RZ/T2ME グループ

ローダプログラムとアプリケーションプログラムのプロジェクト分割例

要旨

本アプリケーションノートでは、ローダプログラムとアプリケーションプログラムのプロジェクトを分割したサンプルプログラムについて説明します。

本サンプルプログラムの特徴を以下に示します。

- 本サンプルプログラムは xSPI0 ブートモード (x1 ブートシリアルフラッシュ)版、16 ビットバスブートモード (NOR フラッシュ)版の 2 つの動作モードに対応しています。
- 本サンプルプログラムにはローダプログラムとアプリケーションプログラムの 2 つのサンプルプログラムがあり、それぞれのプロジェクトに分割されています。
- ローダプログラムはアプリケーションプログラムを外付けフラッシュから内蔵 RAM や外部 RAM へコピーするためのプログラムです。ローダプログラム内に定義されているローダテーブルの情報(コピー元アドレス、コピー先アドレス、サイズ)に従ってアプリケーションプログラムのコピー処理を行います。
- アプリケーションプログラムはローダプログラムでコピーされるサンプルプログラムです。ローダプログラム終了後に動作し、本デバイスに必要な初期設定を行い、LED の点滅処理を行います。

動作確認デバイス

RZ/T2ME グループ

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. 仕様

1.1 動作環境

本アプリケーションノートのサンプルプログラムは、下記の環境を想定しています。

表 1.1 動作環境

項目	内容
使用マイコン	RZ/T2ME グループ (R9A07G075M29GBG)
動作周波数	CPU core0 : 800MHz (Arm® Cortex®-R52) CPU core1 : 800MHz (Arm® Cortex®-R52)注 1
動作電圧	3.3V / 1.8V / 1.1V
統合開発環境	• IAR システムズ製 Embedded Workbench® for Arm Version 9.60.2 + patch • Renesas 製 e² studio 2024-04
動作モード	• xSPI0 ブートモード(x1 シリアルフラッシュ) • 16 ビットバスブートモード(NOR フラッシュ)
使用ボード	Renesas Starter Kit+ for RZ/T2ME
Flexible Software Package (FSP)	Version 2.1.0 (RZ/T2 FSP)

【注】 1. CPU core1 を使用する場合は「5.3 CPU1 プログラムのデバッグ方法」を参照してください。

1.2 ファイル構成

本パッケージのファイル構成と内容物の詳細を下記に示します。

```
RZT2ME_loader_application
├──r01an7841jj0200-rzt2me.pdf
├──r01an7841ej0200-rzt2me.pdf
├──iccarm : EWARM 環境用
│   ├──16bitbusboot : NOR フラッシュメモリ用サンプルプログラム
│   │   └──Loader_application_projects
│   │       ├──application : アプリケーションプログラム用プロジェクト
│   │       ├──loader : ローダプログラム用プロジェクト
│   │       ├──cpu1 : CPU1 用プロジェクト
│   │       ├──settings : ワークスペースの設定ファイル
│   │       ├──multicore_setup.xml : マルチコアデバッグの設定ファイル
│   │       └──RZT2ME_bsp_16bitbusboot_app_loader.eww : EWARM ワークスペース
│   └──xspi0bootx1 : SPI フラッシュメモリ用サンプルプログラム
│       └──Loader_application_projects
│           ├──application : アプリケーションプログラム用プロジェクト
│           ├──loader : ローダプログラム用プロジェクト
│           ├──cpu1 : CPU1 用プロジェクト
│           ├──settings : ワークスペースの設定ファイル
│           ├──multicore_setup.xml : マルチコアデバッグの設定ファイル
│           └──RZT2ME_bsp_xspi0bootx1_app_loader.eww : EWARM ワークスペース
└──gcc : e2 studio 環境用
    ├──16bitbusboot : NOR フラッシュメモリ用サンプルプログラム
    │   └──Loader_application_projects.zip
    │       ├──RZT2ME_bsp_16bitbusboot_app : アプリケーションプログラム用プロジェクト
    │       ├──RZT2ME_bsp_16bitbusboot_loader : ローダプログラム用プロジェクト
    │       └──RZT2ME_cpu1 : CPU1 用プロジェクト
    └──xspi0bootx1 : SPI フラッシュメモリ用サンプルプログラム
        └──Loader_application_projects.zip
            ├──RZT2ME_bsp_xspi0bootx1_app : アプリケーションプログラム用プロジェクト
            ├──RZT2ME_bsp_xspi0bootx1_loader : ローダプログラム用プロジェクト
            └──RZT2ME_cpu1 : CPU1 用プロジェクト
```

本サンプルプログラムのパッケージは EWARM と e2 studio 環境で準備されており、それぞれの環境で NOR フラッシュメモリ用と SPI フラッシュメモリ用に別れています。

本サンプルプログラムはローダプログラムのプロジェクトとアプリケーションプログラムのプロジェクトと CPU1 のプロジェクトで構成されています。

各開発環境におけるサンプルプログラムの使用方法については、付録. 各開発環境における補足内容を参照してください。

1.3 ボードのスイッチ/ジャンパ設定

本サンプルプログラムを動作させるために必要なスイッチ/ジャンパ設定を下記に示します。各設定の詳細につきましては、Renesas Stater Kit+ for RZ/T2M, RZ/T2ME ユーザーズマニュアルを参照してください。

表 1.2 スイッチの設定

プロジェクト	SW4-1	SW4-2	SW4-3	SW4-4	SW4-5	SW6-1
16 ビットバスブートモード版	ON	OFF	ON	ON	OFF	ON
xSPI0 ブートモード版	ON	ON	ON	ON	OFF	-

表 1.3 ジャンパの設定

プロジェクト	CN8	CN17
16 ビットバスブートモード版	-	Short 1-2
xSPI0 ブートモード版	Short 2-3	-

2. ハードウェア説明

2.1 使用周辺機能

下記に本サンプルプログラムで使用する周辺機能と用途を示します。

表 2.1 使用する周辺機能と用途

項目	内容
クロック発生回路(CGC)	CPU クロックおよび各周辺モジュールクロックで使用
割り込みコントローラ(ICU)	ソフトウェア割り込み(INTCPU0)で使用
バスステートコントローラ(BSC)	CS0 空間に NOR フラッシュメモリを接続するために使用
拡張シリアルペリフェラルインタフェース(xSPI)	外部アドレス空間 xSPI0 にシリアルフラッシュメモリを接続するために使用
汎用入出力ポート	LED の点灯および消灯のための端子制御に使用

各周辺機能についての基本的な内容は、RZ/T2ME グループ・ユーザーズマニュアル ハードウェア編を参照してください。

2.2 使用端子一覧

表 2.2 に使用端子と機能を示します。

表 2.2 使用端子と機能

端子名	入出力	内容
A1 ~ A25	出力	NOR フラッシュメモリ
D0 ~ D15	入出力	NOR フラッシュメモリ
CS0#	出力	CS0 空間に接続された NOR フラッシュメモリへのデバイス選択信号出力
RD#	出力	リード中を示すストローブ信号出力
WE0#/DQMLL	出力	D15 ~ D8 に対するライトストローブ信号出力
WE1#/DQMLU	出力	D7 ~ D0 に対するライトストローブ信号出力
XSPI0_CKP	出力	クロック出力
XSPI0_CS0	出力	CS0 空間に接続された QSPI フラッシュメモリへのデバイス選択信号出力
XSPI0_RESET0	出力	マスタリセットステータス出力
XSPI0_IO0 ~ XSPI0_IO3	入出力	データ入出力
MD0	入力	動作モードの選択入力 • MD0 = “L”, MD1 = “L”, MD2 = “L” (xSPI0 ブートモード) • MD0 = “L”, MD1 = “H”, MD2 = “L” (16 ビットバスブートモード)
MD1	入力	
MD2	入力	
P19_6	出力	LED0 の点灯および消灯
P19_4	出力	LED1 の点灯および消灯
P20_0	出力	LED2 の点灯および消灯
P23_4	出力	LED3 の点灯および消灯

【注】 #は負論理(またはアクティブロー)を示す記号です。

3. ソフトウェア説明

以降、特に明記しない場合は EWARM(IAR システムズ社製)を使用した場合について説明を行います。

本書ではローダプロジェクトに含まれるプログラムをローダプログラム、アプリケーションプロジェクトに含まれるプログラムをアプリケーションプログラムと呼びます。また、各ローダ/アプリケーションプログラムはスタートアップ処理部とメイン処理部に分かれています。

3.1 動作概要

本デバイスは、リセット解除後のブート処理で、各動作モード(16 ビットバスブート/xSPI0 ブート)に対応する外付けフラッシュ(NOR フラッシュ/シリアルフラッシュ)メモリに格納されたローダプログラムを BTCM へコピーします。

ブート処理後、BTCM にコピーしたローダプログラムを実行します。ローダプログラムはアプリケーションプログラムを外付けフラッシュ(NOR フラッシュ/シリアルフラッシュ)メモリから内蔵 RAM(System SRAM)へコピーします。ローダプログラム終了後、コピーされたアプリケーションプログラムが RAM 上で動作します。

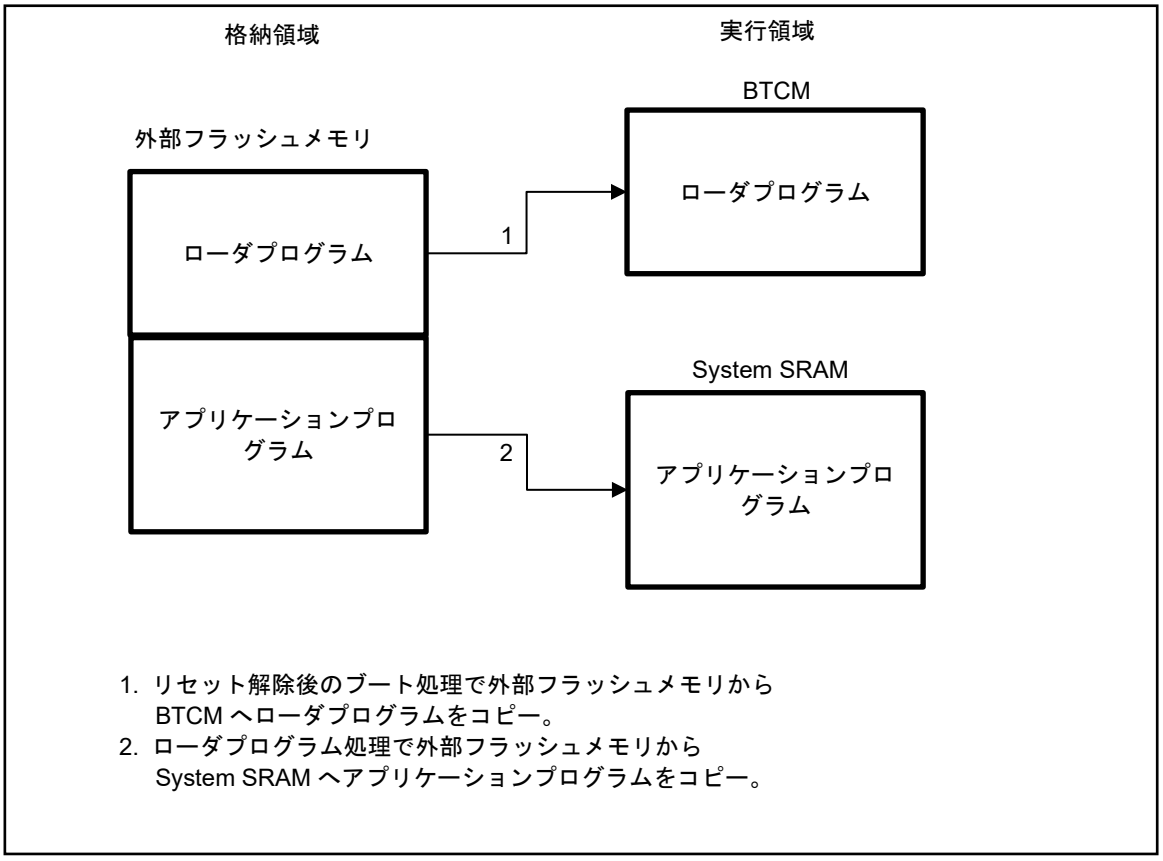

図 3-1 動作概要

3.1.1 ローダプログラム

ローダプログラムではスタートアップ処理として、Exception Level の変更、クロック設定などの初期設定を行います。その後、メイン処理が実行されます。メイン処理ではローダテーブルのパラメータに従って、外付けフラッシュ(NOR フラッシュ/シリアルフラッシュ)メモリに格納されたアプリケーションプログラムを System SRAM へコピーします。ローダテーブルはローダプログラムがアプリケーションプログラムをコピーする際に参照するテーブルです。ローダテーブルの詳細につきましては、「3.2 ローダテーブル」を参照してください。

また、コピー処理開始の合図として LED0 が点灯しコピー処理終了の合図として LED3 が点灯します。コピー処理終了後はアプリケーションプログラムに分岐します。

ローダプログラムの動作概要を図 3-2 に示します。

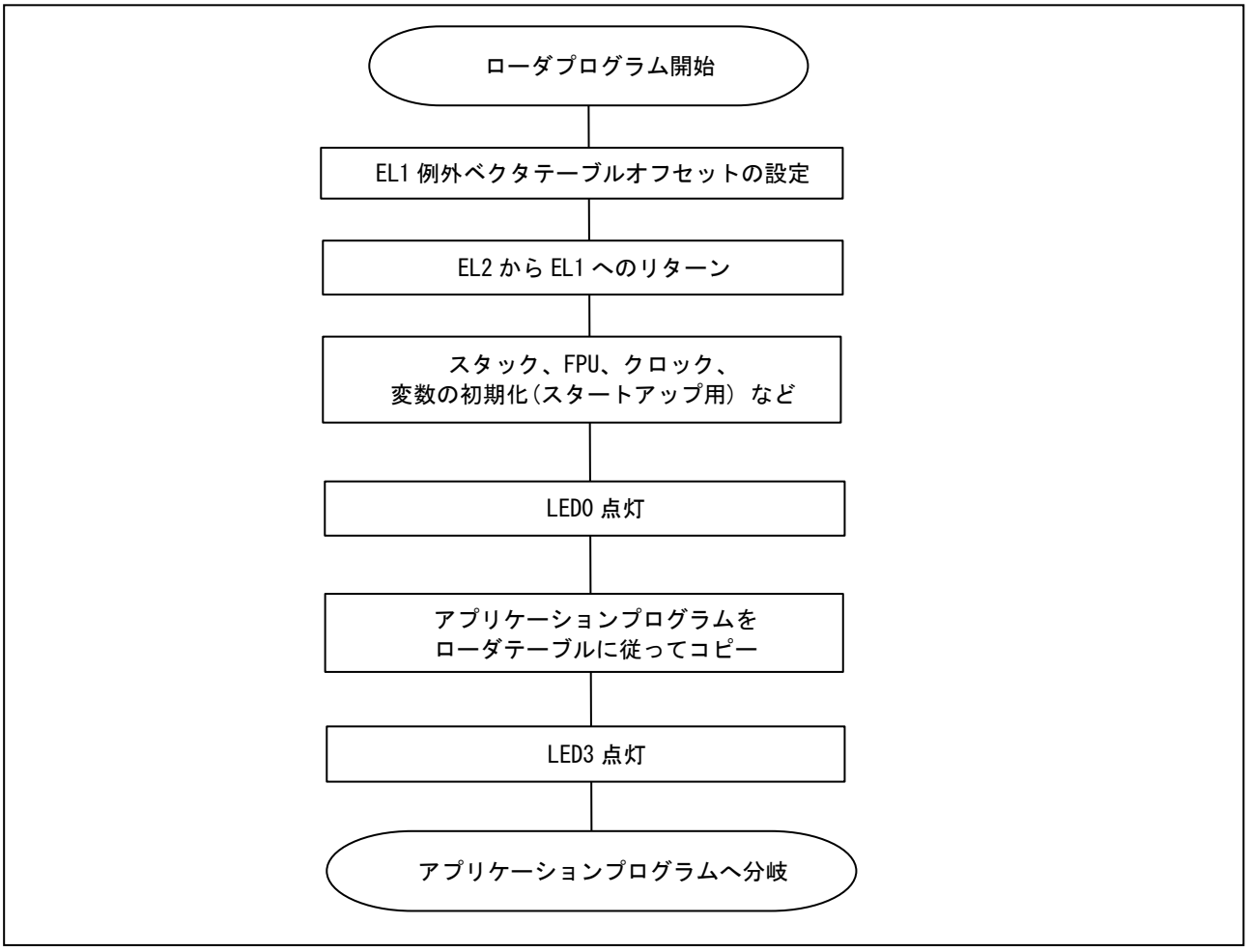

図 3-2 ローダプログラムの動作概要

3.1.2 アプリケーションプログラム

アプリケーションプログラムではスタートアップ処理として、クロック設定、ポート初期設定、割り込み設定などの初期設定を行います。ローダプログラム処理時に点灯した LED0 と LED3 はポート設定時に消灯します。その後、メイン処理部が実行されます。

System SRAM 上で動作するメイン処理では、LED の点滅処理が実行されます。LED の点滅処理はソフトウェア割り込み(INTCPU0)によって処理され、LED0 と LED1 が点滅します。

アプリケーションプログラムの動作概要を図 3-3 に示します。

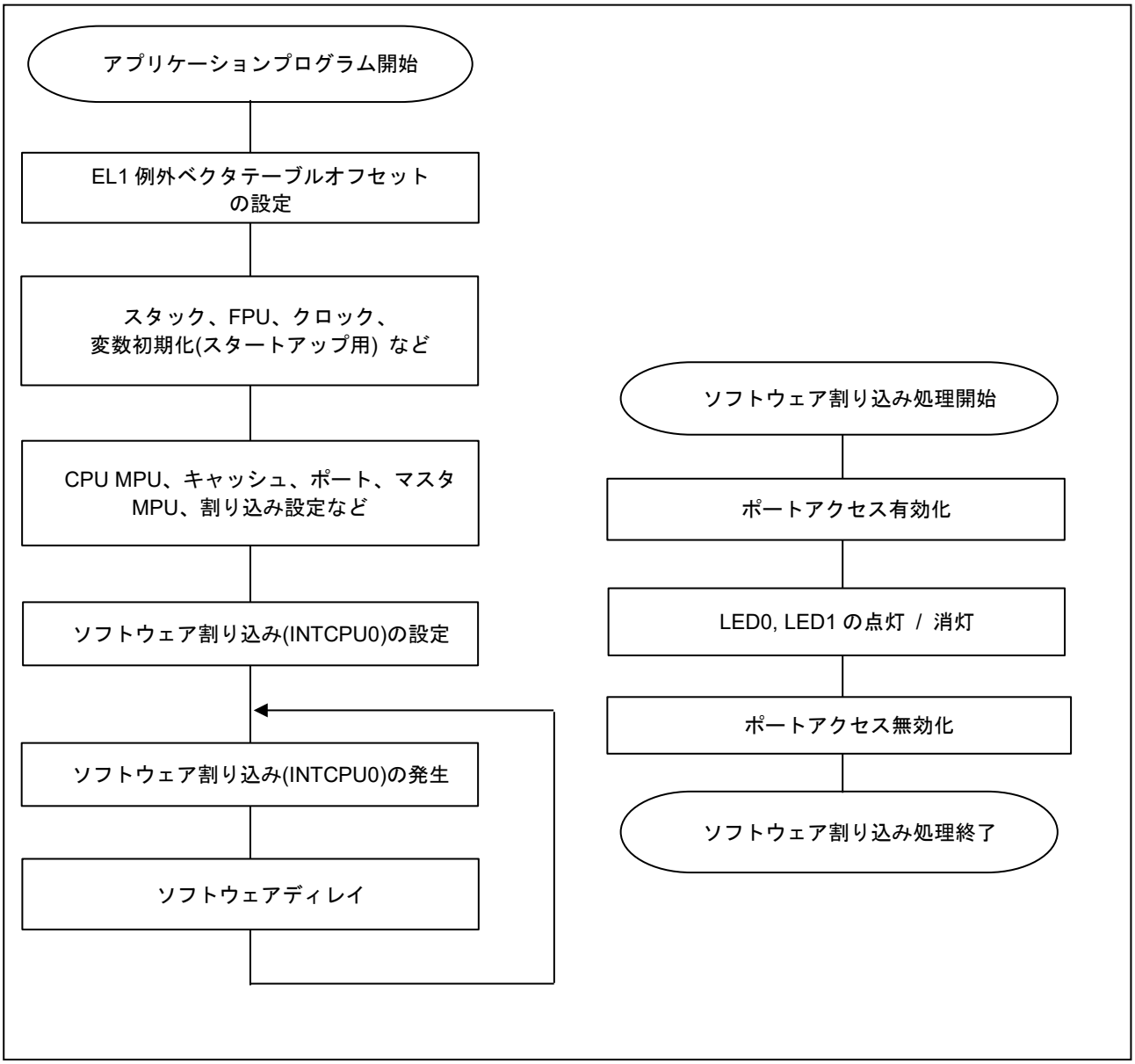

図 3-3 アプリケーションプログラムの動作概要

3.2 ローダテーブル

ローダテーブルはローダプログラムがアプリケーションプログラムをコピーする際に参照するテーブルです。ローダテーブルにはプログラムのコピーに必要なパラメータが定義されており、ローダプログラムはテーブルのパラメータに従ってコピー処理を行います。ローダテーブルは必要に応じて複数準備することが可能で、各テーブルにパラメータを保持できます。

ローダテーブルにはコピー元アドレス、コピー先アドレス、コピーサイズ、テーブル有効/無効フラグの 4 つのパラメータがあります。ローダテーブルのパラメータの詳細を表 3.1 に示します。

本サンプルプログラムではローダプログラムの loader_table.c に 4 つのローダテーブルが準備されています。使用する動作モードに応じてコピー元のアドレスが変わります。本サンプルプログラムにおけるローダテーブルのパラメータを表 3.2、表 3.3 に示します。

表 3.1 ローダテーブルのパラメータ

引数	パラメータ	説明
1	Src	コピーするプログラムのコピー元アドレス。
2	Dst	コピーするプログラムのコピー先アドレス。
3	Size	コピーするプログラムのサイズ。
4	Enable flag	テーブルの有効/無効を決定するフラグ。 本フラグが無効の場合、他のパラメータを設定してもコピー処理が行われません。 0 : Disable 1 : Enable

表 3.2 本サンプルプログラムにおけるローダテーブルのパラメータ(xSPI0 ブートモード版)

テーブル	Src	Dst	Size	Enable flag
0	0x6010_0000	0x1008_0000	0x0000_231C	0x1
1 注1 注2	0xFFFF_FFFF or 0x6020_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_0F70	0x0 or 0x1
2 注1	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0
3 注1	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0

【注】 1. 本サンプルプログラムにおいて、テーブル 1,2,3 は無効です。ユーザが自由に変更可能です。
2. CPU1 プログラムを有効にした場合、テーブル 1 は CPU1 プログラム用のパラメータを保持します。詳細については「5.3 CPU1 プログラムのデバッグ方法」を参照してください。

表 3.3 本サンプルプログラムにおけるローダテーブルのパラメータ(16 ビットバスブートモード版)

テーブル	Src	Dst	Size	Enable flag
0	0x7010_0000	0x1008_0000	0x0000_2304	0x1
1 注1 注2	0xFFFF_FFFF or 0x7018_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_0F70	0x0 or 0x1
2 注1	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0
3 注1	0xFFFF_FFFF	0xFFFF_FFFF	0xFFFF_FFFF	0x0

【注】 1. 本サンプルプログラムにおいて、テーブル 1,2,3 は無効です。ユーザが自由に変更可能です。
2. CPU1 プログラムを有効にした場合、テーブル 1 は CPU1 プログラム用のパラメータを保持します。詳細については「5.3 CPU1 プログラムのデバッグ方法」を参照してください。

3.3 メモリマップ

3.3.1 フラッシュメモリのプログラム配置

表 3.4、表 3.5 に本サンプルプログラムのフラッシュメモリに配置されるプログラムを示します。使用する動作モードに応じてフラッシュメモリのアドレスが変わります。デバッグ接続時、プログラムはフラッシュメモリへダウンロードされます。各プログラムはブート処理やローダプログラムによってロード先アドレスに展開され、RAM 上で実行されます。

表 3.4 フラッシュメモリのプログラム配置とロード先アドレス(xSPI0 ブートモード版)

フラッシュメモリアドレス	内容物	ロード先アドレス
0x6000_0000	ローダ用パラメータ	-
0x6000_004C	ローダプログラム	0x0010_2000 (BTCM)
0x6008_0000	ローダテーブル	-
0x6010_0000	アプリケーションプログラム	0x1008_0000 (System SRAM)
0x6020_0000 注	CPU1 プログラム	0x1000_0000 (System SRAM)

【注】 CPU1 プログラムはデフォルトで無効です。有効にしたい場合は「5.3 CPU1 プログラムのデバッグ方法」を参照してください。

表 3.5 フラッシュメモリのプログラム配置とロード先アドレス(16 ビットバスブートモード版)

フラッシュメモリアドレス	内容物	ロード先アドレス
0x7000_0000	ローダ用パラメータ	-
0x7000_004C	ローダプログラム	0x0010_2000 (BTCM)
0x7008_0000	ローダテーブル	-
0x7010_0000	アプリケーションプログラム	0x1008_0000 (System SRAM)
0x7018_0000 注	CPU1 プログラム	0x1000_0000 (System SRAM)

【注】 CPU1 プログラムはデフォルトで無効です。有効にしたい場合は「5.3 CPU1 プログラムのデバッグ方法」を参照してください。

3.3.2 サンプルプログラムにおける各セクション配置

3.3.2.1 EWARM

表 3.6 にローダプログラムで使用するセクション、表 3.7 にアプリケーションプログラムで使用するセクションを示します。これらのセクションはリンカスクリプトで定義されています。

表 3.6 ローダプログラムで使用するセクション (EWARM)

セクション/ブロックの名前	内容	格納/実行領域[注1]
LOADER_PARAM_BLOCK	ローダパラメータ	Flash
PRG_RBLOCK	コード領域(格納用)	Flash
USER_DATA_RBLOCK	変数領域(格納用)	Flash
PRG_WBLOCK	コード領域(実行用)	BTCM
USER_DATA_WBLOCK	初期値あり変数領域(実行用)	BTCM
USER_DATA_ZBLOCK	初期値なし変数領域(実行用)	BTCM
APPLICATION_PRG_RBLOCK	アプリケーションプログラム領域(格納用)	Flash
APPLICATION_PRG_WBLOCK	アプリケーションプログラム領域(実行用)	System SRAM
CPU1_PRG_RBLOCK [注2]	CPU1 プログラム領域(格納用)	Flash
CPU1_PRG_WBLOCK [注2]	CPU1 プログラム領域(実行用)	System SRAM

【注】 1. xSPI0 バスブート版ではシリアルフラッシュメモリ、16 ビットバスブート版では NOR フラッシュメモリが格納領域となります。
2. CPU1 プログラムはデフォルトで無効です。有効にしたい場合は「5.3 CPU1 プログラムのデバッグ方法」を参照してください。

表 3.7 アプリケーションプログラムで使用するセクション (EWARM)

セクション/ブロックの名前	内容	格納/実行領域[注1]
PRG_RBLOCK	コード領域(格納用)	Flash
USER_DATA_RBLOCK	変数領域(格納用)	Flash
PRG_WBLOCK	コード領域(実行用)	System SRAM
USER_DATA_WBLOCK	初期値あり変数領域(実行用)	System SRAM
USER_DATA_ZBLOCK	初期値なし変数領域(実行用)	System SRAM

【注】 1. xSPI0 バスブート版ではシリアルフラッシュメモリ、16 ビットバスブート版では NOR フラッシュメモリが格納領域となります。

3.3.2.2 e² studio

表 3.8 にローダプログラムで使用するセクション、表 3.9 にアプリケーションプログラムで使用するセクションを示します。これらのセクションはリンカスクリプトで定義されています。

表 3.8 ローダプログラムで使用するセクション (e² studio)

セクション/ブロックの名前	内容	格納/実行領域[注1]
.loader_param	ローダパラメータ	Flash
.flash_contents	コード領域(格納用)	Flash
.flash_contents	変数領域(格納用)	Flash
.text .intvec .reset_handler .loader_text	コード領域(実行用)	BTCM
.data .rodata	初期値あり変数領域(実行用)	BTCM
.bss	初期値なし変数領域(実行用)	BTCM
.IMAGE_APP_FLASH_section	アプリケーションプログラム領域(格納用)	Flash
.IMAGE_APP_RAM	アプリケーションプログラム領域(実行用)	System SRAM
.IMAGE_CPU1_FLASH_section[注2]	CPU1 プログラム領域(格納用)	Flash
.IMAGE_CPU1_RAM[注2]	CPU1 プログラム領域(実行用)	System SRAM

【注】 1. xSPI0 バスブート版ではシリアルフラッシュメモリ、16 ビットバスブート版では NOR フラッシュメモリが格納領域となります。
2. CPU1 プログラムはデフォルトで無効です。有効にしたい場合は「5.3 CPU1 プログラムのデバッグ方法」を参照してください。

表 3.9 アプリケーションプログラムで使用するセクション (e² studio)

セクション/ブロックの名前	内容	格納/実行領域[注1]
.flash_contents	コード領域(格納用)	Flash
.flash_contents	変数領域(格納用)	Flash
.text .intvec .reset_handler .loader_text	コード領域(実行用)	System SRAM
.data .rodata	初期値あり変数領域(実行用)	System SRAM
.bss	初期値なし変数領域(実行用)	System SRAM

【注】 1. xSPI0 バスブート版ではシリアルフラッシュメモリ、16 ビットバスブート版では NOR フラッシュメモリが格納領域となります。

3.3.3 CPU MPU の設定

本サンプルプログラムにおいて CPU がアクセスを行う領域の CPU MPU の設定を表 3.10 に示します。アプリケーションプログラムのスタートアップ処理時に本設定が適用されます。

表 3.10 CPU MPU の設定

内容	アドレス	メモリタイプ
システム SRAM	0x1000_0000 ~ 0x101F_FFFF	領域 2 ノーマル、キャッシュ可、非共有
システム SRAM(ミラー領域)	0x3000_0000 ~ 0x301F_FFFF	領域 3 ノーマル、キャッシュ不可、共有
外部アドレス空間(ミラー領域) xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x4000_0000 ~ 0x5FFF_FFFF	領域 4 ノーマル、キャッシュ不可、共有
外部アドレス空間 xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x6000_0000 ~ 0x7FFF_FFFF	領域 5 ノーマル、キャッシュ可、非共有
ノンセーフティ周辺モジュール	0x8000_0000 ~ 0x80FF_FFFF	領域 6 デバイス(nGnRE)、命令フェッチ不可
セーフティ周辺モジュール	0x8100_0000 ~ 0x81FF_FFFF	領域 7 デバイス(nGnRE)、命令フェッチ不可

3.3.4 例外処理ベクタテーブル

RZ/T2ME の Exception Level1 には 7 種類の例外処理(リセット、未定義命令、SVC、プリフェッチアボート、データアボート、IRQ、FIQ)があり、指定されたオフセット番地から 32 バイトの領域に配置されます。例外処理ベクタテーブルには、各例外処理への分岐命令を記述します。

表 3.11 に本サンプルプログラムにおける例外処理ベクタテーブルの内容を示します。必要に応じて自由に変更してください。

表 3.11 例外処理ベクタテーブル

例外	ハンドラアドレス[注1]	備考[注2]
RESET	オフセット	スタートアッププログラムに分岐
未定義命令	オフセット + 0x0000 0004	Default_Handler へ分岐
SVC	オフセット + 0x0000 0008	Default_Handler へ分岐
プリフェッチアボート	オフセット + 0x0000 000C	Default_Handler へ分岐
データアボート	オフセット + 0x0000 0010	Default_Handler へ分岐
Reserved	オフセット + 0x0000 0014	Default_Handler へ分岐
IRQ	オフセット + 0x0000 0018	IRQ_Handler へ分岐(割り込みで使用)
FIQ	オフセット + 0x0000 001C	Default_Handler へ分岐

【注】 1. 本サンプルプログラムのオフセットは下記のように設定されています。
ローダプログラム : 0x0010_2000
アプリケーションプログラム : 0x1008_0000
CPU1 プログラム : 0x1000_0000
2. Default_Handler ではソフトウェアブレーク命令が実行されます。

3.4 関数仕様

サンプルプログラムの関数仕様を示します。

3.4.1 system_init

system_init	
概要	システム初期設定部 1
宣言	void system_init (void)
説明	Exception Level 2 から Exception Level 1 への移行、例外処理ベクタテーブルオフセットの設定などを行い、stack_init へ分岐します。
引数	なし
リターン値	なし
補足	ブート処理終了後、スタートアップ処理で最初に実行される関数です。

3.4.2 stack_init

stack_init	
概要	システム初期設定部 2
宣言	void stack_init (void)
説明	スタックの設定、FPU の設定、クロック設定、スタートアップ処理に使用される変数の初期化、CPU MPU の設定、キャッシュの設定、ポートの設定などを行い、main 処理へ分岐します。
引数	なし
リターン値	なし
補足	なし

3.4.3 hal_entry

hal_entry	
概要	メイン処理
宣言	void hal_entry (void)
説明	• ローダプログラム：アプリケーションプログラムをローダテーブルの情報に従って RAM へ展開します。プログラムのコピー開始前に LED0 が点灯し、コピー完了後に LED3 が点灯します。 • アプリケーションプログラム：ソフトウェア割り込み(INTCPU0)で LED0 と LED1 が点滅します。
引数	なし
リターン値	なし
補足	なし

3.4.4 bsp_copy_multibyte

bsp_copy_multibyte	
概要	コピー処理
宣言	void bsp_copy_multibyte (uintptr_t *src, uintptr_t *dst, uintptr_t bytesize)
説明	引数で指定したサイズ分データのコピーを行います。
引数	• uintptr_t *src : コピー元アドレス • uintptr_t *dst : コピー先アドレス • uintptr_t bytesize :コピーするデータサイズ
リターン値	なし
補足	なし

3.5 フローチャート

3.5.1 ローダプログラム

3.5.1.1 system_init 処理

図 3-4 にローダプログラムの system_init 処理のフローチャートを示します。

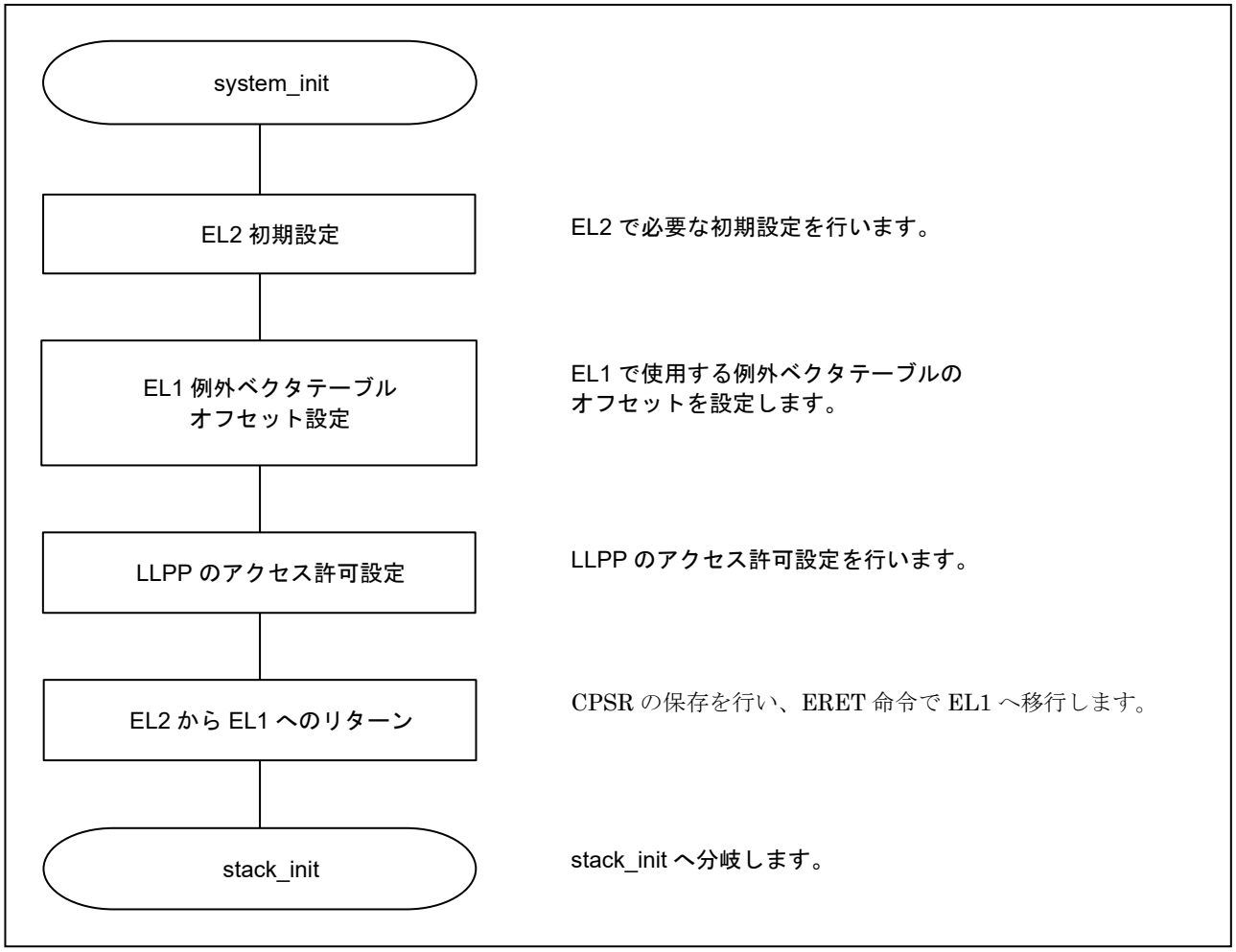

図 3-4 system_init 処理(ローダプログラム)

3.5.1.2 stack_init 処理

図 3-5 にローダプログラムの stack_init 処理のフローチャートを示します。

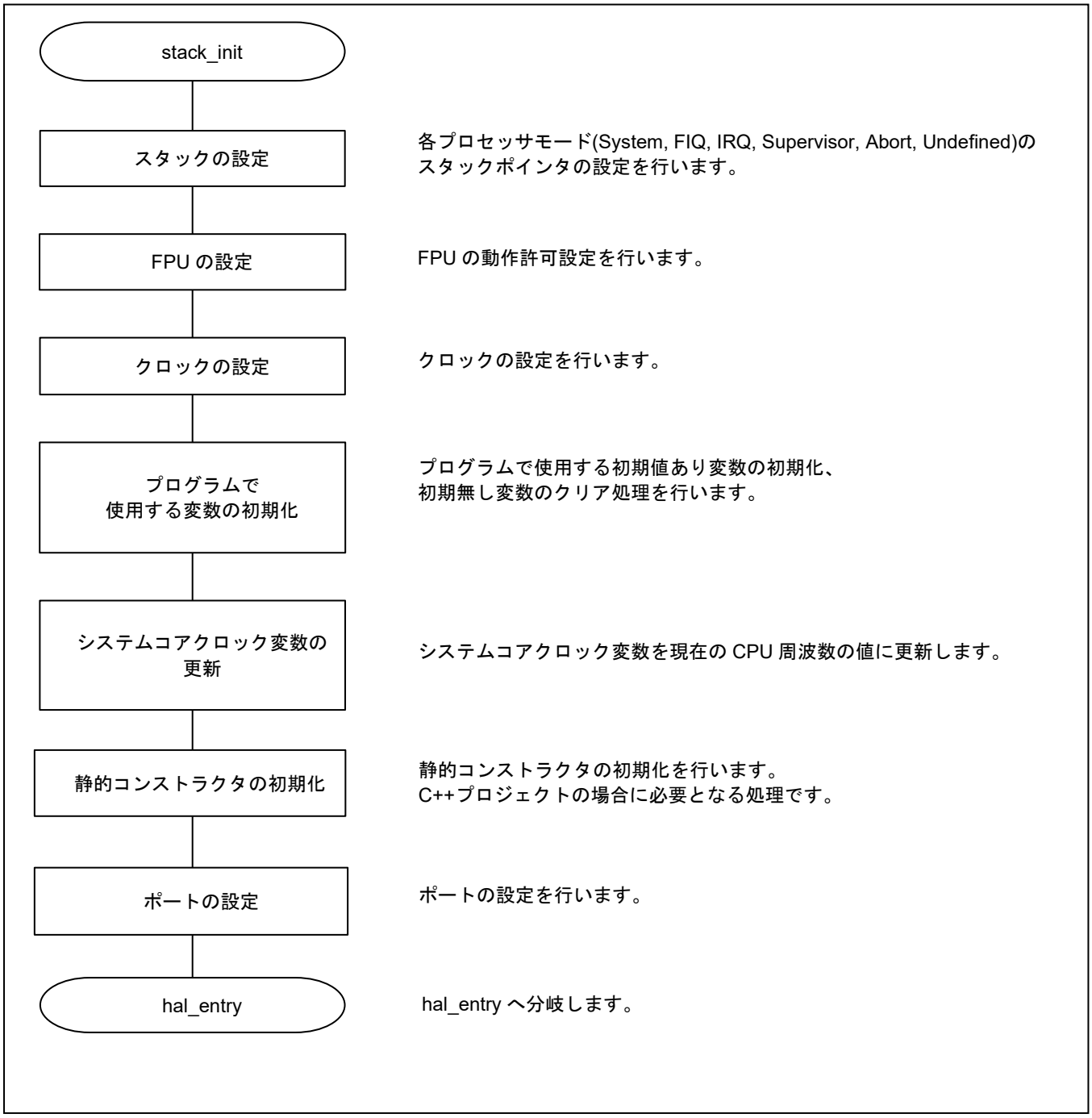

図 3-5 stack_init 処理(ローダプログラム)

3.5.1.3 hal_entry 処理

図 3-6 にローダプログラムの hal_entry 処理のフローチャートを示します。

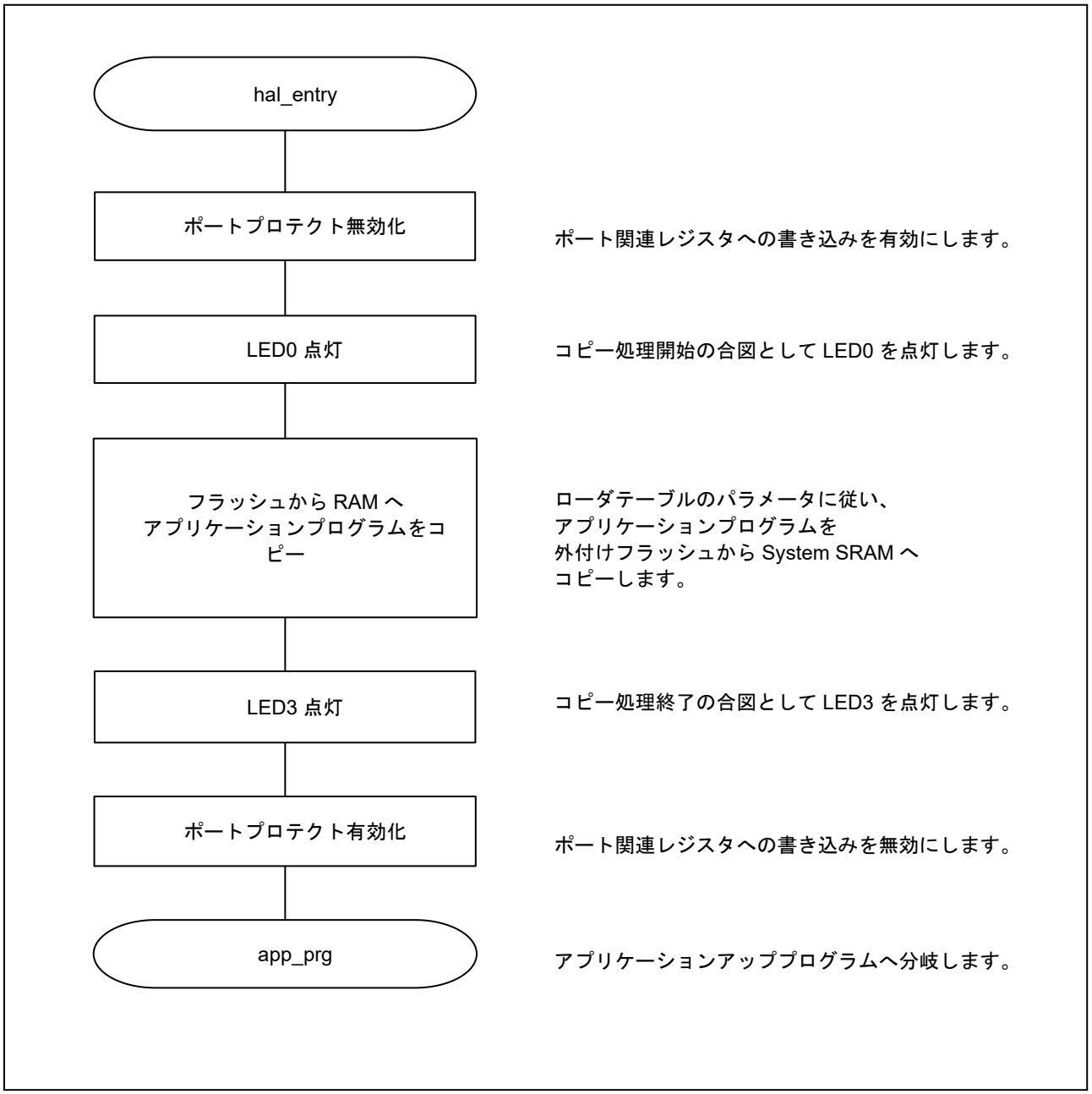

図 3-6 hal_entry 処理(ローダプログラム)

3.5.2 アプリケーションプログラム

3.5.2.1 system_init 処理

図 3-7 にアプリケーションプログラムの system_init 処理のフローチャートを示します。

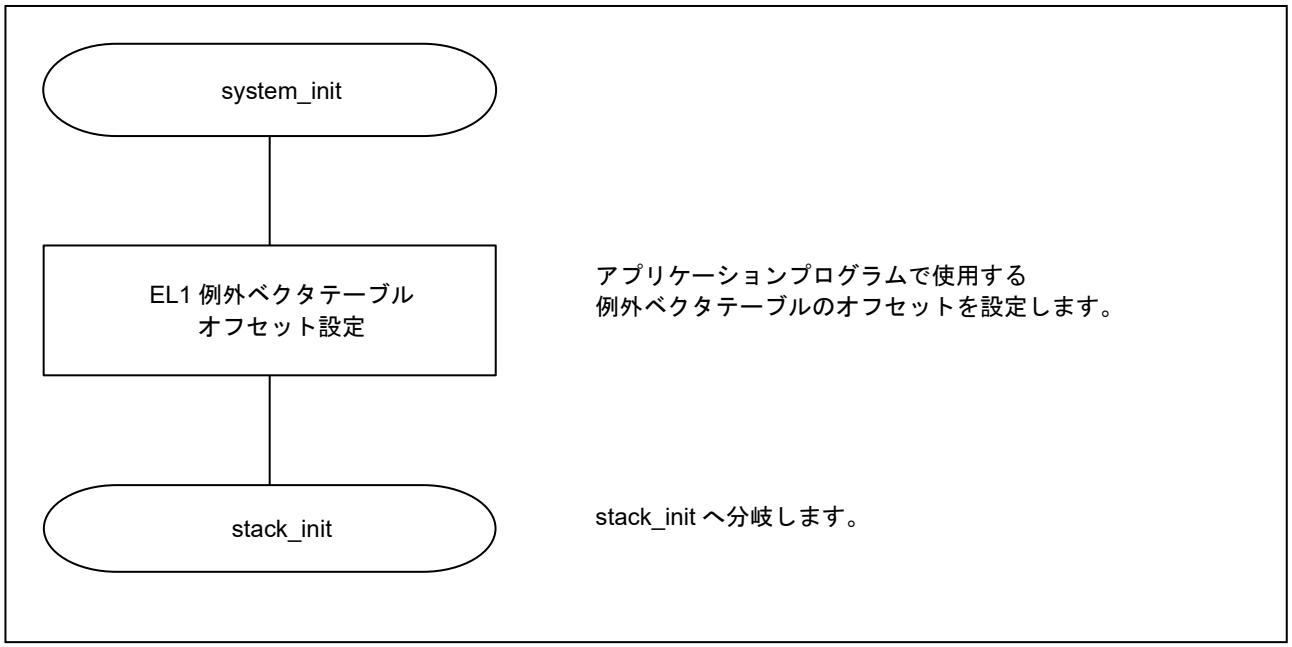

図 3-7 system_init 処理(アプリケーションプログラム)

3.5.2.2 stack_init 処理

図 3-8、図 3-9 にアプリケーションプログラムの stack_init 処理のフローチャートを示します。

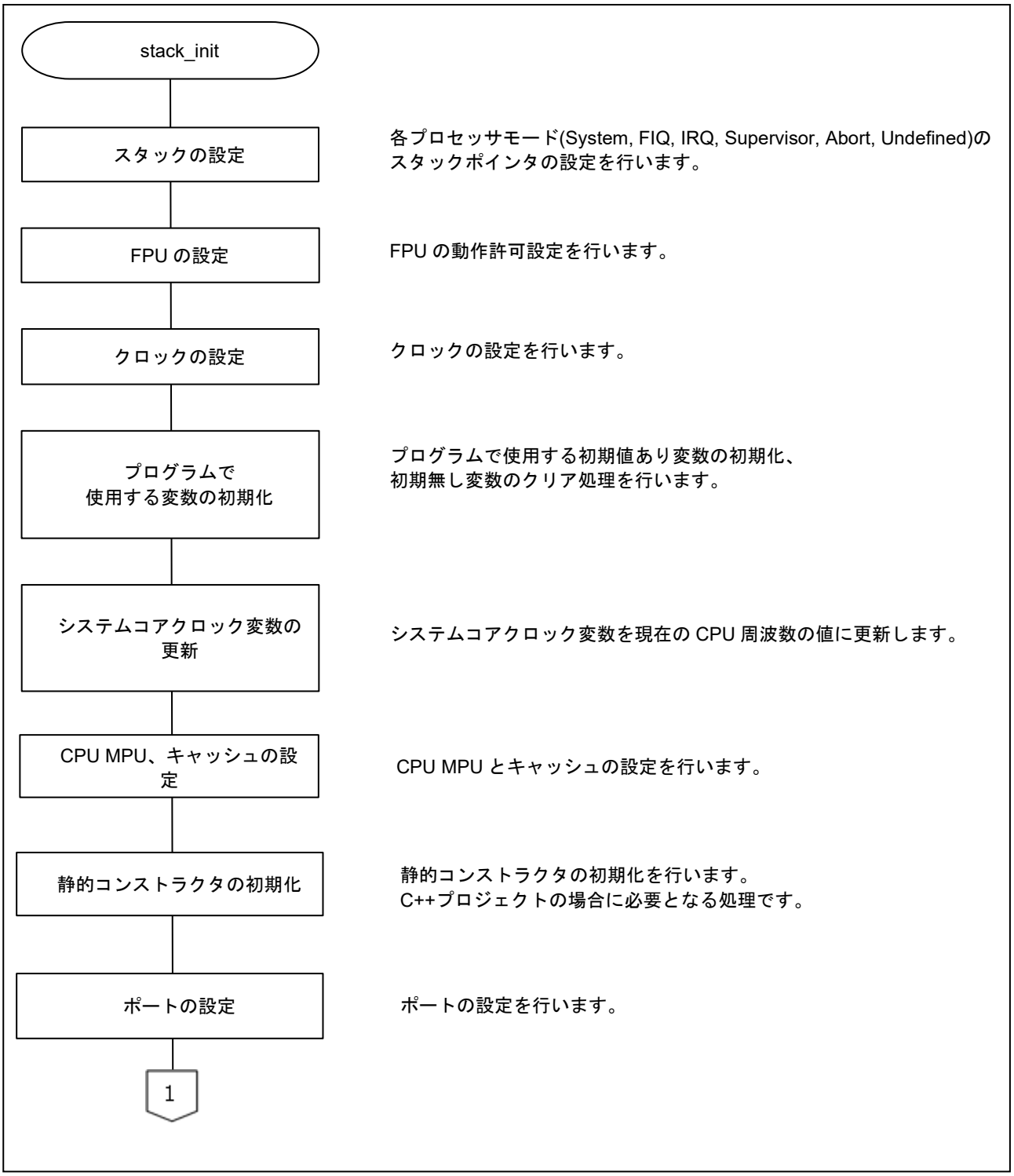

図 3-8 stack_init 処理 (1/2)(アプリケーションプログラム)

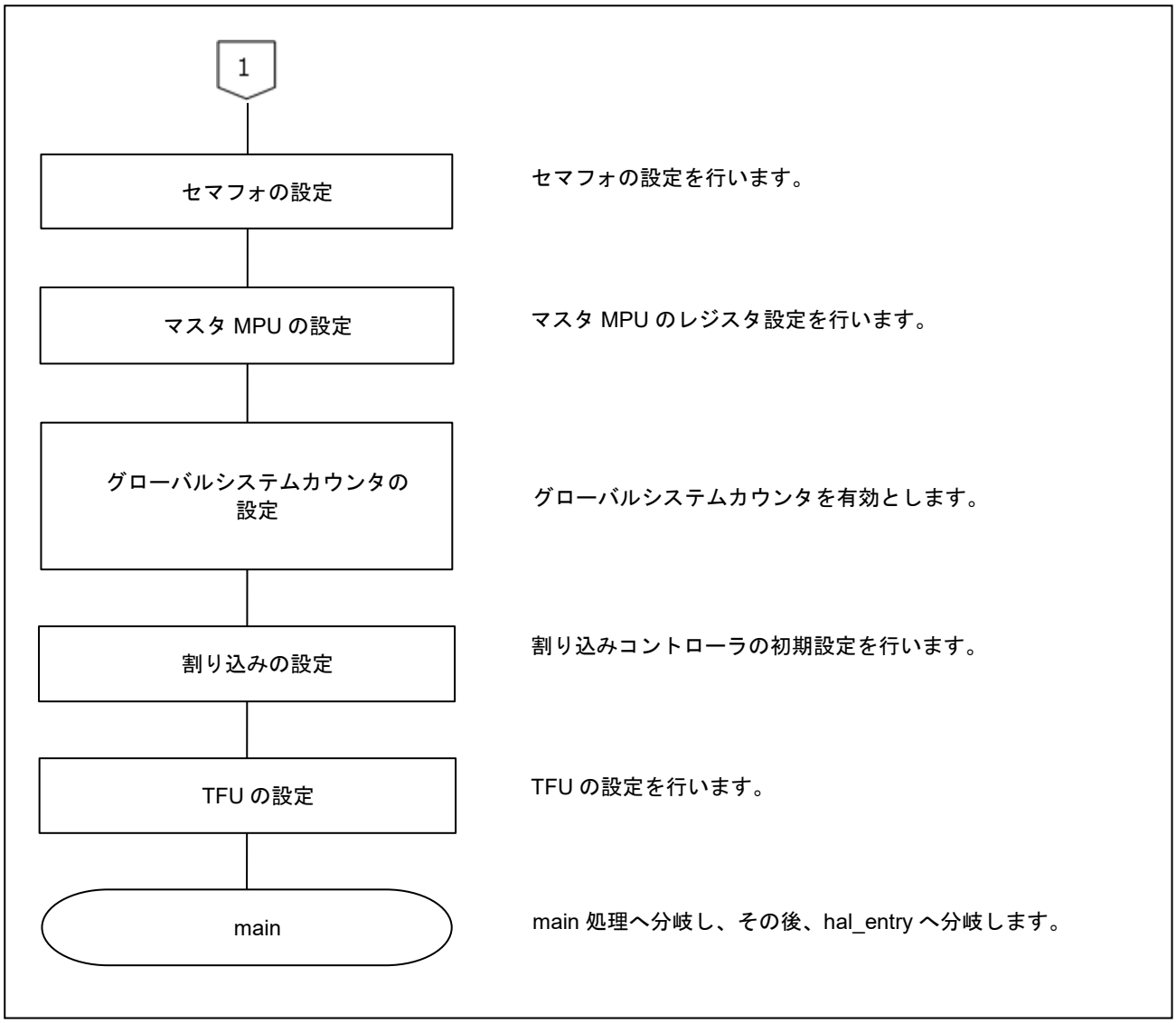

図 3-9 stack_init 処理(2/2)(アプリケーションプログラム)

3.5.2.3 hal_entry 処理

図 3-10 にアプリケーションプログラムの hal_entry 処理のフローチャートを示します。

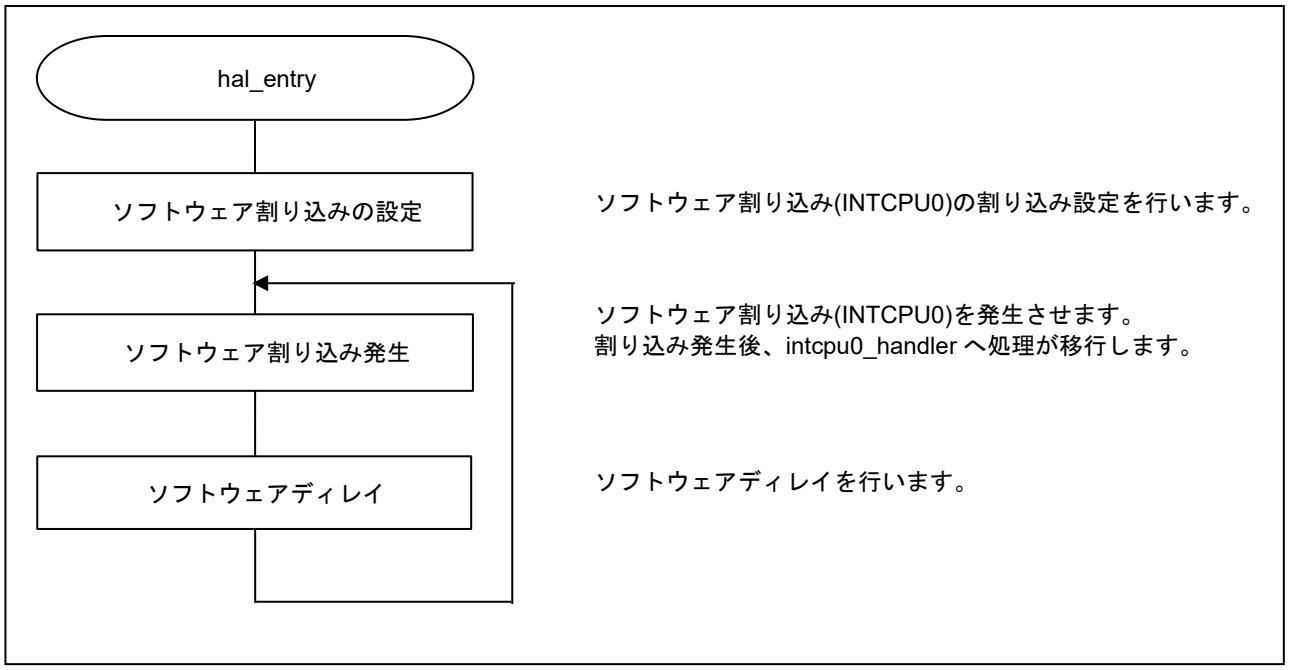

図 3-10 hal_entry 処理(アプリケーションプログラム)

図 3-11 にアプリケーションプログラムの割り込み処理のフローチャートを示します。

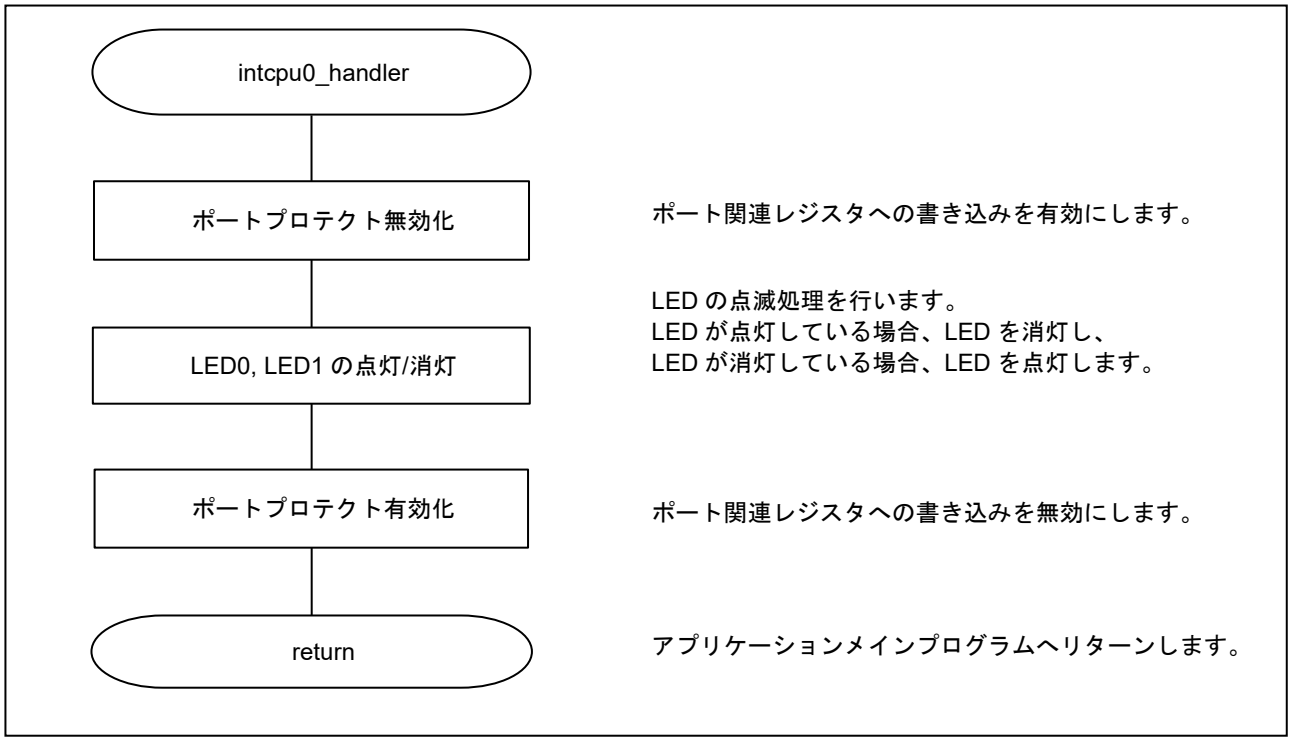

図 3-11 割り込み処理(アプリケーションプログラム)

4. 参考ドキュメント

- ユーザーズマニュアル : ハードウェア
  RZ/T2ME グループ ユーザーズマニュアル ハードウェア編
  (最新版をルネサスエレクトロニクスホームページから入手してください。)

  Renesas Starter Kit+ for RZ/T2M, RZ/T2ME ユーザーズマニュアル
  (最新版をルネサスエレクトロニクスホームページから入手してください。)

- テクニカルアップデート / テクニカルニュース
  (最新版をルネサスエレクトロニクスホームページから入手してください。)

- ユーザーズマニュアル : 開発環境
  IAR 統合開発環境(IAR Embedded Workbench® for Arm)に関しては、最新版を IAR システムズ社ホームページから入手してください。
  ルネサスエレクトロニクス統合開発環境(e² studio)に関しては、最新版をルネサスエレクトロニクスホームページから入手してください。

5. 付録. 各開発環境における補足内容

ここでは、本サンプルプログラムを本開発環境で使用する際のデバッグまでの手順を説明します。xSPI0 ブートモード版を使用した例を示します。

5.1 サンプルプログラムのデバッグ手順

5.1.1 EWARM : IAR システムズ社製

1. EWARM を起動し、[ファイル] -> [開く] -> [ワークスペース]で Loader_application_projects¥RZT2ME_bsp_xpi0bootx1_app_loader.eww を指定し、サンプルプロジェクトを開きます。このワークスペースにはローダプロジェクトとアプリケーションプロジェクトが含まれています。
2. RZ/T2ME_bsp_xspi0bootx1_app のプロジェクトをワークスペースから選択し、[プロジェクト]→[全てを再ビルド]を実行します。
3. RZ/T2ME_bsp_xspi0bootx1_loader のプロジェクトをワークスペースから選択し、[プロジェクト]→[全てを再ビルド]を実行します。
4. RZ/T2ME_bsp_xspi0bootx1_loader のプロジェクトをワークスペースから選択し、RZ/T2ME 評価ボードと I-jet を接続した状態で、[プロジェクト]->[ダウンロードしてデバッグ]を選択します。
5. エミュレータ接続後、専用フラッシュダウンローダにより外付けシリアルフラッシュメモリへプログラムの書き込みが行われた後に、デバッグが開始されます。この時、ローダプログラムとアプリケーションプログラムが同時に外付けシリアルフラッシュメモリへ書き込まれます。

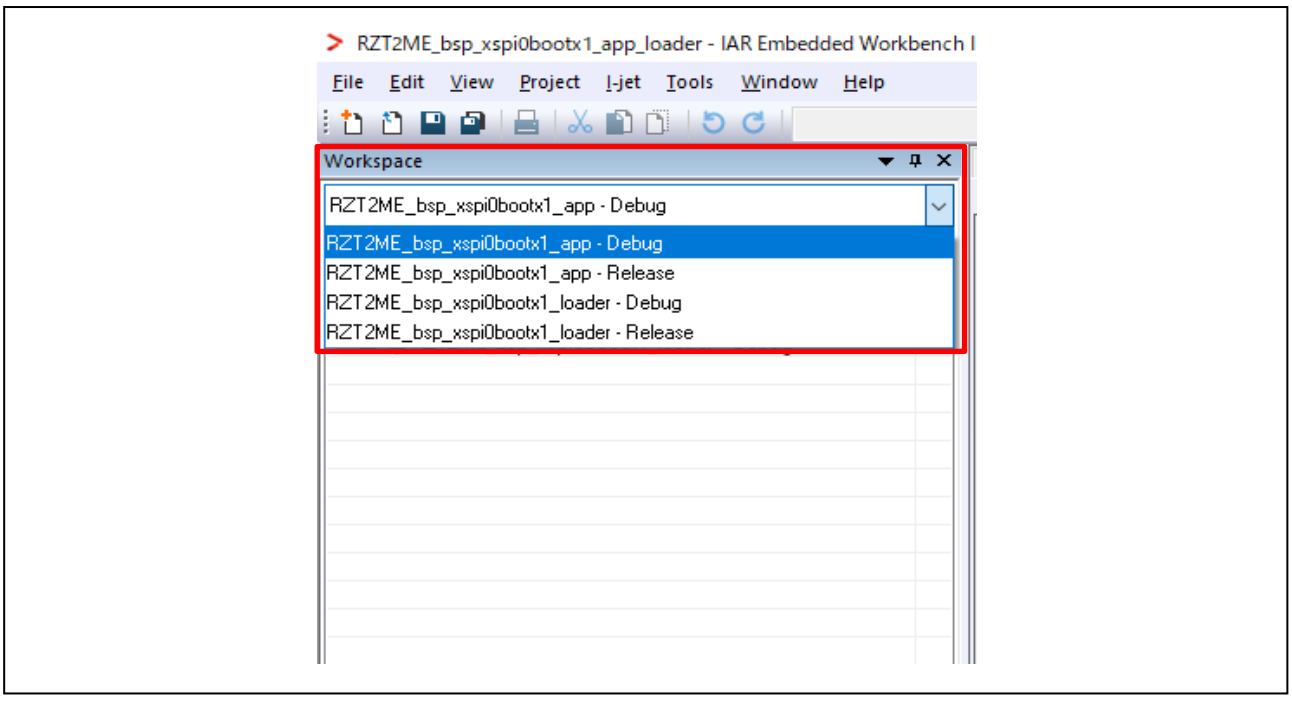

図 5-1 プロジェクト選択

EWARM 環境における補足事項

デバッグで動作を確認する際、ローダプログラムのプロジェクトを選択してデバッグを行ってください。下記のオプション設定により、ローダプロジェクトでデバッグ接続時にローダプログラムとアプリケーションプログラムが同時に外付けシリアルフラッシュメモリへ書き込まれます。

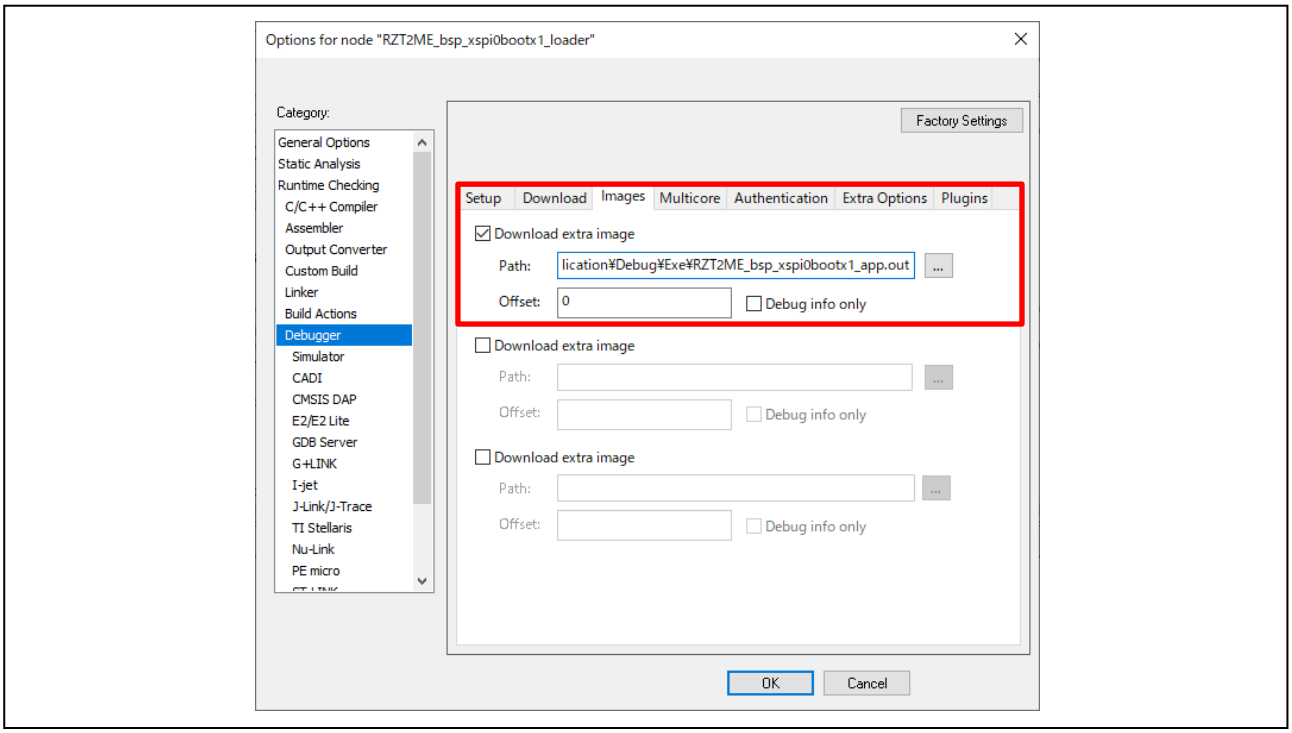

図 5-2 EWARM オプション設定

5.1.2 e² studio : Renesas 社製

1. e² studio を起動しワークスペースへ移動後、[ファイル] -> [インポート]をクリックし、一般 -> 既存プロジェクトをワークスペースへ を選択して[次へ]をクリックします。
2. プロジェクトのインポート画面でアーカイブ・ファイルの選択にて Loader_application_projects.zip を指定して[終了]をクリックします。
3. RZ/T2ME_bsp_xspi0bootx1_app, RZ/T2ME_bsp_xspi0bootx1_loader の順でビルドを実行します。
4. “RZ/T2ME_bsp_xspi0bootx1_loader”プロジェクトのデバッグ接続設定を選択し、RZ/T2ME 評価ボードと J-Link を接続した状態で、[デバッグ]を選択しデバッグを開始します。
5. エミュレータ接続後、専用フラッシュダウンローダにより外付けシリアルフラッシュメモリへプログラムの書き込みが行われた後に、デバッグが開始されます。この時、ローダプログラムとアプリケーションプログラムが同時に外付けシリアルフラッシュメモリへ書き込まれます。

e² studio 環境における補足事項

デバッグで動作を確認する際、ローダプログラムのプロジェクトを選択してデバッグを行ってください。下記のデバッグ接続設定により、ローダプロジェクトでデバッグ接続時にローダプログラムとアプリケーションプログラムが同時に外付けシリアルフラッシュメモリへ書き込まれます。

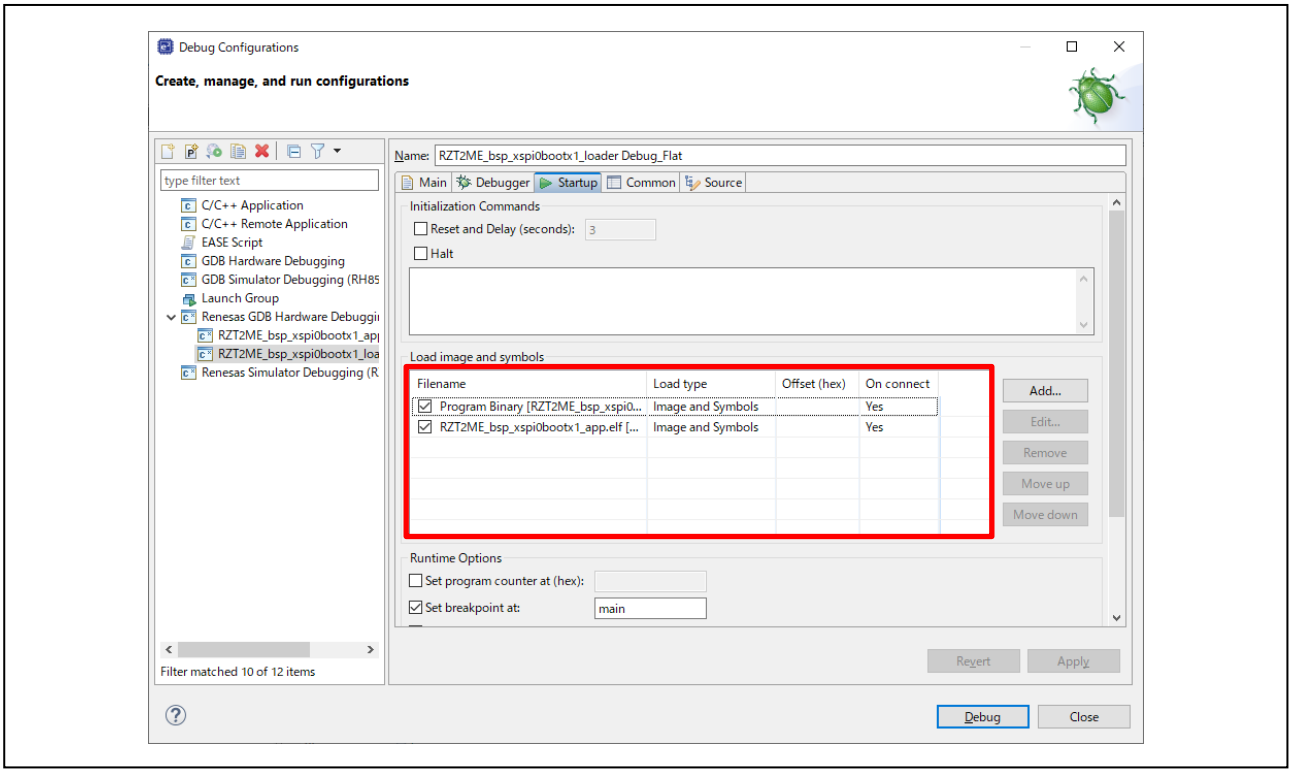

図 5-3 e2 studio デバッグ接続設定

5.2 アプリケーションプログラムの RAM 配置の変更例

本サンプルプログラムはローダテーブルで指定されたコピー元アドレスからコピー先アドレスへアプリケーションプログラムのコピーを行います。ユーザは必要に応じてコピー元アドレスとコピー先アドレスを書き換えることでアプリケーションプログラムの配置を変更することが可能です。

5.2.1 EWARM : IAR システムズ社製

下記にアプリケーションプログラムの配置を System SRAM から ATCM に変更する例(xspi0boot のプロジェクト)を示します。

fsp_xspi0_boot_loader.icf (ローダプログラムのリンカスクリプト)

```
  Default
  ~~~
  /* Internal memory */
  define region BTCM_LDR_region  =   mem:[from 0x00102000 size  56K];
  define region APPLICATION_RAM_region  =  mem:[from 0x10080000 size 64K];

  /* Flash memory */
  define region LOADER_TABLE_region  =  mem:[from 0x60080000 size 64K];
  define region APPLICATION_ROM_region  =  mem:[from 0x60100000 size 64K];
  ~~~

  ↓

  After changing
  ~~~
  /* Internal memory */
  define region BTCM_LDR_region  =   mem:[from 0x00102000 size  56K];
  define region APPLICATION_RAM_region  =  mem:[from 0x00000000 size 64K]; /* Change copy destination
address to ATCM */

  /* Flash memory */
  define region LOADER_TABLE_region  =  mem:[from 0x60080000 size 64K];
  define region APPLICATION_ROM_region  =  mem:[from 0x60100000 size 64K];
  ~~~
```

fsp_xspi0_boot_app.icf (アプリケーションプログラムのリンカスクリプト)

```
  Default
  ~~~
  place at start of SYSTEM_RAM_PRG_region  { block PRG_WBLOCK };
  place in SYSTEM_RAM_PRG_region          { block USER_DATA_WBLOCK };
  place in SYSTEM_RAM_PRG_region          { block USER_DATA_ZBLOCK };
  place in SYSTEM_RAM_PRG_region          { rw data,
                                            rw section .sys_stack,
                                            rw section .svc_stack,
                                            rw section .irq_stack,
                                            rw section .fiq_stack,
                                            rw section .und_stack,
                                            rw section .abt_stack };
  place in SYSTEM_RAM_PRG_region          { rw section HEAP };
  ~~~

  ↓

  After changing
  ~~~
  place at start of ATCM_region  { block PRG_WBLOCK };          /* Change code area to ATCM */
  place in ATCM_region           { block USER_DATA_WBLOCK };    /* Change data area to ATCM */
  place in ATCM_region           { block USER_DATA_ZBLOCK };    /* Change bss area to ATCM */
  place in ATCM_region           { rw data,                     /* Change stack area to ATCM */
                                   rw section .sys_stack,
                                   rw section .svc_stack,
                                   rw section .irq_stack,
                                   rw section .fiq_stack,
                                   rw section .und_stack,
                                   rw section .abt_stack };
```

```
place in ATCM_region            { rw section HEAP };    /* Change HEAP area to ATCM */
~~~
```

5.2.2 e2studio : Renesas 社製

下記にアプリケーションプログラムの配置を System SRAM から ATCM に変更する例(xspi0boot のプロジェクト)を示します。

fsp_xspi0_boot_loader.ld (ローダプログラムのリンカスクリプト)

```
Default
~~~
.IMAGE_APP_RAM 0x10080000 : AT (0x10080000)
{
      IMAGE_APP_RAM_start = .;
      KEEP(*(APP_IMAGE_RAM))
}
.IMAGE_APP_FLASH_section 0x60100000 : AT (0x60100000)
{
      IMAGE_APP_FLASH_section_start = .;
      KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
      IMAGE_APP_FLASH_section_end = .;
}
~~~

↓

After changing
~~~
.IMAGE_APP_RAM 0x00000000 : AT (0x00000000) /* Change copy destination address to ATCM */
{
      IMAGE_APP_RAM_start = .;
      KEEP(*(APP_IMAGE_RAM))
}
.IMAGE_APP_FLASH_section 0x60100000 : AT (0x60100000)
{
      IMAGE_APP_FLASH_section_start = .;
      KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
      IMAGE_APP_FLASH_section_end = .;
}
~~~
```

fsp_xspi0_boot_app.ld (アプリケーションプログラムのリンカスクリプト)

```
Default
~~~
.text 0x10080000 : AT (TEXT_IMAGE)
{
   ~~~
} > SYSTEM_RAM
.rvectors :
{
   ~~~
} > SYSTEM_RAM
.ARM.extab :
{
   ~~~
} > SYSTEM_RAM
.ARM.exidx :
{
   ~~~
} > SYSTEM_RAM
.data _text_end :  AT (DATA_IMAGE)
{
   ~~~
} > SYSTEM_RAM
.got :
{
   ~~~
} > SYSTEM_RAM
```

```
.bss :
{
    ~~~
} > SYSTEM_RAM
.heap (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.aarch64_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.thread_stack (NOLOAD):
{
    ~~~
} > SYSTEM_RAM
.sys_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.svc_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.irq_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.fiq_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.und_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM
.abt_stack (NOLOAD) :
{
    ~~~
} > SYSTEM_RAM

~~~

↓

After Changing
~~~
.text 0x00000000  : AT (TEXT_IMAGE) /* Change code area to ATCM */
{
    ~~~
} > ATCM
.rvectors :
{
    ~~~
} > ATCM
.ARM.extab :
{
    ~~~
} > ATCM
.ARM.exidx :
{
    ~~~
} > ATCM
.data _text_end :  AT (DATA_IMAGE) /* Change data area to ATCM */
{
    ~~~
} > ATCM
.got :
{
    ~~~
} > ATCM
.bss : /* Change bss area to ATCM */
{
    ~~~
```

```
} > ATCM
.heap (NOLOAD) : /* Change heap area to ATCM */
{
    ~~~
} > ATCM
.aarch64_stack (NOLOAD) :
{
    ~~~
} > ATCM
.thread_stack (NOLOAD):
{
    ~~~
} > ATCM
.sys_stack (NOLOAD) : /* Change stack area to ATCM */
{
    ~~~
} > ATCM
.svc_stack (NOLOAD) :
{
    ~~~
} > ATCM
.irq_stack (NOLOAD) :
{
    ~~~
} > ATCM
.fiq_stack (NOLOAD) :
{
    ~~~
} > ATCM
.und_stack (NOLOAD) :
{
    ~~~
} > ATCM
.abt_stack (NOLOAD) :
{
    ~~~
} > ATCM
~~~
```

5.3 CPU1 プログラムのデバッグ方法

本サンプルプログラムはデフォルトで CPU0 のシングルコア動作となります。プロジェクトのオプションに「USE_CPU1」という定義を追加しており、その値を変更することで CPU1 を動作させるために必要なプログラムが有効化されます。

CPU1 を有効化すると、ローダテーブルにあるテーブル 1 のパラメータが CPU1 プログラムのコピーに必要な情報に置き換わります。これにより、ローダプログラムはテーブル 1 のパラメータを参照し、アプリケーションプログラムに加えて CPU1 プログラムのコピーも行います。

また、アプリケーションプログラムには CPU1 のリセット解除処理が追加されます。この処理が実行された後、CPU1 プログラムは System SRAM の先頭番地(0x1000_0000)から実行されます。

各開発環境において CPU1 プログラムのデバッグを行うための詳細な手順を次のページから示します。

5.3.1 EWARM : IAR システムズ社製

1. CPU1 プログラムのビルド

Loader_application_projects¥cpu1¥RZT2ME_cpu1.eww を開き、CPU1 プログラムのビルドを行います。以下のプロジェクトオプション設定により、ローバイナリでビルド成果物が出力されます。
CPU1 プログラムをビルド後、RZ/T2ME_cpu1.eww のワークスペースを閉じてください。

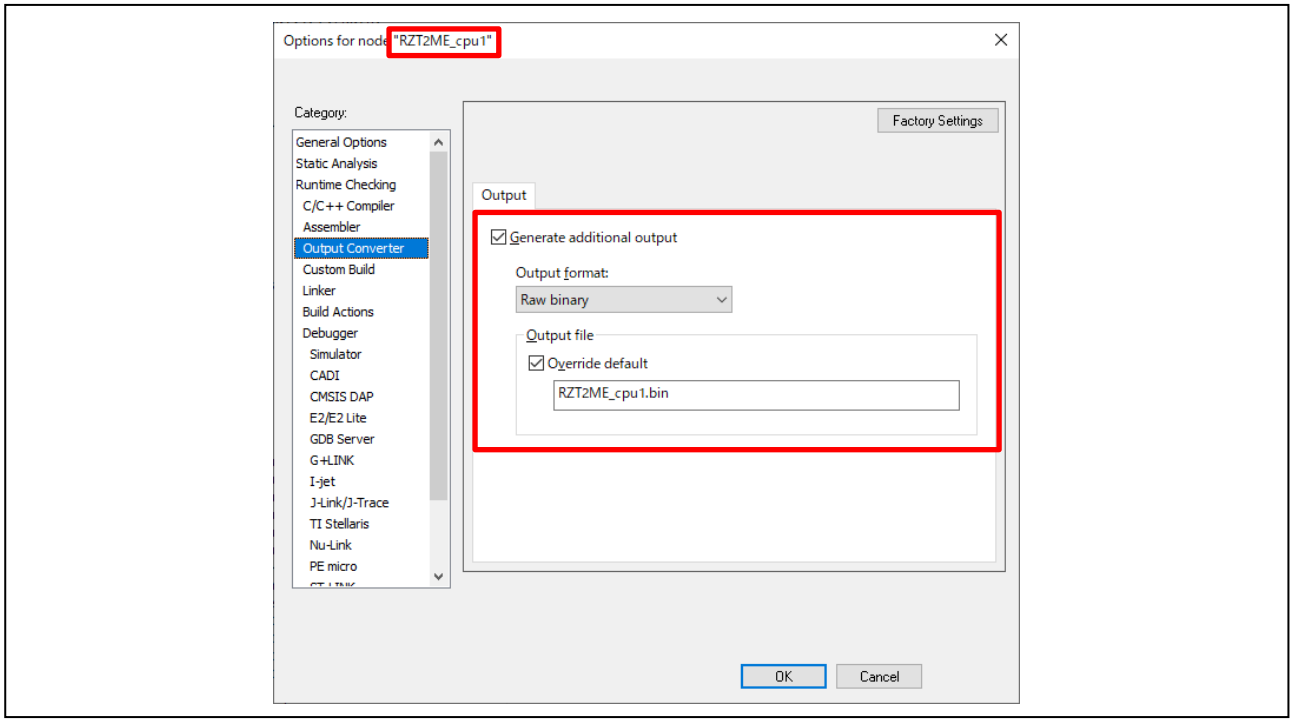

図 5-4 CPU1 プログラムのオプション設定

2. CPU1 プログラムを CPU0 ローダプログラムへリンク

CPU1 のバイナリを CPU0 のローダプログラムにリンクするために以下のプロジェクトオプション設定を追加します。

ローダプログラムのプロジェクト [オプション] -> [リンカ] -> [入力]タブ

シンボルをキープ	: CPU1_SECTION
ファイル	: $PROJ_DIR$¥..¥cpu1¥Debug¥Exe¥RZT2ME_cpu1.bin
シンボル	: CPU1_SECTION
セクション	: CPU1_SECTION
アラインメント	: 4

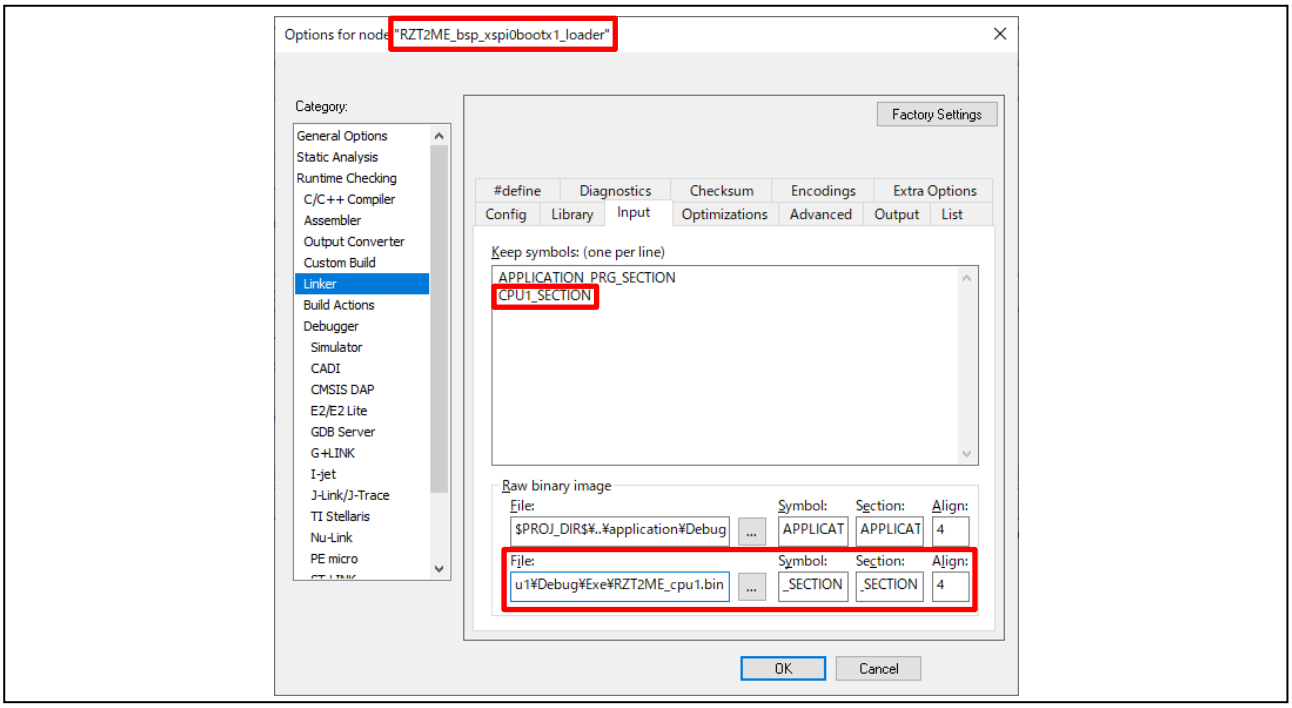

図 5-5 CPU0 のローダプログラムのオプション設定

3. USE_CPU1 定義の有効化

CPU1 プログラムを動作させるための定義を有効化します。プロジェクトオプションに定義されている「USE_CPU1」の値を 0 から 1 へ変更してください。

・ローダプログラムのプロジェクト

[オプション] -> [C/C++コンパイラ] -> [プリプロセッサ]タブ: USE_CPU1=1
[オプション] -> [リンカ] -> [設定]タブ: USE_CPU1=1

・アプリケーションプログラムのプロジェクト

[オプション] -> [C/C++コンパイラ] -> [プリプロセッサ]タブ: USE_CPU1=1

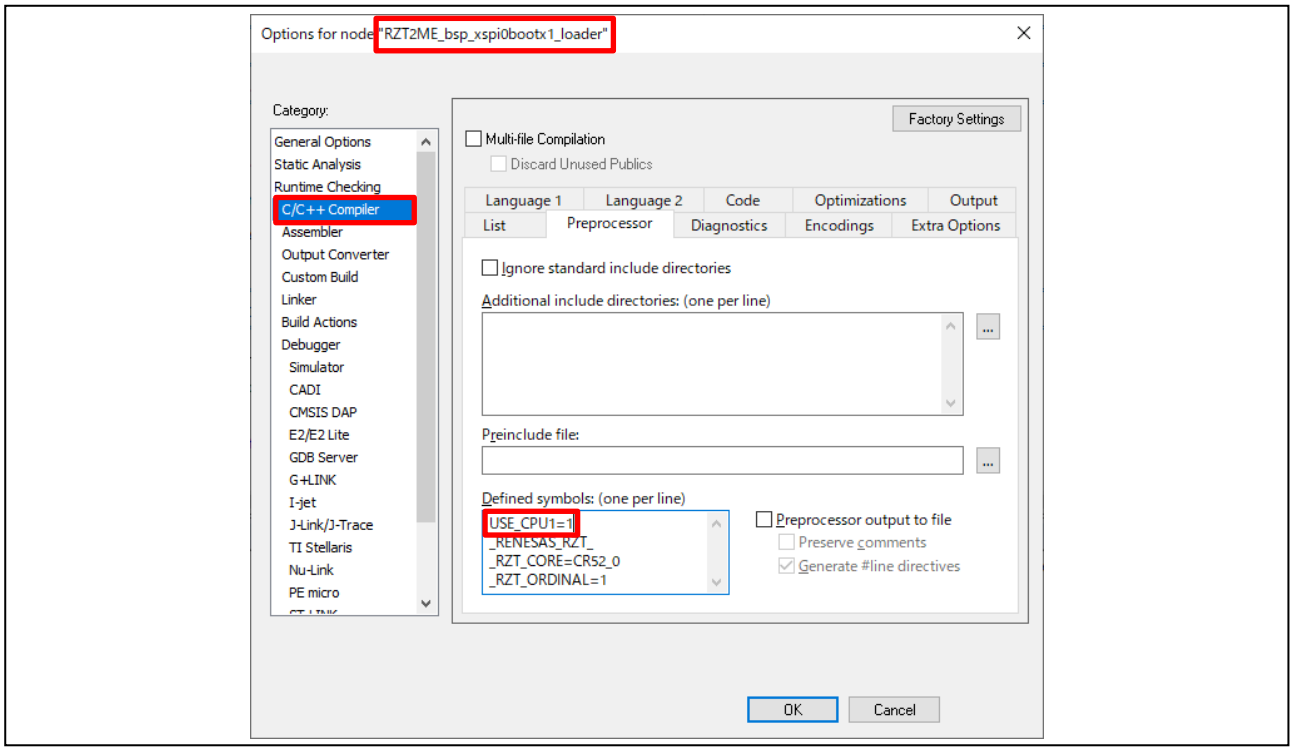

図 5-6 CPU0 のローダプログラムの USE_CPU1 定義有効化 (1/2)

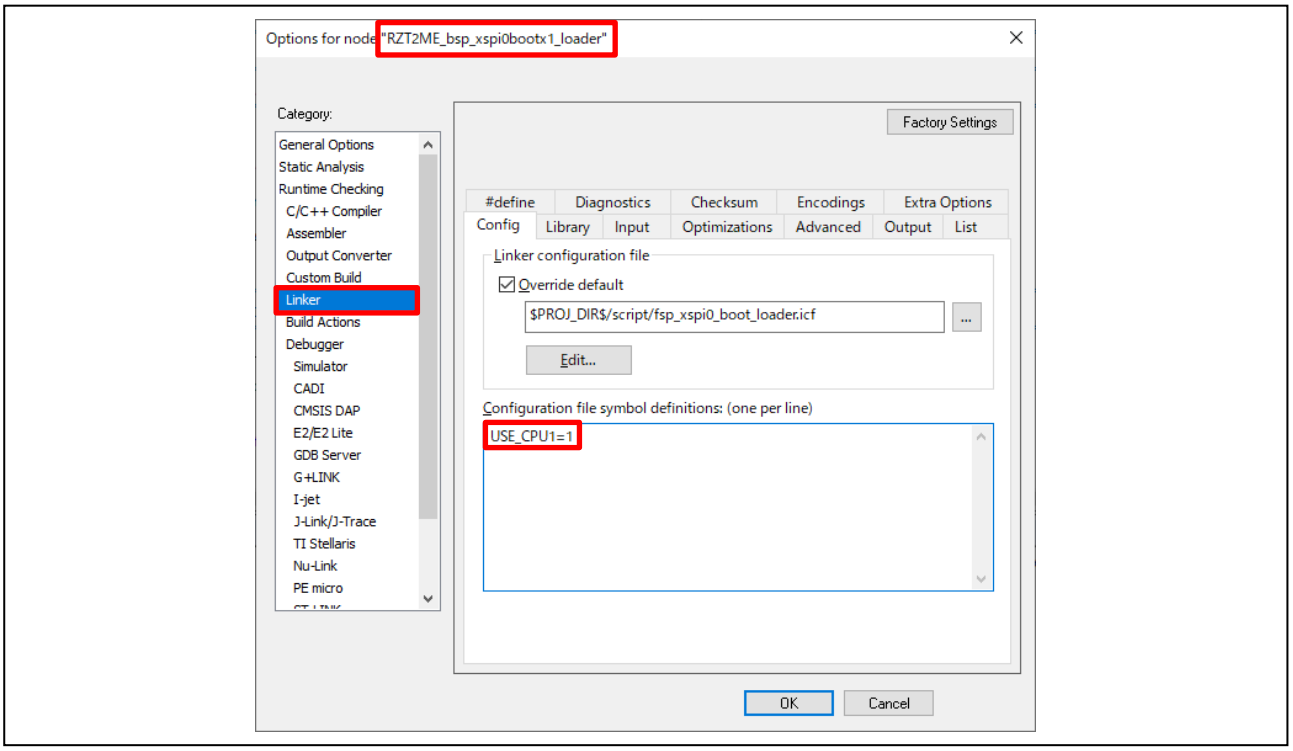

図 5-7 CPU0 のローダプログラムの USE_CPU1 定義有効化 (2/2)

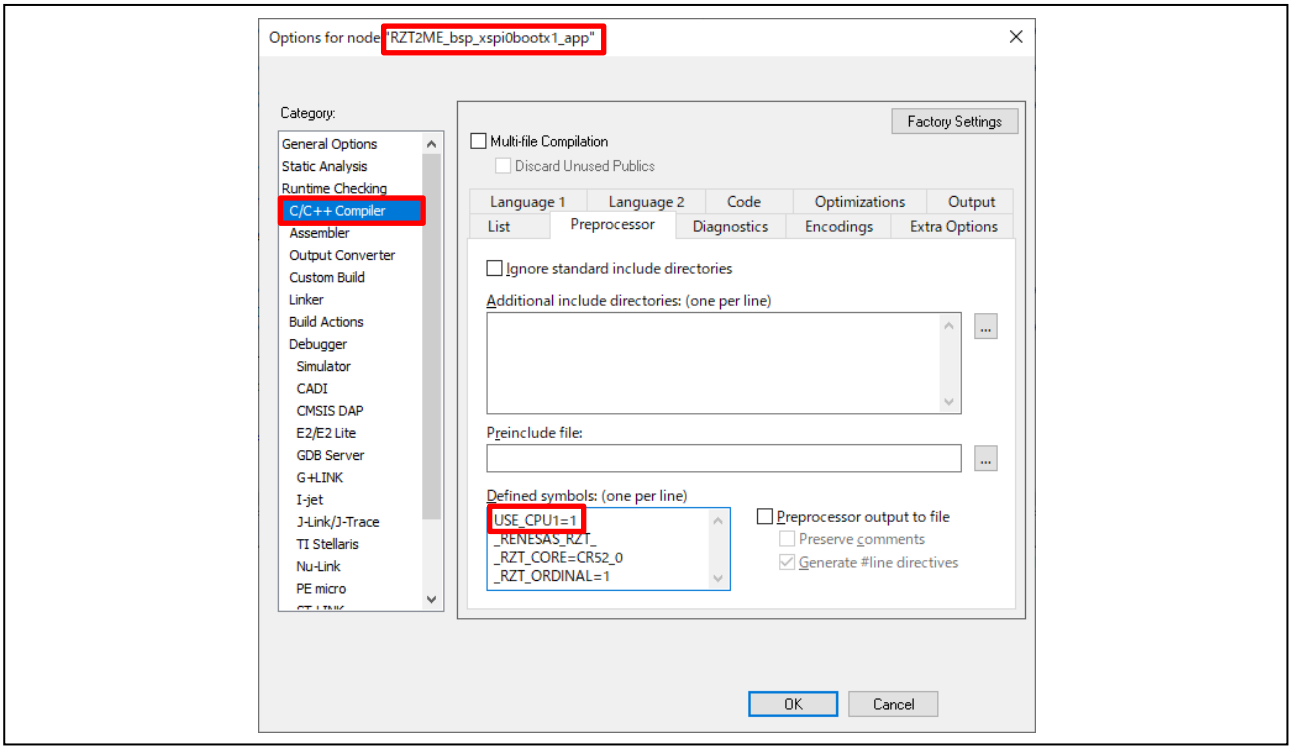

図 5-8 CPU0 のアプリケーションプログラムの USE_CPU1 定義有効化

4. CPU1 プロジェクトのデバッグを行う設定

CPU1 プログラムのデバッグを行う場合、EWARM のマルチコアデバッグ機能を使用します。以下のプロジェクトオプション設定をローダプログラムのプロジェクトに追加することで、CPU1 プロジェクトのデバッグが可能となります。

ローダプログラムのプロジェクト [オプション] -> [デバッガ] -> [マルチコア]タブ
非対称型マルチコア : “シンプル” を有効化
スレーブワークスペース : $PROJ_DIR$¥..¥cpu1¥RZT2ME_cpu1.eww
スレーブプロジェクト : RZT2ME_cpu1
スレーブ構成 : Debug

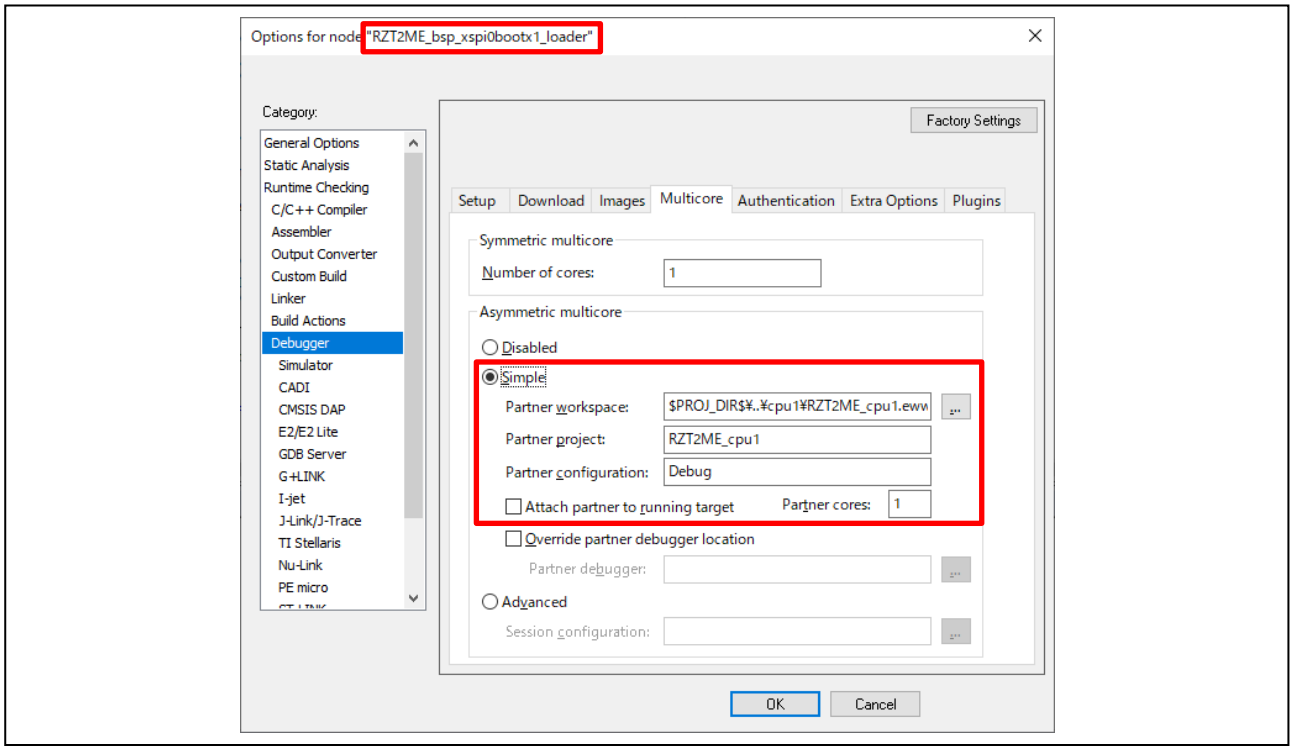

図 5-9 CPU0 のローダプログラムのマルチコアデバッグ設定

5. プロジェクトの再ビルドと実行

ローダプログラムのプロジェクトを再ビルド後、「ダウンロードしてデバッグ」をクリックしてローダプログラムのデバッグを開始してください。

デバッグ接続時、CPU1 のプロジェクトも自動的に立ち上がります。以降、CPU0 と CPU1 プロジェクトのデバッグが可能となります。
CPU1 プログラムが実行されると、LED2 と LED3 が点滅します。

5.3.2 e2studio : Renesas 社製

1. CPU1 プログラムのビルド

RZT2ME_cpu1 プロジェクトのビルドを行います。以下のプロジェクトオプション設定により、ローバイナリでビルド成果物が出力されます。

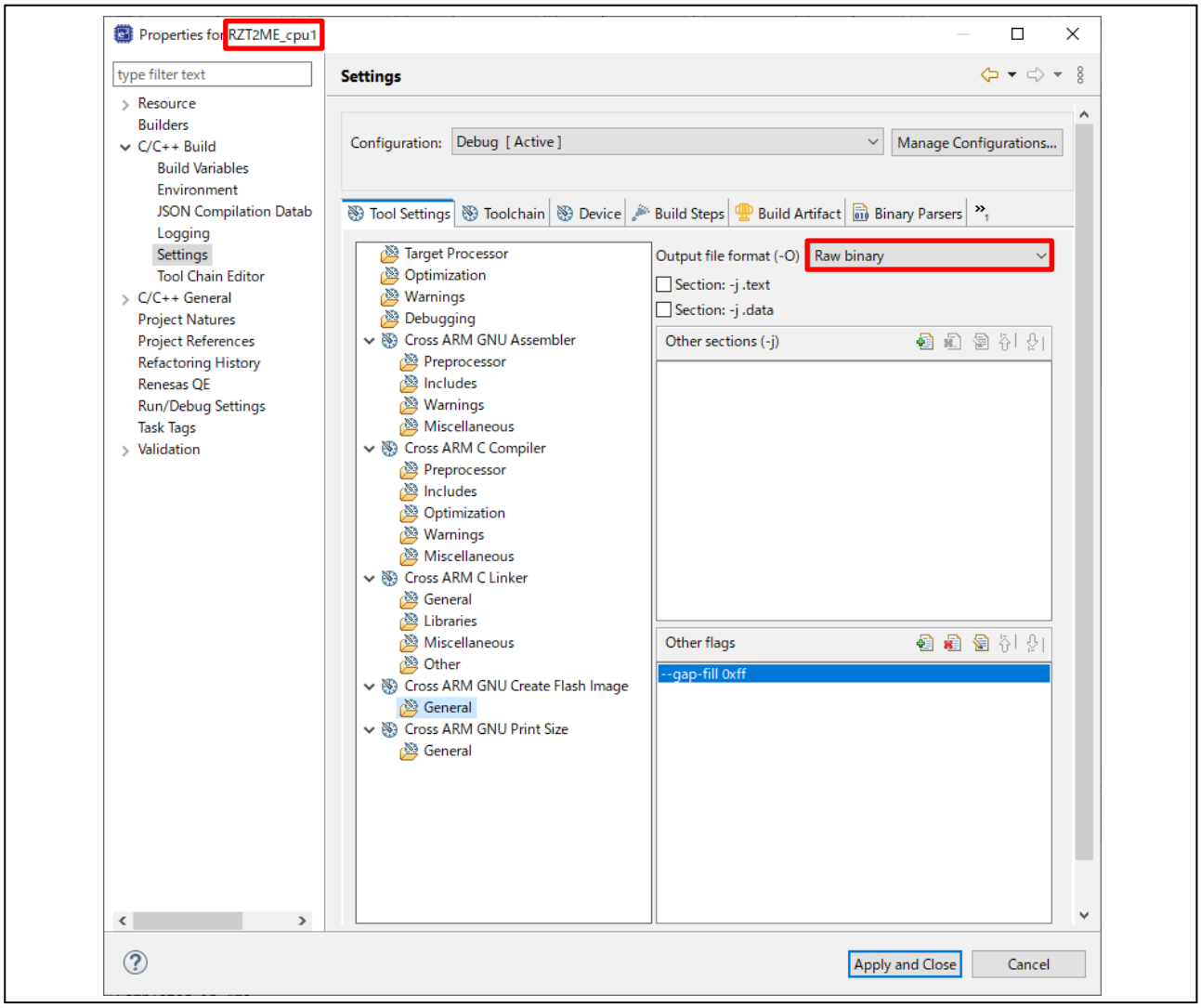

図 5-10 CPU1 プログラムのオプション設定

2. CPU1 プログラムを CPU0 ローダプログラムへリンク

CPU1 のバイナリを CPU0 のローダプログラムにリンクするためのセクション定義をリンカスクリプトファイルに追加します。

fsp_xspi0_boot_loader.ld (ローダプログラムのリンカスクリプト)

```
SECTIONS
{
    .IMAGE_APP_RAM 0x10080000 : AT (0x10080000)
    {
        IMAGE_APP_RAM_start = .;
        KEEP(*(APP_IMAGE_RAM))
    }
    .IMAGE_APP_FLASH_section 0x60100000 : AT (0x60100000)
    {
        IMAGE_APP_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
        IMAGE_APP_FLASH_section_end = .;
    }
    .IMAGE_CPU1_RAM 0x10000000 : AT (0x10000000) /* CPU1 program, RAM section for execution. */
    {
        IMAGE_CPU1_RAM_start = .;
        KEEP(*(CPU1_IMAGE_RAM))
    }
    .IMAGE_CPU1_FLASH_section 0x60200000 : AT (0x60200000) /* CPU1 program, ROM section. */
    {
        IMAGE_CPU1_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_CPU1_FLASH_section))
        IMAGE_CPU1_FLASH_section_end = .;
    }
    .loader_param LOADER_PARAM_ADDRESS : AT (LOADER_PARAM_ADDRESS)
    {
        KEEP(*(.loader_param))
    } > FLASH_CONTENTS
    .flash_contents FLASH_CONTENTS_ADDRESS : AT (FLASH_CONTENTS_ADDRESS)
    {
        _mtext = .;
        . = (1 == _RZT_ORDINAL) ? . + (_text_end - _text_start) : .;
        . = ALIGN(8);
        _mdata = .;
        . = (1 == _RZT_ORDINAL) ? . + (_data_end - _data_start) : .;
        flash_contents_end = .;
    } > FLASH_CONTENTS
~~~~~~~~
}

IMAGE_APP_FLASH_section_size = SIZEOF(.IMAGE_APP_FLASH_section);
IMAGE_CPU1_FLASH_section_size = SIZEOF(.IMAGE_CPU1_FLASH_section);
```

3. USE_CPU1 定義の有効化

CPU1 プログラムを動作させるための定義を有効化します。**ローダプログラム**と**アプリケーションプログラム**のプロジェクトオプションに定義されている「USE_CPU1」の値を 0 から 1 へ変更してください。

[Properties] -> [C/C++ Build] -> [Settings] -> [Tool Settings]
[Cross ARM GNU Assembler] -> [Preprocessor]: USE_CPU1=1
[Cross ARM C Compiler] -> [Preprocessor]: USE_CPU1=1

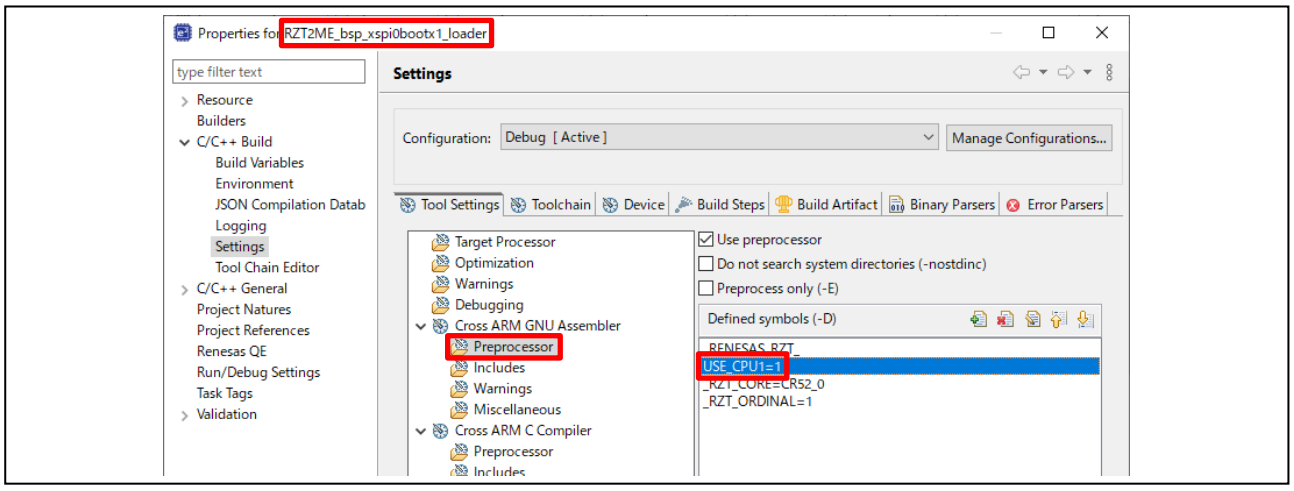

図 5-11 CPU0 のローダプログラムの USE_CPU1 定義有効化 (1/2)

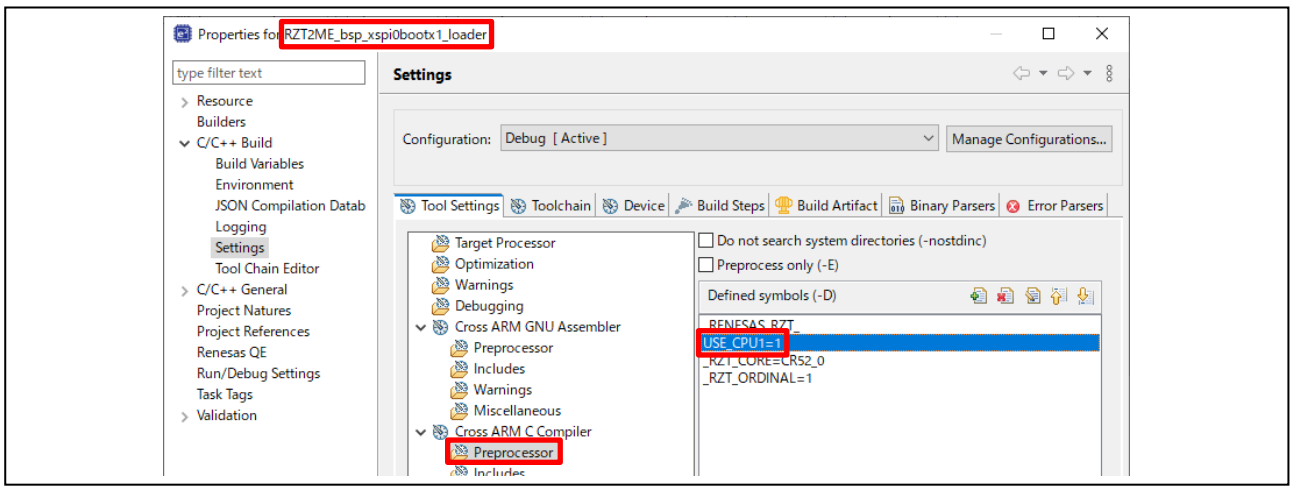

図 5-12 CPU0 のローダプログラムの USE_CPU1 定義有効化 (2/2)

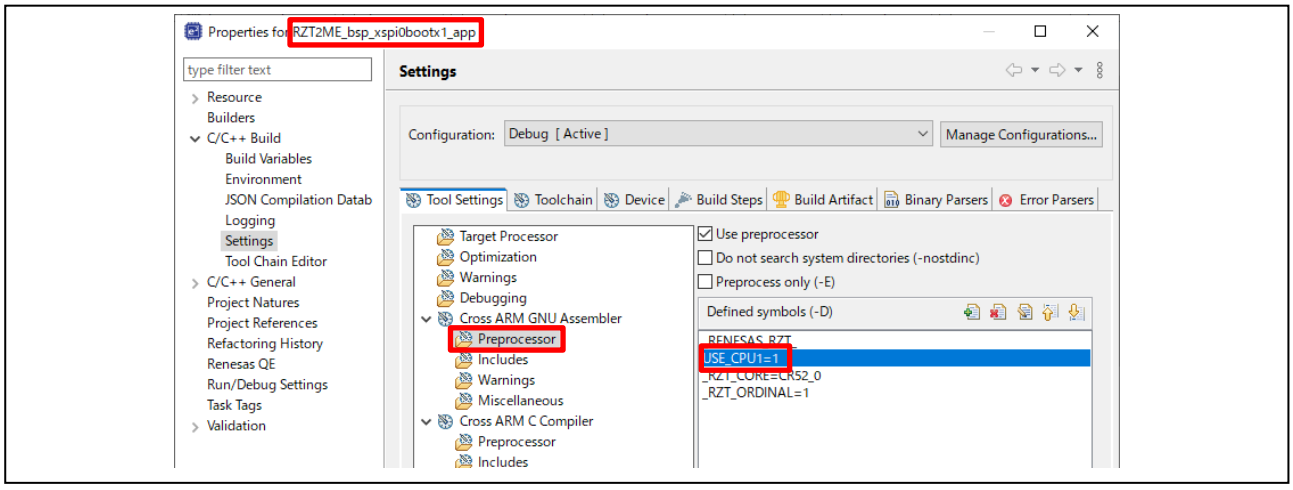

図 5-13 CPU0 のアプリケーションプログラムの USE_CPU1 定義有効化 (1/2)

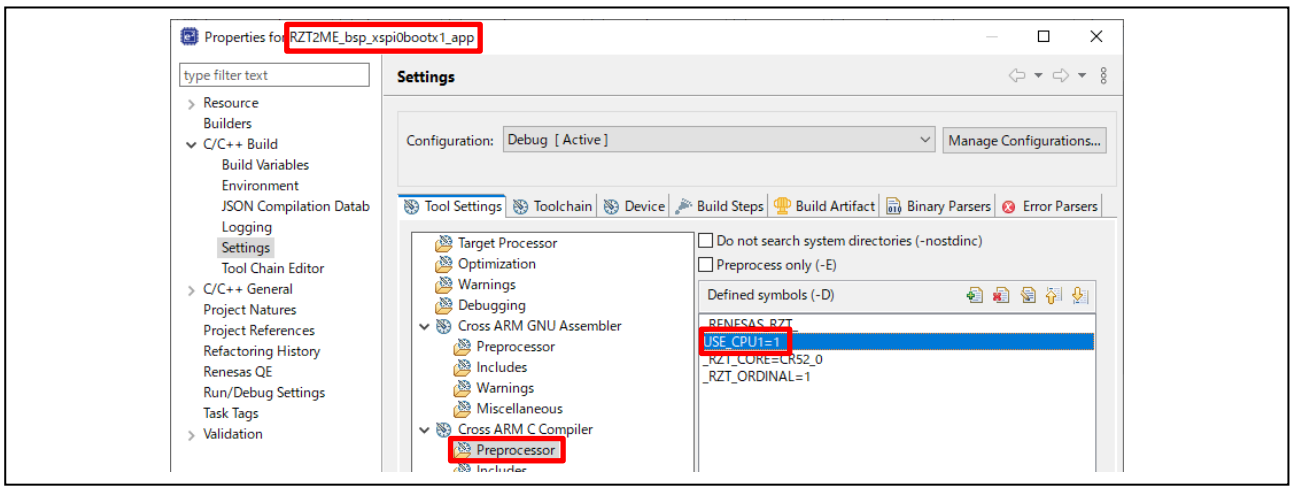

図 5-14 CPU0 のアプリケーションプログラムの USE_CPU1 定義有効化 (2/2)

4. プロジェクトの再ビルドと実行

ローダプログラムのプロジェクトを再ビルド後、ローダプログラムのデバッグを開始してください。デバッグ接続時に CPU1 プログラムも同時にフラッシュメモリに書き込まれます。

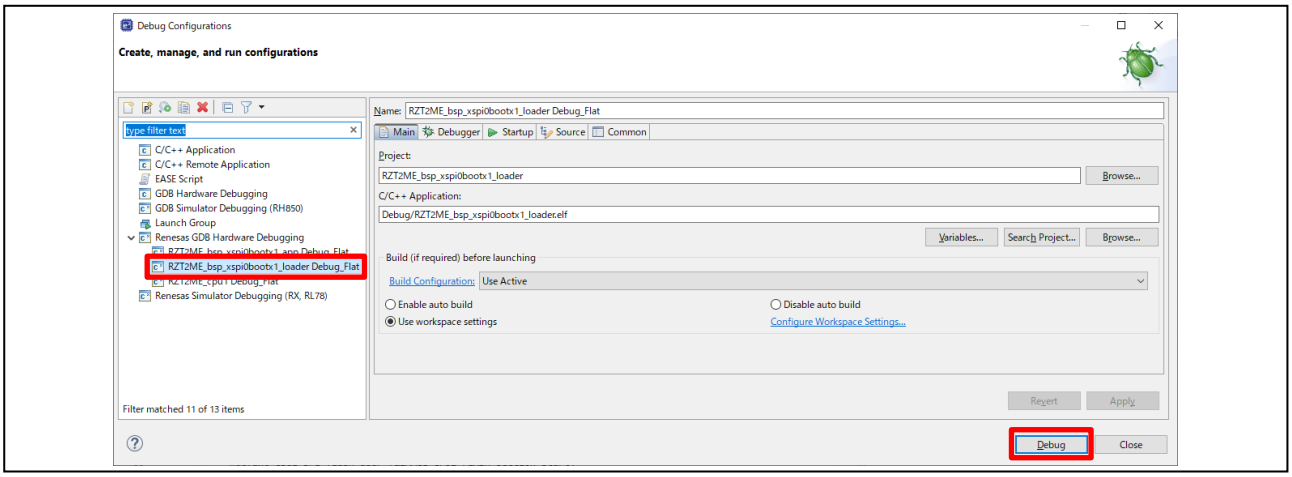

図 5-15 CPU0 のローダプログラムのデバッグ開始

CPU1 プログラムのデバッグを行う場合、ローダプログラムのデバッグ接続を行った後に CPU1 プロジェクトのデバッグ接続を開始してください。

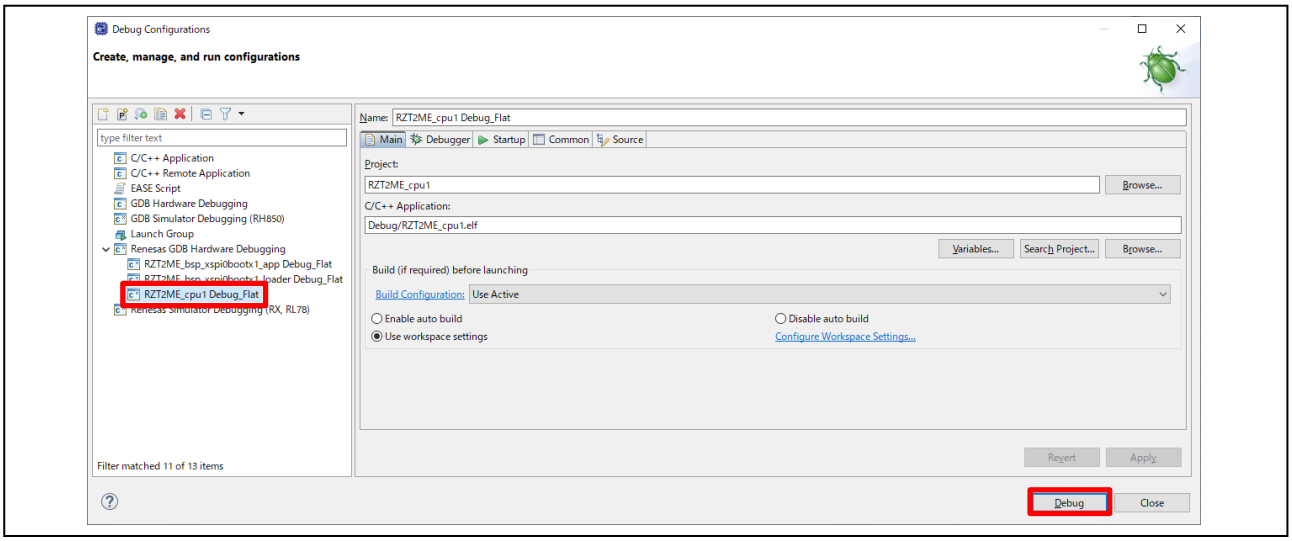

図 5-16 CPU1 プロジェクトのデバッグ開始

この際、下記のメッセージが表示されますが、「No」を選択してください。

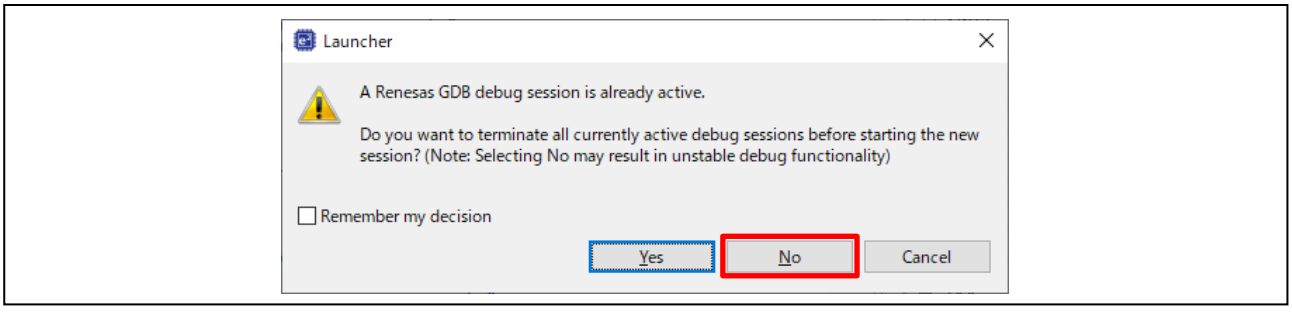

図 5-17 CPU0 のローダプログラムのデバッグ継続確認画面

CPU1 プロジェクトのデバッグ接続が成功すると CPU0 と CPU1 がデバッガに接続された状態となります。以降、CPU0 と CPU1 プロジェクトのデバッグが可能となります。

画面左にある「Debug」ビューの Thread を選択することで、デバッグ対象コアを CPU0 と CPU1 で切り替えることが可能です。

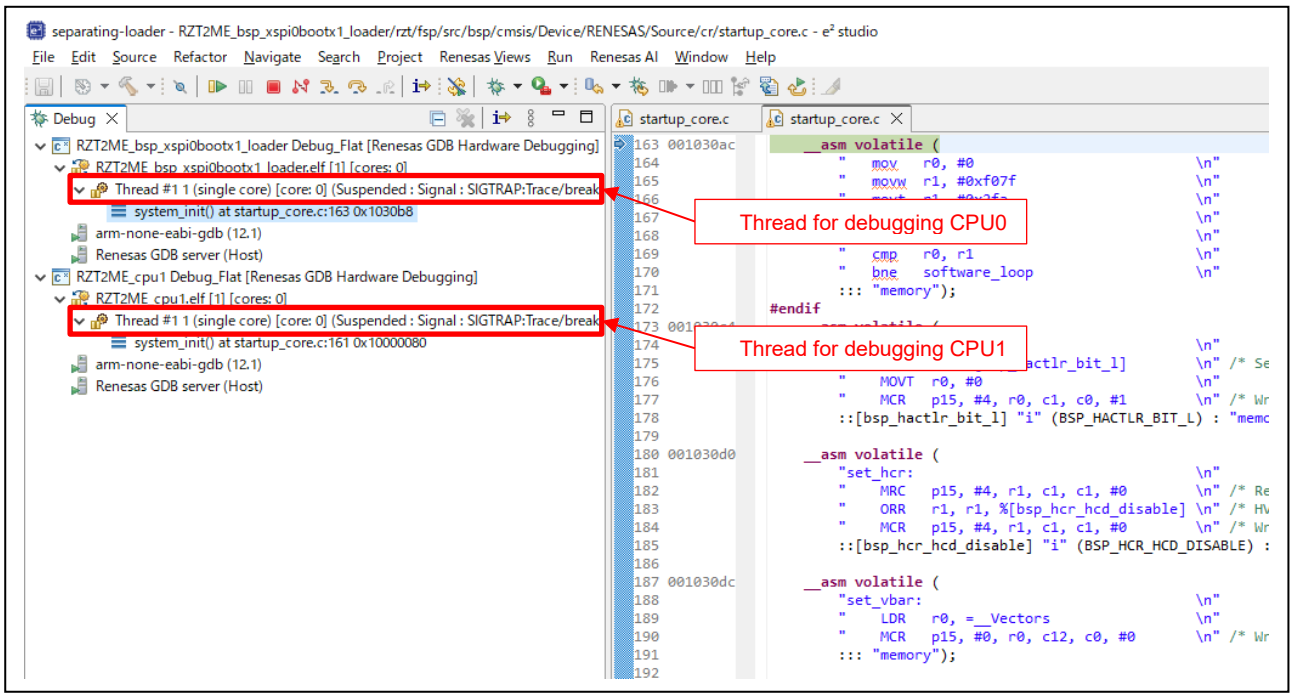

図 5-18 e²studio のデバッグ

CPU0 プロジェクトの Thread を選択し実行すると、ローダプログラムとアプリケーションプログラムが実行され、LED0 と LED1 が点滅します。
CPU1 プロジェクトの Thread を選択し実行すると、CPU1 プログラムが実行され、LED2 と LED3 が点滅します。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2025.06.16	-	初版

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. 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。

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。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、お客様の責任において、お客様の機器・システムを設計ください。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. 当社製品または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. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 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。
6. 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。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 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。
9. 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。
10. 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。
11. 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。
12. 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。
13. 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。
14. 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.5.0-1　2020.10)

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。