期化するレジスタの一覧を表 3.8、表 3.9 に示します。設定値は例であり、システムによって最適な値で初期化してください。

表 3.8 初期化レジスター覧(MULTI) (その 1)

#	レジスタ分類	レジスタ名	設定値例	説明
1	プログラムレジスタ	r1～r31	0	
2	基本システムレジスタ	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：E1 レベル例外受付禁止 CU2-0=1：FPU 有効化
13	FPU 機能レジスタ	FPSR	0x00220000	リセット後の値
14		FPEPC	0	
15		FPST	0	
16		FPCC	0	
17	MPU 機能レジスタ	MCA	0	
18		MCS	0	
19		MCR	0	
20		MPLA 注 1	0	
21		MPUA 注 1	0	
22		MPAT 注 1	0	
23		MPID0	0	
24		MPID1	0	
25		MPID2	0	
26		MPID3	0	
27		MPID4	0	
28		MPID5	0	

注 1 MPU の 32 保護領域の設定を、それぞれ初期化する必要があります。MPIDX レジスタ 0~31 に対応する MPLA, MPUA, MPAT レジスタをそれぞれ設定してください。

表 3.9 初期化レジスター覧(MULTI) (その 2)

#	レジスタ分類	レジスタ名	設定値例	説明
29	MPU 機能レジスタ	MPID6	0	
30		MPID7	0	
31		MCI	0	
32	キャッシュ操作機能レジスタ	ICTAGL	0	
33		ICTAGH	0	
34		ICDATL	0	
35		ICDATH	0	
36		ICERR	0	
37	仮想サポート機能システムレジスタ	HVSB	0	
38	ゲストコンテキストレジスタ	GMEIPC	0	
39		GMFEPC	0	
40		GMEBASE	0	
41		GMINTBP	0	
42		GMEIWR	0	
43		GMFEWR	0	
44		GMMEA	0	
45		GMMEI	0	

プログラムコード例を図 3.4 に示します。

```
   mov     r0, r1
   mov     r0, r2
(中略)
   ldsr    r0,    0, 0                  -- SR0,0    EIPC
   ldsr    r0,    2, 0                  -- SR2,0    FEPC
   ldsr    r0, 16, 0                    -- SR16,0 CTPC
(中略)
   mov     0x00010020, r10
   ldsr    r10, 5, 0                    -- SR5,0    PSW
(後略)
```

図 3.4 レジスタ初期化プログラムコード例(MULTI)

3.3.3.3 クロックギアアップ設定

PE0 の起動後、システムクロックを PLL に変更してクロックギアアップを実施します。

本処理は以下の条件を全て満たす場合のみ実行されます。

- ENABLE_CLOCK_GEARUP が 1
- 処理 PE が PE0(PEID bit0:2(PEID)=0)
- Main OSC と PLL が有効(PLLS=0x00000003)

参考としてクロックギアアップの設定方法例を図 3.5、図 3.6 に示します。

```
.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 ・・・
(中略)
   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
```

図 3.5 クロックギアアップ設定プログラムコード例(MULTI) (その 1)

```
   andi        0x1, r2, r0              -- confirm that the value of CKSC_CPUS・・・
   bnz         .L.clock_gearup.4        -- if the CKSC_CPUS.CPUCLKSACT is ・・・
(中略)
   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 ・・・
(中略)
   mov         0xA5A5A500, r2           -- set 0xA5A5A500 in CLKKCPROT1・・・
   st.w        r2, 0xff980700[r0]       -- set CLKKCPROT1
```

図 3.6 クロックギアアップ設定プログラムコード例(MULTI) (その 2)

3.3.3.4 モジュールスタンバイ設定

使用する機能に応じたモジュールスタンバイレジスタの設定（使用機能へのクロック供給の有無の設定）を行います。

本処理は以下の条件を満たす場合のみ実行されます。

・ENABLE_MODULE_STANDBY_SET が 1

・処理 PE が PE0(PEID bit0:2(PEID)=0)

参考として RS-CANFD のモジュールスタンバイレジスタの設定方法例を図 3.7 に示します。

```
   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
```

図 3.7 RS-CANFD モジュールスタンバイレジスタ設定プログラムコード例(MULTI)

3.3.3.5 PE1～3 有効化

PE1～3 を有効化します。有効化する PE に応じた BOOTCTRL (PE1 bit1(BC1), PE2 bit2(BC2), PE3 bit3(BC3))を 1 に設定します。

本処理は以下の条件を全て満たす場合のみ実行されます。

- PE1 を有効化する場合、ENABLE_PE1_BY_PE0 が 1
- PE2 を有効化する場合、ENABLE_PE2_BY_PE0 が 1
- PE3 を有効化する場合、ENABLE_PE3_BY_PE0 が 1
- 処理 PE が PE0(PEID bit0:2(PEID)=0)

プログラムコード例を図 3.8 に示します。

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

図 3.8 PE1～3 有効化プログラムコード例(MULTI)

3.3.3.6 RAM 領域初期化

Local RAM および Cluster RAM の初期化を行います。

本プロジェクトでは、スタートアップ時間を短縮するため、PE0～3 の全ての PE が持ち回りで均等に、指定されたアドレスの RAM 領域の初期化処理を実施します。

PE0 にて初期化する RAM は以下のとおりです。(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)

PE0 にて初期化する RAM は以下のとおりです。(U2A6)

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

PE1 にて初期化する RAM は以下のとおりです。(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)

PE1 にて初期化する RAM は以下のとおりです。(U2A6)

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

PE2 にて初期化する RAM は以下のとおりです。(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)

PE3 にて初期化する RAM は以下のとおりです。(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)

一部の PE を起動しない場合、起動する PE 毎に均等に RAM 初期化が行われるように指定アドレスを調整してください。

プログラムコード例を図 3.9 に示します。

```
    -- 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
  (中略)
_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
```

図 3.9 RAM 初期化プログラムコード例(PE0 用) (MULTI)

3.3.3.7 PE0～3 タイミング同期

各 PE が一緒のタイミングで以降の処理に進めるように、処理が先行している PE が待機します。

各 PE が共に本処理まで到達したタイミングで以降の処理が実行されます。

本処理は以下の条件を満たす場合に実行されます。

- PE1 を有効化する場合、ENABLE_PE1_BY_PE0 が 1
- PE2 を有効化する場合、ENABLE_PE2_BY_PE0 が 1
- PE3 を有効化する場合、ENABLE_PE3_BY_PE0 が 1

PE0 における待機処理のプログラムコード例を図 3.10 に示します。

```
.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:
```

図 3.10 PE0～3 タイミング同期プログラムコード例(PE0 用) (MULTI)

3.3.3.8 割り込みハンドラアドレス設定

テーブル参照方式のベースポインタアドレスを INTBP に設定します。

設定するベースポインタアドレスは、EIINTTBL セクションの先頭アドレスです。

直接ベクタ方式のベースアドレスは、レジスタ初期化(2.3.4.2)にて PSW:bit15(EBV)に 0 を設定しているため、RBASE の初期値（PE0,PE1 は 0x00000000、PE2,PE3 は 0x00800000）を使用します。

PSW:bit15(EBV)に 1 が設定された場合はレジスタ初期化にて EBASE に設定した 0 が使用されます。

本処理は以下の条件を全て満たす場合のみ実行されます。

・USE_TABLE_REFERENCE_METHOD が 1

プログラムコード例を図 3.11、図 3.12 に示します。

```
#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
```

図 3.11 割り込みハンドラアドレス設定プログラムコード例(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:
```

図 3.12 割り込みハンドラアドレス設定プログラムコード例(MULTI)

3.3.3.9 各ポインタ初期化

グローバルポインタ、テキストポインタ、スタックポインタを設定します。

以下に各ポインタに設定する値を示します。

・グローバルポインタ

.sdabase セクションの先頭アドレス

・テキストポインタ

.robase セクションの先頭アドレス

・スタックポインタ

.stack セクションの末尾アドレス

プログラムコード例を図 3.13 に示します。

```
-- 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
```

図 3.13 各ポインタ初期化プログラムコード例(MULTI)

3.3.3.10 コプロセッサ設定

FEPSW bit16(CU0)を 1 に設定し、FPU を有効化します。

FPU が必要ない場合は、PSW bit16(CU0)に 0 を設定してください。

プログラムコード例を図 3.14 に示します。

```
-- 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
```

図 3.14 コプロセッサ設定プログラムコード例(MULTI)

3.3.3.11 ユーザアプリケーション main 関数呼び出し

ユーザアプリケーションの main 関数として、main に遷移します。

プログラムコード例を図 3.15 に示します。

```
    jr          __start
```

図 3.15 ユーザアプリケーション main 関数呼び出しプログラムコード例(MULTI)

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
0.50	2019.04.15	全項	初版作成。
0.60	2019.11.06	16, 17	表 3.10、表 3.11 を修正。 （SCBP 追加、および各レジスタの順序を修正）
		18, 19	図 3.5、図 3.6 を修正。 （クロックギアアップ設定プログラムコード例(CS+)を修正）
		20	図 3.8 を修正。 （PE1～3 有効化プログラムコード例(CS+)を修正）
		22	図 3.9 を修正。 （RAM 初期化プログラムコード例(PE0 用) (CS+)を修正）
		27	図 3.13 を修正。 （.data セクション .bss セクション初期化プログラムコード例(CS+)を修正）
		28	図 3.14 を修正。 （コプロセッサ設定プログラムコード例(CS+)を修正）
		39, 40	表 4.8、表 4.9 を修正。 （SCBP 追加、および各レジスタの順序を修正）
		41, 42	図 4.5、図 4.6 を修正。 （クロックギアアップ設定プログラムコード例(MULTI)を修正）
		43	図 4.8 を修正。 （PE1～3 有効化プログラムコード例(MULTI)を修正）
		51	図 4.14 を修正。 （コプロセッサ設定プログラムコード例(MULTI)を修正）
0.70	2020.03.20	16,39	図 3.10、図 4.8 を修正 (SCBP,CTBP,MPCFG 削除)
		16,39	図 3.10、図 4.8 に注釈を追加 (MPLA、MPUA、MPAT)
1.00	2020.09.30	全頁	表と図の番号を対応するものに変更
		17.40	表 2.11、表 3.9 を修正 (GMFEPSW ,GMEIIC,GMFEIC 削除) (GMMEA, GMMEI 追加)
		18,41	2.3.4.3 / 3.3.4.3 クロックギアアップ設定 クロックギアアップ処理が実行される条件の修正
1.01	2021.03.18	1,3	対象製品に RH850/U2A8 を追加 1.1 ご使用上の注意を追加
	2021.03.18	10,33	初期停止コアについての注釈を追記
1.10	2022.01.31	1,3	対象製品に RH850/U2A6 を追加
		18,19,42,41	図 2.5, 図 2.6, 図 3.5, 図 3.6 クロックギアアップ設定プログラムコード例を修正
		19,42	図 2.7, 図 3.7 RS-CANFD モジュールスタンバイレジスタ設定プログラムコード例を修正
		21,44	U2A6 の初期化 RAM 領域を追加。U2A16 PE3 による Cluster RAM (Cluster2, Cluster3)の初期化範囲の修正

© 2022 Renesas Electronics Corporation. All rights reserved.

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 1. 静電気対策

CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

© 2022 Renesas Electronics Corporation. All rights reserved.

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2022 Renesas Electronics Corporation. All rights reserved.