RZ/T2H 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、xSPI1 ブートモード (x1 ブートシリアルフラッシュ)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/T2H グループ

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/T2H グループ (R9A09G077M44GBG)
動作周波数	CPU core0 : 1000MHz (Arm® Cortex®-R52) CPU core0 : 1200MHz (Arm® Cortex®-A55)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 9.60.3 + patch • Renesas 製 e² studio 2025-04.1 (25.4.1) (R20250508-1327)
動作モード	• xSPI0 ブートモード(x1 シリアルフラッシュ) • xSPI1 ブートモード(x1 シリアルフラッシュ)
使用ボード	RZ/T2H Evaluation Board
Flexible Software Package (FSP)	Version 3.0.0 (RZ/T2 FSP)

【注】 1. Cortex-A55 CPU core0 を使用する場合は「5.3 Cortex-A55 CPU0 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。

```
RZT2H_loader_application
 ├──r01an7758jj0301-rzt2h.pdf
 ├──r01an7758ej0301-rzt2h.pdf
 ├──iccarm : EWARM 環境用
 │    └──xspi0bootx1 : SPI0 フラッシュメモリ用サンプルプログラム
 │    │    └──Loader_application_projects.zip
 │    │         ├──RZT2H_bsp_xspi0bootx1_app : アプリケーションプログラム用プロジェクト
 │    │         ├──RZT2H_bsp_xspi0bootx1_loader : ローダプログラム用プロジェクト
 │    │         ├──RZT2H_bsp_xspi0bootx1_app_CA55_0 : Cortex-A55 CPU0 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
 │    │         └──RZT2H_bsp_xspi0bootx1_separating_loader.eww : EWARM ワークスペース
 │    └──xspi1bootx1 : SPI1 フラッシュメモリ用サンプルプログラム
 │         └──Loader_application_projects.zip
 │              ├──RZT2H_bsp_xspi1bootx1_app : アプリケーションプログラム用プロジェクト
 │              ├──RZT2H_bsp_xspi1bootx1_loader : ローダプログラム用プロジェクト
 │              ├──RZT2H_bsp_xspi1bootx1_app_CA55_0  : Cortex-A55 CPU0 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
 │              └──RZT2H_bsp_xspi1bootx1_separating_loader.eww : EWARM ワークスペース
 └──gcc : e2 studio 環境用
      └──xspi0bootx1 : SPI0 フラッシュメモリ用サンプルプログラム
      │    └──Loader_application_projects.zip
      │         ├──RZT2H_bsp_xspi0bootx1_app : アプリケーションプログラム用プロジェクト
      │         ├──RZT2H_bsp_xspi0bootx1_loader : ローダプログラム用プロジェクト
      │         └──RZT2H_bsp_xspi0bootx1_app_CA55_0 : Cortex-A55 CPU0 用プロジェクト
      └──xspi1bootx1 : SPI1 フラッシュメモリ用サンプルプログラム
           └──Loader_application_projects.zip
                ├──RZT2H_bsp_xspi1bootx1_app : アプリケーションプログラム用プロジェクト
                ├──RZT2H_bsp_xspi1bootx1_loader : ローダプログラム用プロジェクト
                └──RZT2H_bsp_xspi1bootx1_app_CA55_0 : Cortex-A55 CPU0 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 SPI0 フラッシュメモリ用と SPI1 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 Cortex-A55 CPU0 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、RZ/T2H Evaluation Board 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(1/2)

プロジェクト	SW14-1	SW14-2	SW14-3	SW14-4	SW14-6
xSPI0 ブートモード版	ON	ON	ON	OFF	OFF
xSPI1 ブートモード版	ON	OFF	ON	OFF	OFF

表 1-3 スイッチの設定(2/2)

プロジェクト	SW1-6	SW5-5	SW5-6	SW8-9	SW8-10
xSPI0 ブートモード版	-	OFF	ON	ON	OFF
xSPI1 ブートモード版	ON	-	-	ON	OFF

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
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、xSPI1 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/T2H および RZ/N2H 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
XSPI0_RESET0#	出力	スレーブ 0 のマスタリセットステータス出力
XSPI0_CS0#	出力	CS0 空間に接続された OSPI フラッシュメモリへのデバイス選択信号出力
XSPI0_DS	入出力	読み出しデータストローブ／書き込みデータマスク
XSPI0_ECS#	入力	スレーブ 0 のエラー訂正状態入力
XSPI0_CKP	出力	クロック正出力
XSPI0_CKN	出力	クロック負出力
XSPI0_IO0 ~ XSPI0_IO7	入出力	データ入出力
XSPI1_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
XSPI1_CKP	出力	クロック出力
XSPI1_IO0 ~ XSPI1_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” (xSPI1 ブートモード)
MD1	入力	
MD2	入力	
P23_1	出力	LED0 の点灯および消灯
P06_7	出力	LED2 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(xSPI0 ブート/xSPI1 ブート)に対応する外付けフラッシュ(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(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。

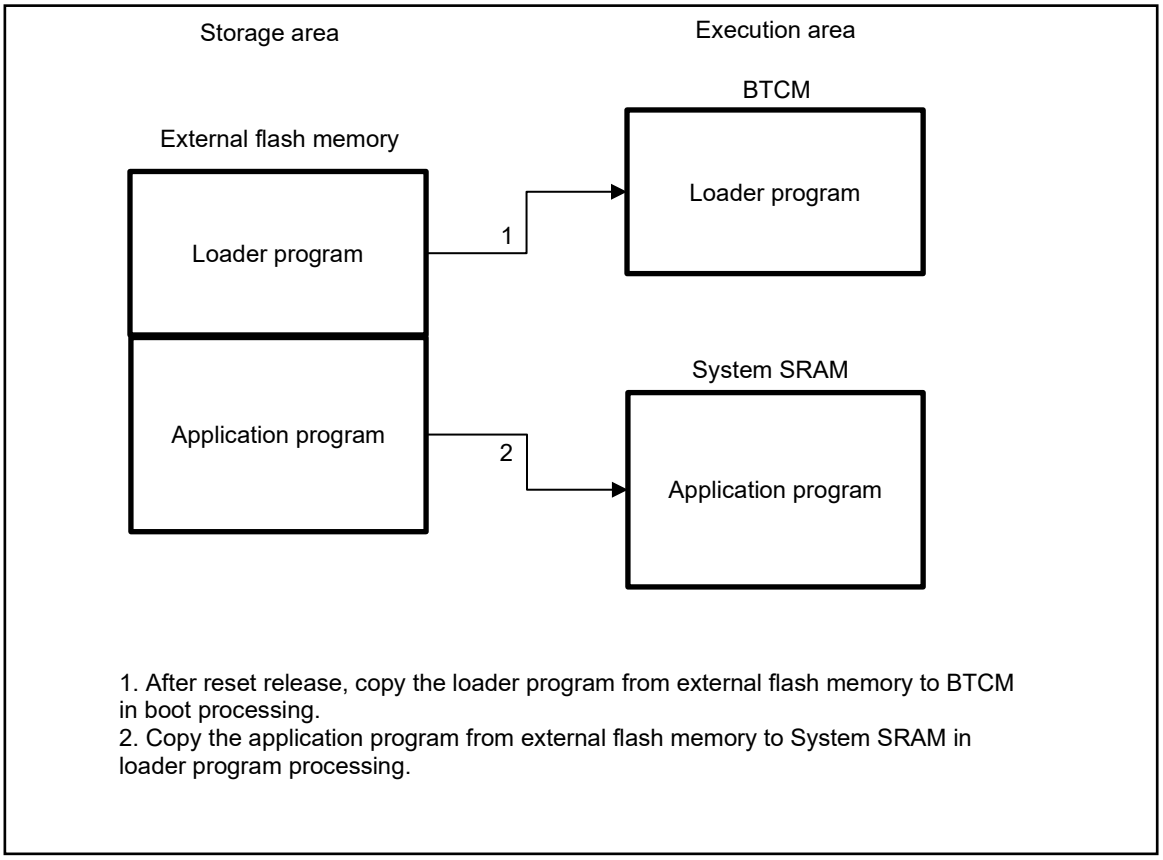

図 3-1 動作概要

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(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 LED2 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。

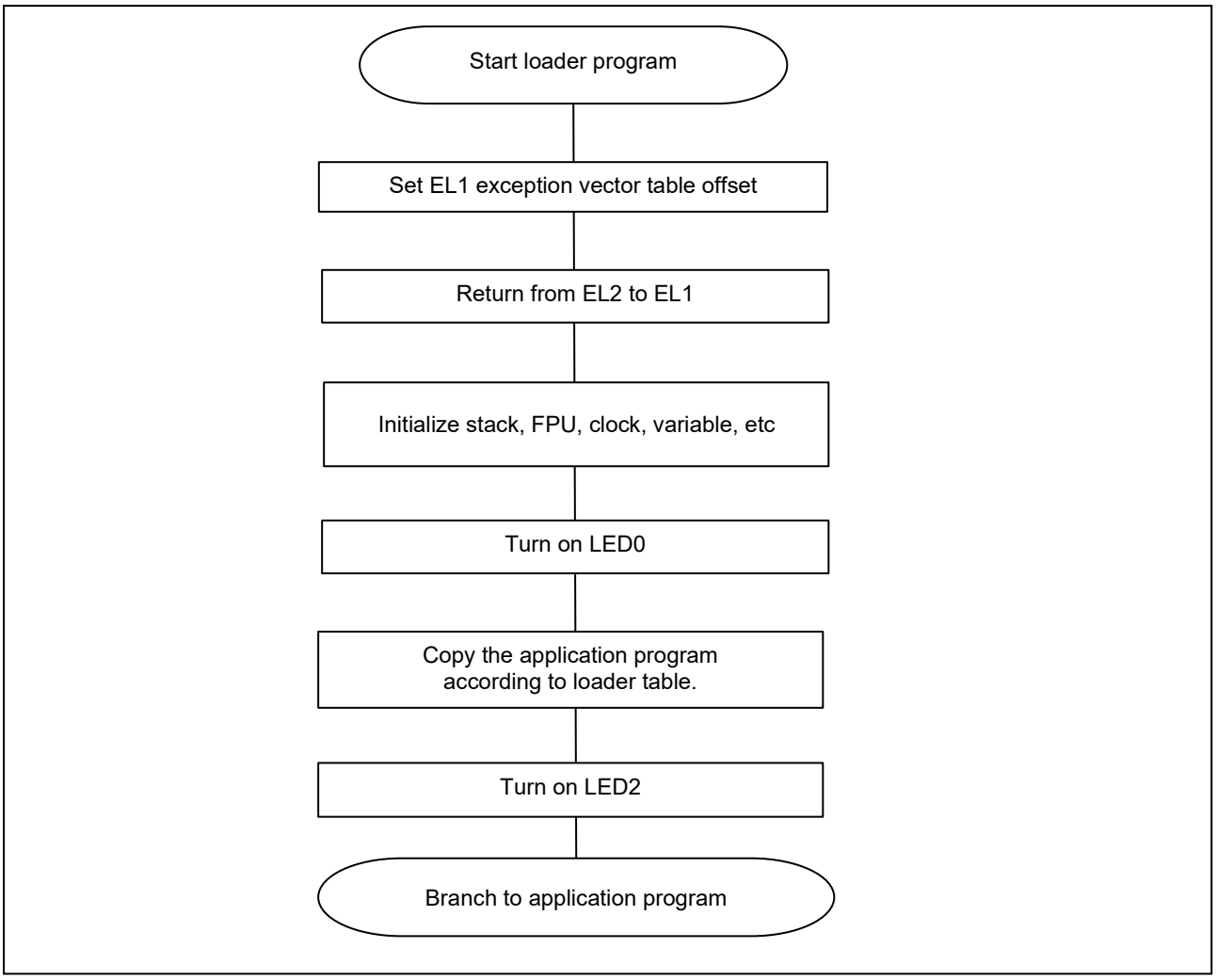

図 3-2 ローダプログラムの動作概要

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 LED2 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 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。

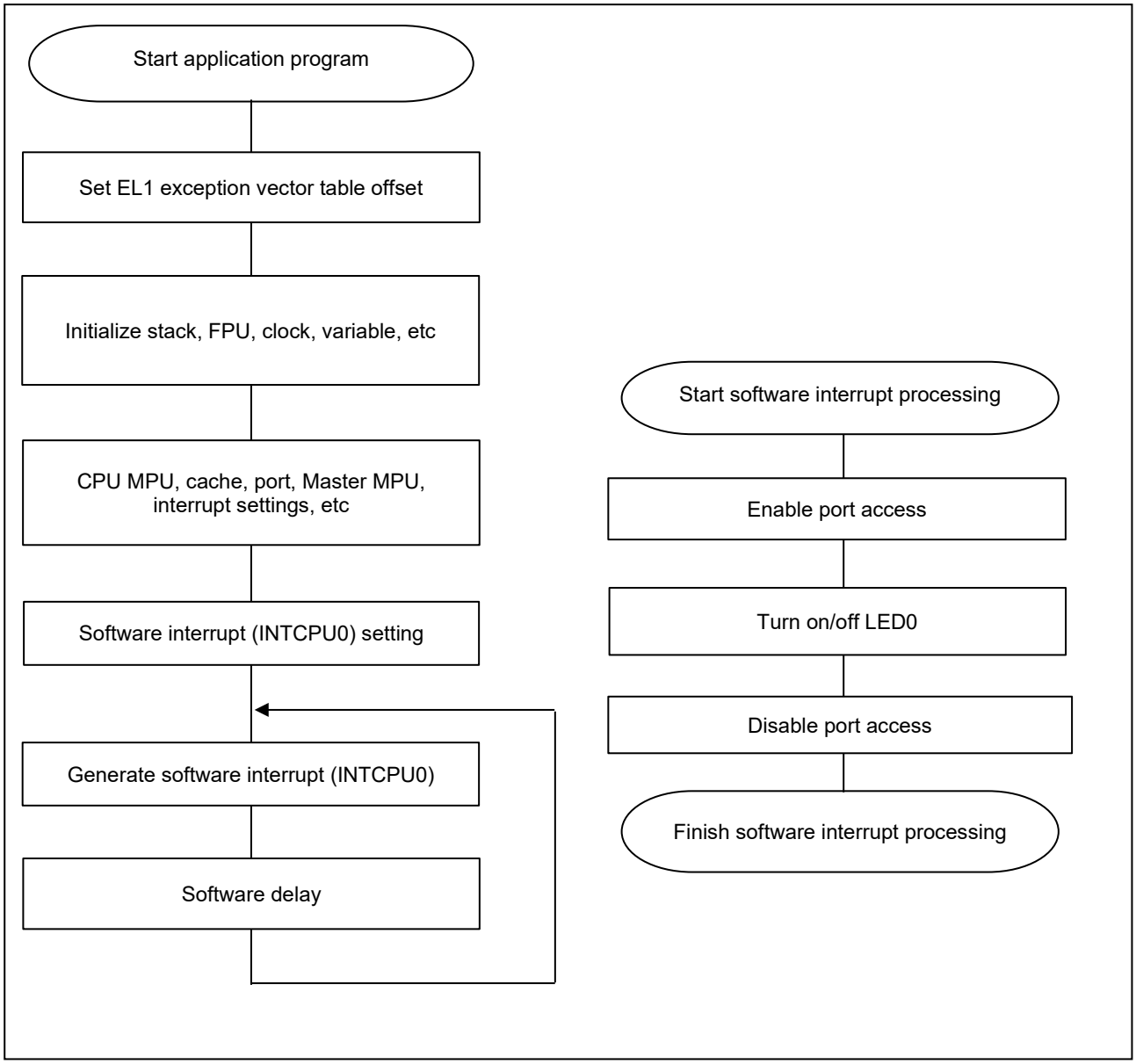

図 3-3 アプリケーションプログラムの動作概要

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	0x4001_0000	0x1008_0000	0x0000_3670	0x1
1 注 1 注 2	0xFFFF_FFFF or 0x4004_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_A0A8	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. Cortex-A55 CPU0 プログラムを有効にした場合、テーブル 1 は Cortex-A55 CPU0 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 Cortex-A55 CPU0 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(xSPI1 ブートモード版)

テーブル	Src	Dst	Size	Enable flag
0	0x5001_0000	0x1008_0000	0x0000_3670	0x1
1 注 1 注 2	0xFFFF_FFFF or 0x5004_0000	0xFFFF_FFFF or 0x1000_0000	0xFFFF_FFFF or 0x0000_A0A8	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. Cortex-A55 CPU0 プログラムを有効にした場合、テーブル 1 は Cortex-A55 CPU0 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 Cortex-A55 CPU0 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
0x4000_0000	ローダ用パラメータ	-
0x4000_0050	ローダプログラム	0x0010_2000 (BTCM)
0x4008_0000	ローダテーブル	-
0x4001_0000	アプリケーションプログラム	0x1008_0000 (System SRAM)
0x4004_0000 注	Cortex-A55 CPU0 プログラム	0x1000_0000 (System SRAM)

【注】 Cortex-A55 CPU0 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 Cortex-A55 CPU0 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(xSPI1 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
0x5000_0000	ローダ用パラメータ	-
0x5000_0050	ローダプログラム	0x0010_2000 (BTCM)
0x5008_0000	ローダテーブル	-
0x5001_0000	アプリケーションプログラム	0x1008_0000 (System SRAM)
0x5004_0000 注	Cortex-A55 CPU0 プログラム	0x1000_0000 (System SRAM)

【注】 Cortex-A55 CPU0 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 Cortex-A55 CPU0 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
CA55_CPU0_PRG_RBLOCK[注2]	Cortex-A55 CPU0 プログラム領域(格納用)	Flash
CA55_CPU0_PRG_WBLOCK[注2]	Cortex-A55 CPU0 プログラム領域(実行用)	System SRAM

【注】 1. xSPI0、xSPI1 ブート版ではシリアルフラッシュメモリが格納領域となります。
2. Cortex-A55 CPU0 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 Cortex-A55 CPU0 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、xSPI1 ブート版ではシリアル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 .warm_start	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
.cpu0_loader_table	ローダテーブル格納領域	Flash

【注】 1. xSPI0、xSPI1 ブート版ではシリアルフラッシュメモリが格納領域となります。
2. Cortex-A55 CPU0 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 Cortex-A55 CPU0 プログラムのデバッグ方法」を参照してください。
3. スタック領域およびヒープ領域は省略されています。

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 .warm_start	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、xSPI1 ブート版ではシリアルフラッシュメモリが格納領域となります。
2. スタック領域およびヒープ領域は省略されています。

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 ~ 0x100F_FFFF	領域 2 ノーマル、キャッシュ可、非共有
システム SRAM	0x1010_0000 ~ 0x101F_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 xSPI0, xSPI1 CS0, CS2, CS3, CS5	0x4000_0000 ~ 0x5FFF_FFFF	領域 6 ノーマル、キャッシュ不可、共有
R-Bus ノンセーフティ/セーフティ周辺モジュール 0	0x8000_0000 ~ 0x81FF_FFFF	領域 8 デバイス(nGnRE)、命令フェッチ不可
R-Bus ノンセーフティ周辺モジュール 1	0x8800_0000 ~ 0x89FF_FFFF	領域 9 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/T2H 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
Cortex-A55 CPU0 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	スタックの設定を行い、fpu_slavetcm_init へ分岐します。
引数	なし
リターン値	なし
補足	なし

3.4.3 fpu_slavetcm_init

fpu_slavetcm_init	
概要	システム初期設定部 3
宣言	void fpu_slavetcm_init (void)
説明	FPU の設定などを行い、SystemInit へ分岐します。
引数	なし
リターン値	なし
補足	なし

3.4.4 SystemInit

SystemInit	
概要	システム初期設定部 4
宣言	void SystemInit (void)
説明	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

3.4.5 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 LED2 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 が点滅します。
引数	なし
リターン値	なし
補足	なし

3.4.6 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。

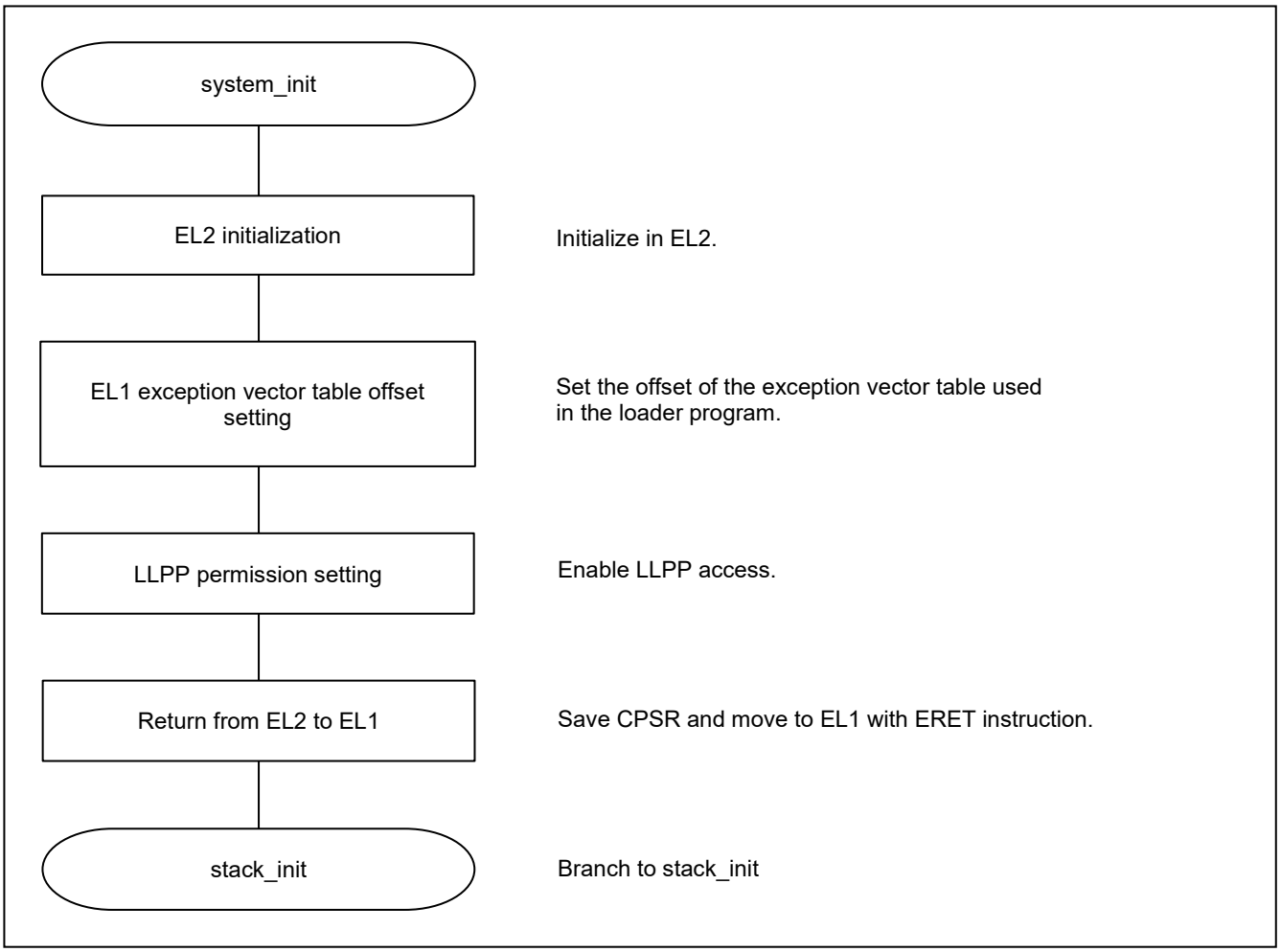

図 3-4 system_init 処理(ローダプログラム)

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。

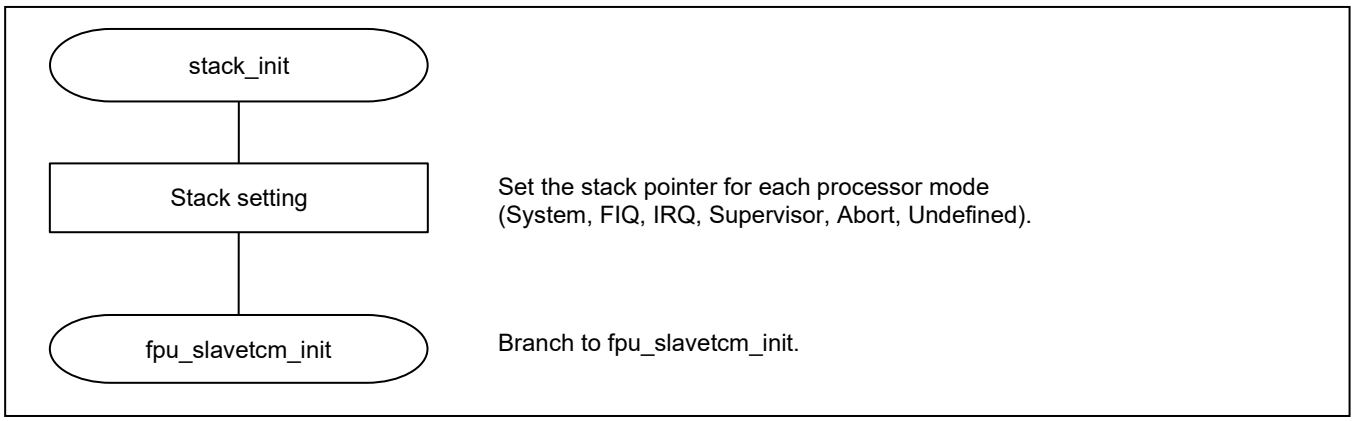

図 3-5 stack_init 処理(ローダプログラム)

3.5.1.3 fpu_slavetcm_init 処理

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

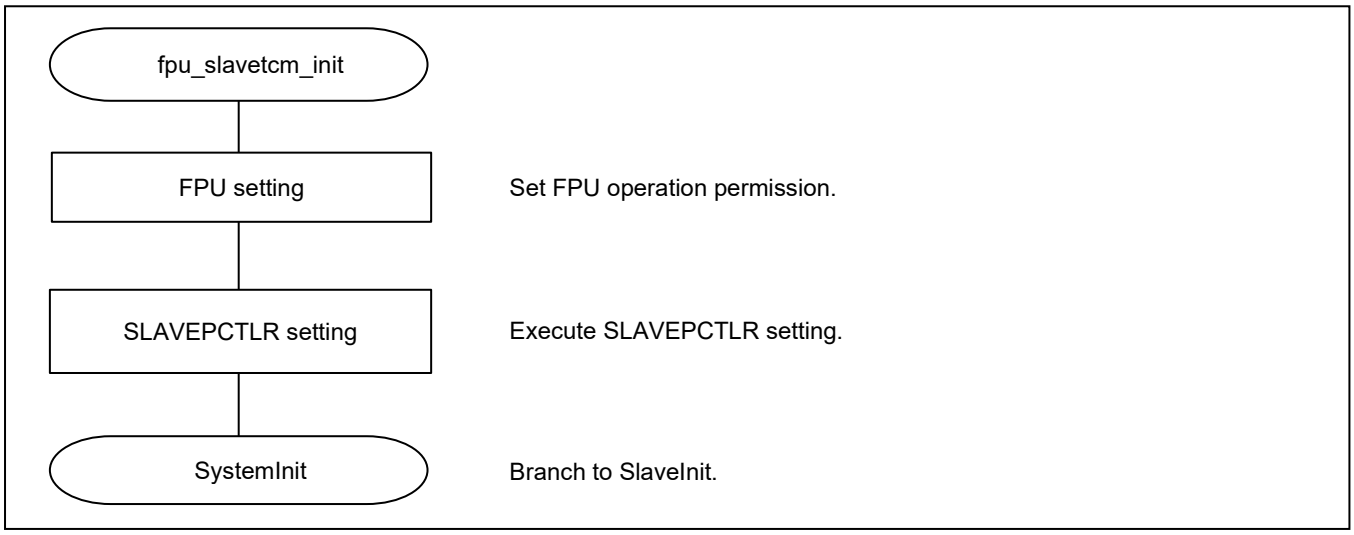

図 3-6 fpu_slavetcm_init 処理(ローダプログラム)

3.5.1.4 SystemInit 処理

図 3-7 にローダプログラムの SystemInit 処理のフローチャートを示します。

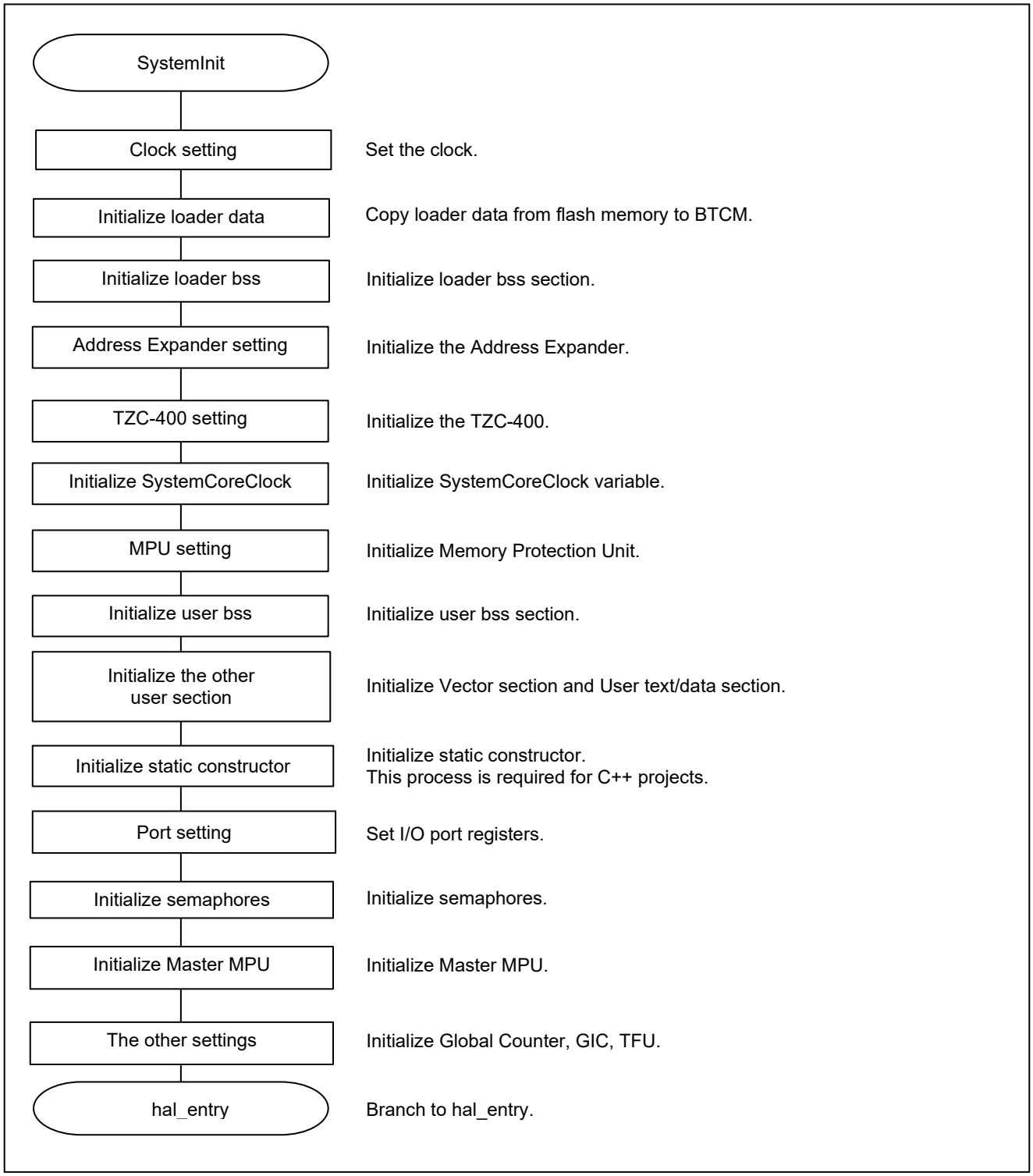

図 3-7 SystemInit 処理(ローダプログラム)

3.5.1.5 hal_entry 処理

図 3-8 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。

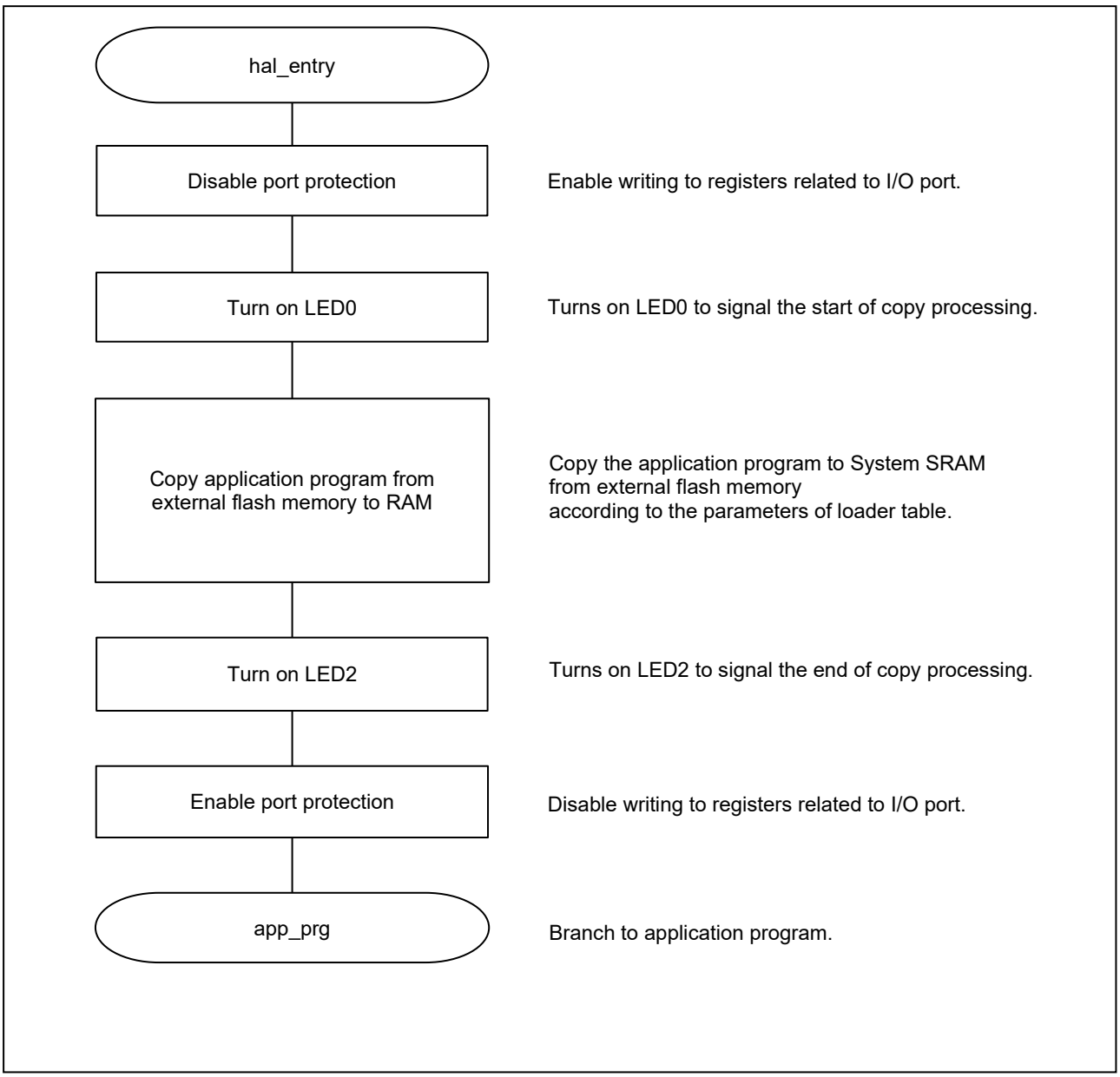

図 3-8 hal_entry 処理(ローダプログラム)

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 3-9 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。

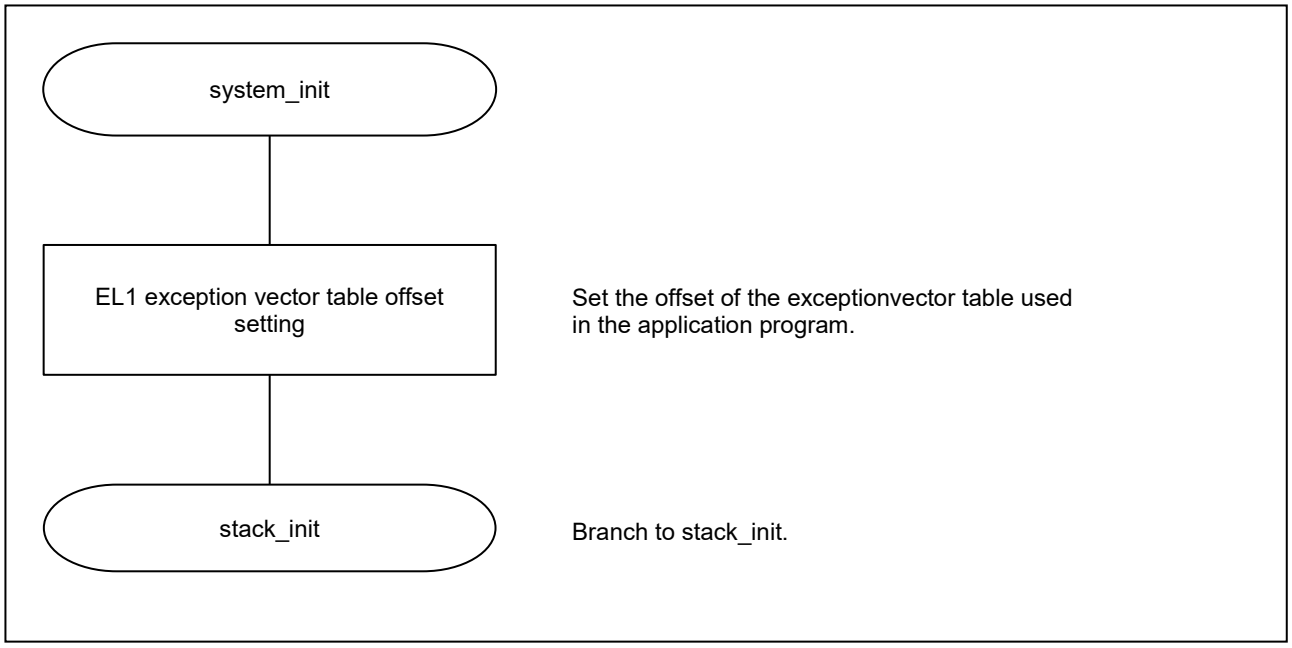

図 3-9 system_init 処理(アプリケーションプログラム)

3.5.2.2 stack_init 処理

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はローダプログラム（図 3-5）と同様です。

3.5.2.3 fpu_slavetcm_init 処理

アプリケーションプログラムの fpu_slavetcm_init 処理のフローチャートはローダプログラム（図 3-6）と同様です

3.5.2.4 SystemInit 処理

図 3-10 にローダプログラムの SystemInit 処理のフローチャートを示します。

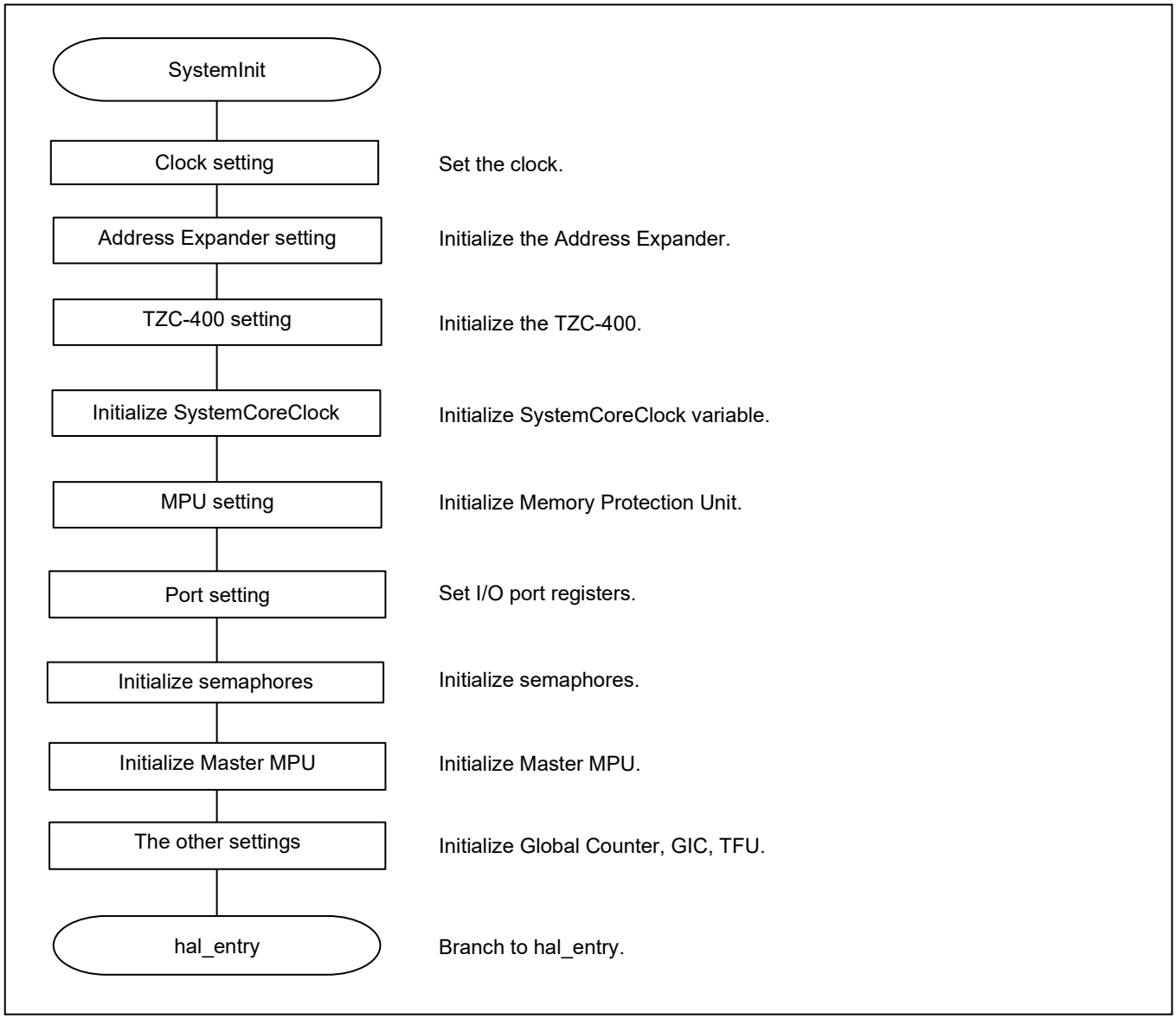

図 3-10 SystemInit 処理(アプリケーションプログラム)

3.5.2.5 hal_entry 処理

図 3-11 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。

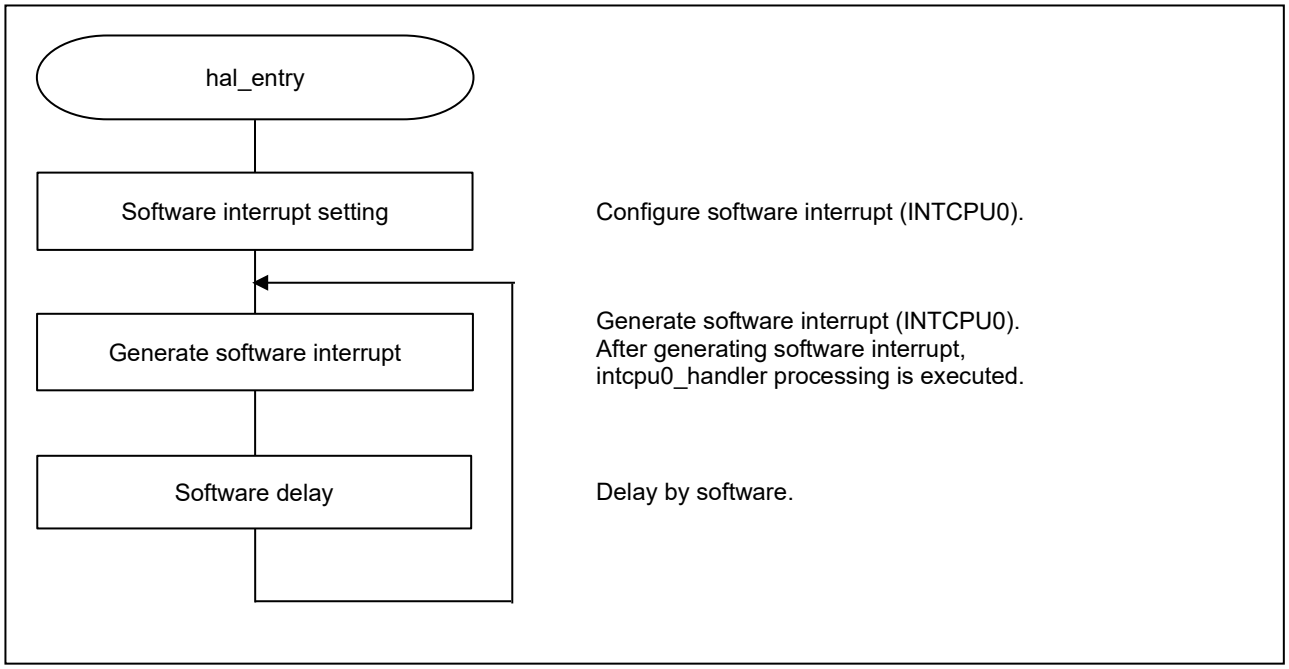

図 3-11 hal_entry 処理(アプリケーションプログラム)

図 3-12 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。

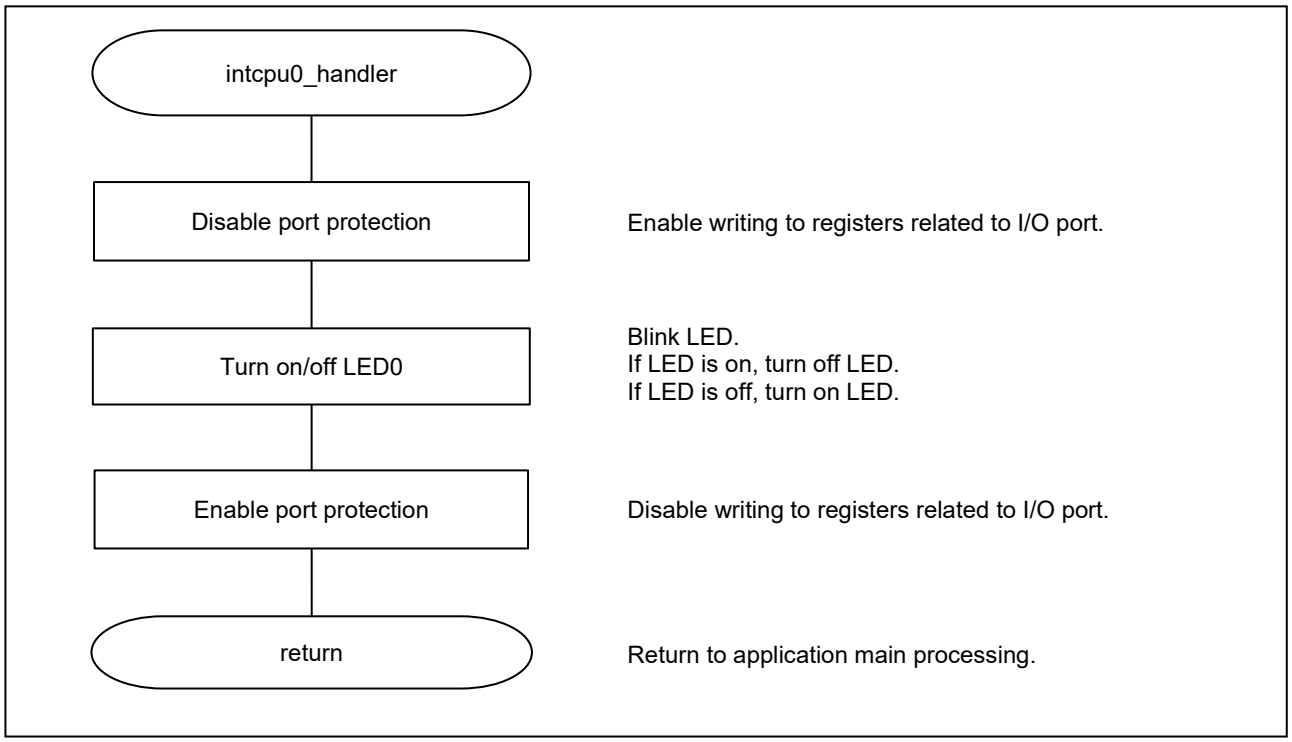

図 3-12 割り込み処理(アプリケーションプログラム)

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/T2H および RZ/N2H 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。)

  RZ/T2H Evaluation Board 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¥RZT2H_bsp_xspi0bootx1_separating_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/T2H_bsp_xspi0bootx1_app Debug のプロジェクトをワークスペースから選択します。

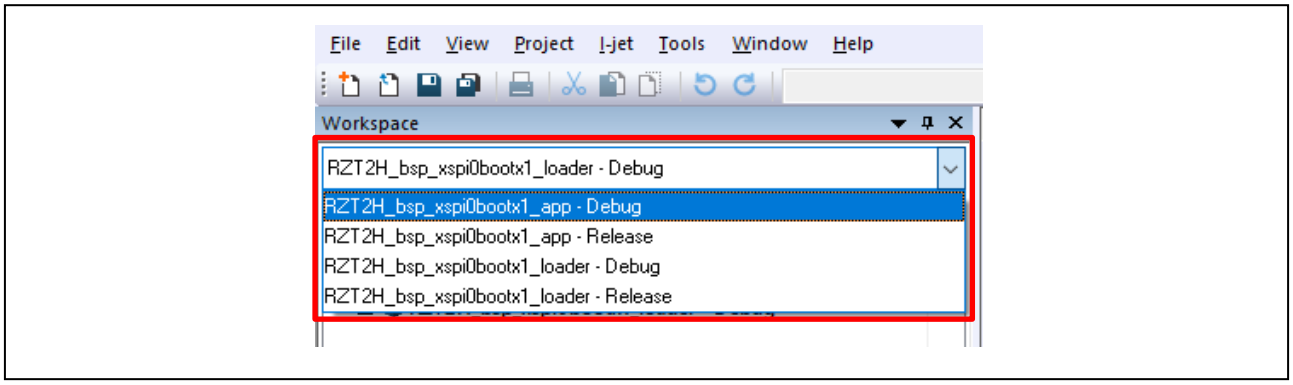

図 5-1 プロジェクトの選択

3. [ツール]から FSP Configurator を開き、[Generate Project Content]ボタンを押してコード生成を実行します。コード生成が完了したら FSP Configurator を閉じます。

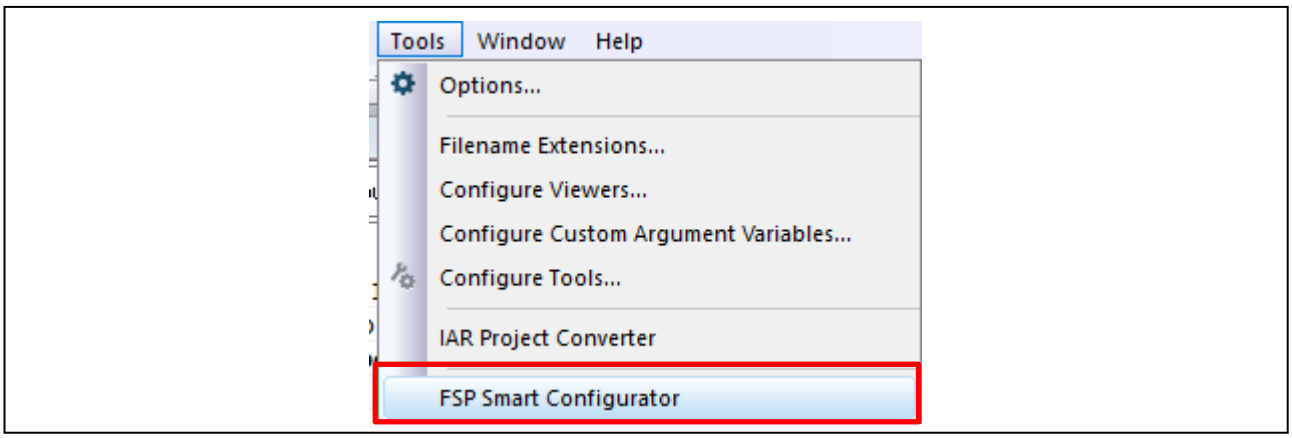

図 5-2 FSP Configurator の起動

※ [Tools]→[Configure Tools…]を開き、使用する FSP を事前に登録しておく必要があります。

表 5-1 FSP Smart Configurator の登録例

設定項目	設定例
Menu Text	FSP Smart Configurator
Command	C:¥Renesas¥rzt¥sc_v2025-4.1_fsp_v3.0.0¥eclipse¥rasc.exe
Argument	--compiler IAR configuration.xml
Initial Directory	$PROJ_DIR$

4. [プロジェクト]→[オプション]を開き、[リンカ]カテゴリを開きます。使用するリンカスクリプトを『 **$PROJ_DIR$/script/fsp_xspi0_boot_app.icf** 』に設定します。

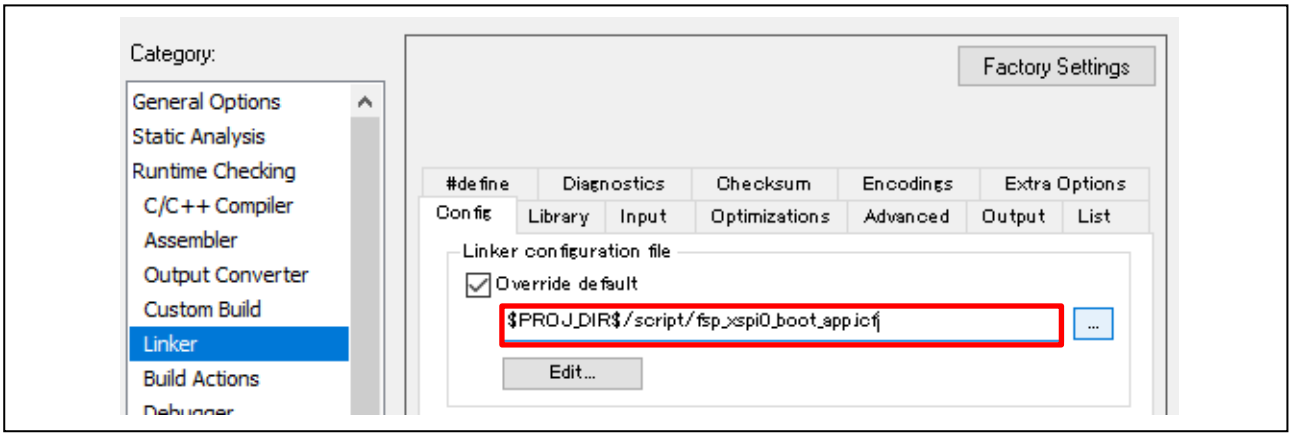

図 5-3　リンカスクリプトの設定（アプリケーションプログラム）

5. [プロジェクト]→[全てを再ビルド]を押し、アプリケーションプログラムのビルドを実行します。
6. RZ/T2H_bsp_xspi0bootx1_loader Debug プロジェクトをワークスペースから選択して開きます。
7. [ツール]から FSP Configurator を開き、[Generate Project Content]ボタンを押してコード生成を実行します。コード生成が完了したら FSP Configurator を閉じます。
8. [プロジェクト]→[オプション]を開き、[リンカ]カテゴリを開きます。使用するリンカスクリプトを『 **$PROJ_DIR$/script/fsp_xspi0_boot_loader.icf** 』に設定します。

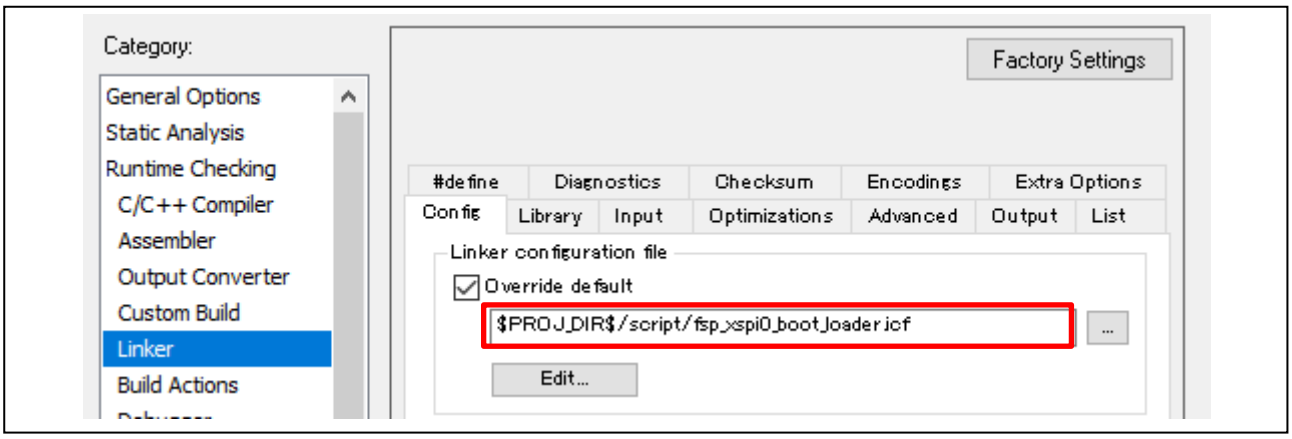

図 5-4　リンカスクリプトの設定（ローダプログラム）

9. [プロジェクト]→[全てを再ビルド]を押し、ローダプログラムのビルドを実行します。
10. RZ/T2H_bsp_xspi0bootx1_loader Debug プロジェクトをワークスペースから選択し、RZ/T2H 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。
11. 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。

EWARM 環境における補足事項①

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。

Path:

$PROJ_DIR$¥..¥RZT2H_bsp_xspi0bootx1_app¥Debug¥Exe¥RZT2H_bsp_xspi0bootx1_app.out

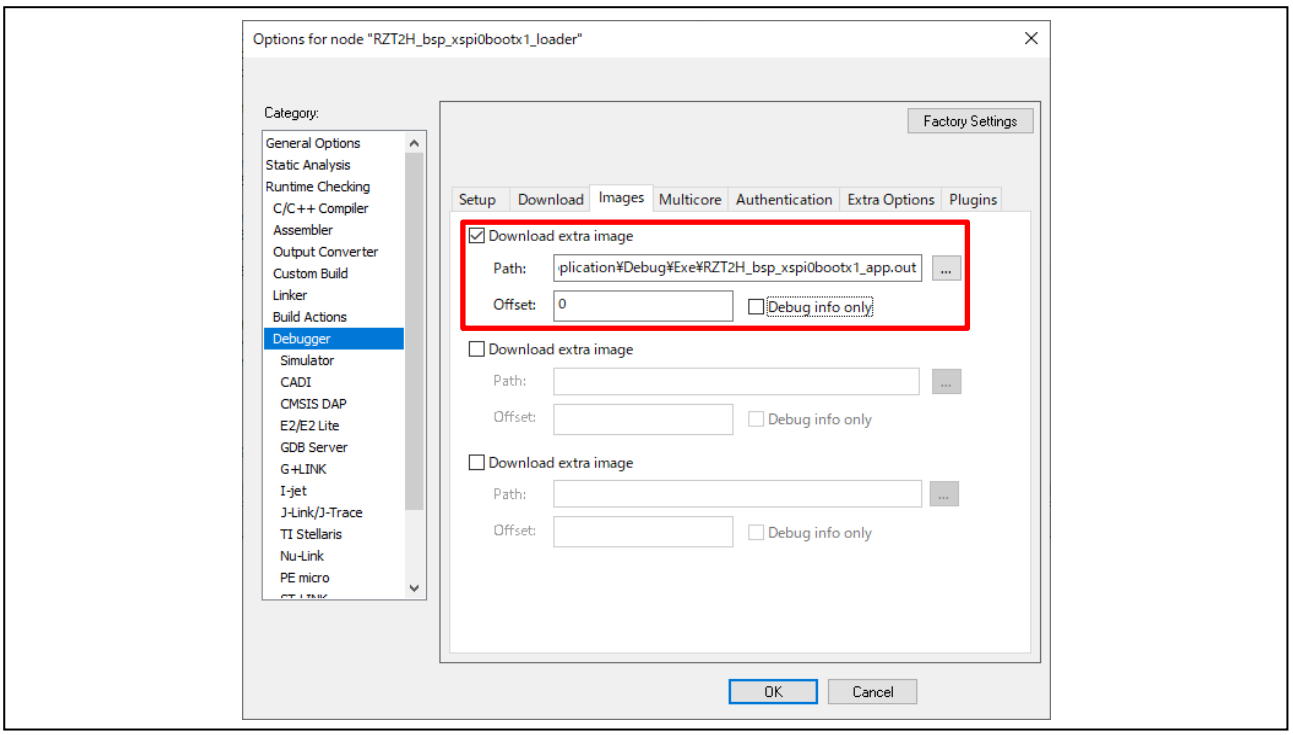

図 5-5　EWARM オプション設定①

EWARM 環境における補足事項②

ローダプログラムのビルド時、下記のオプション設定によりアプリケーションプログラムと一緒にローダプログラムがビルドされます。

Keep symbols:
  APPLICATION_PRG_SECTION
File:
  $PROJ_DIR$¥..¥RZT2H_bsp_xspi0bootx1_app¥Debug¥Exe¥RZT2H_bsp_xspi0bootx1_app.bin
Symbol:
  APPLICATION_PRG_SECTION
Section:
  APPLICATION_PRG_SECTION
Align:
  4

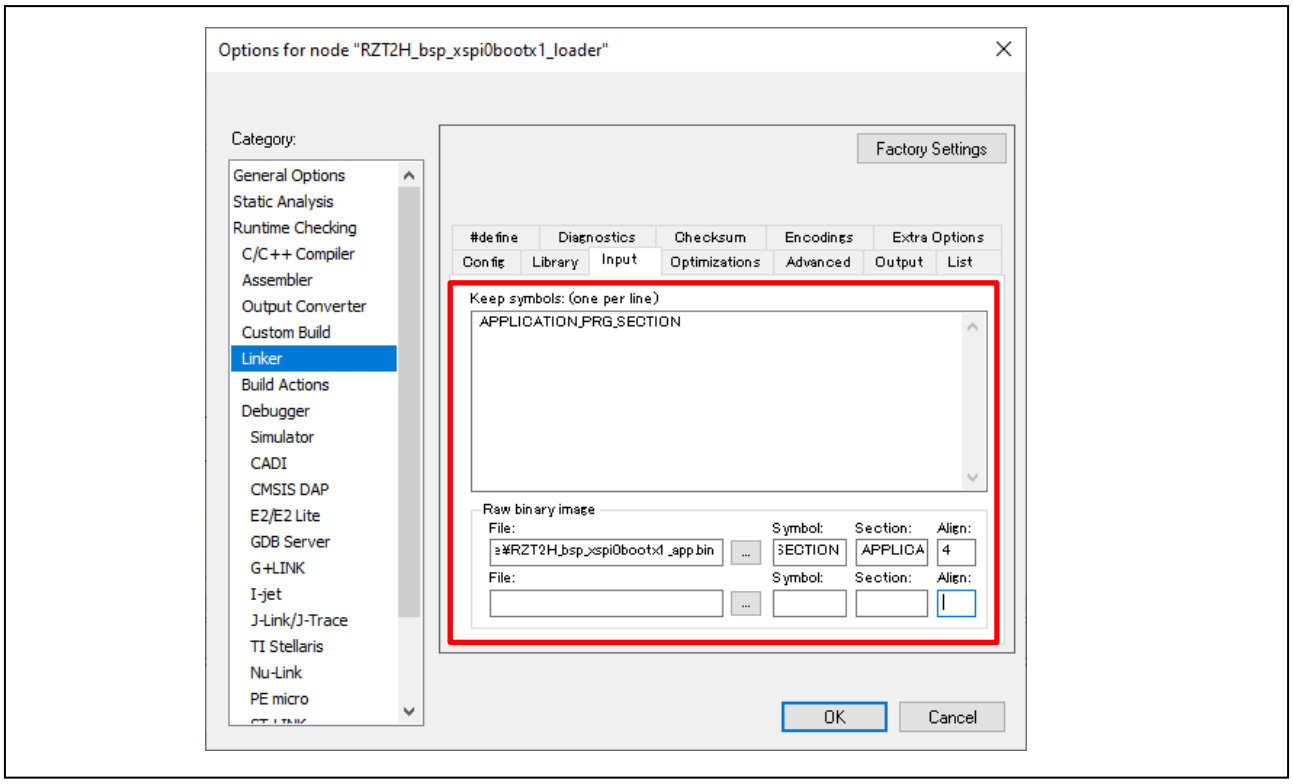

図 5-6　EWARM オプション設定②

EWARM 環境における補足事項③

以下のオプション設定により、アプリケーションプロジェクトのビルド時にローバイナリでビルド成果物が出力されます。

1. [Project]→[Options]を開きます。
2. [Output Converter]を開き、[Output format]を"Raw binary"に変更します。

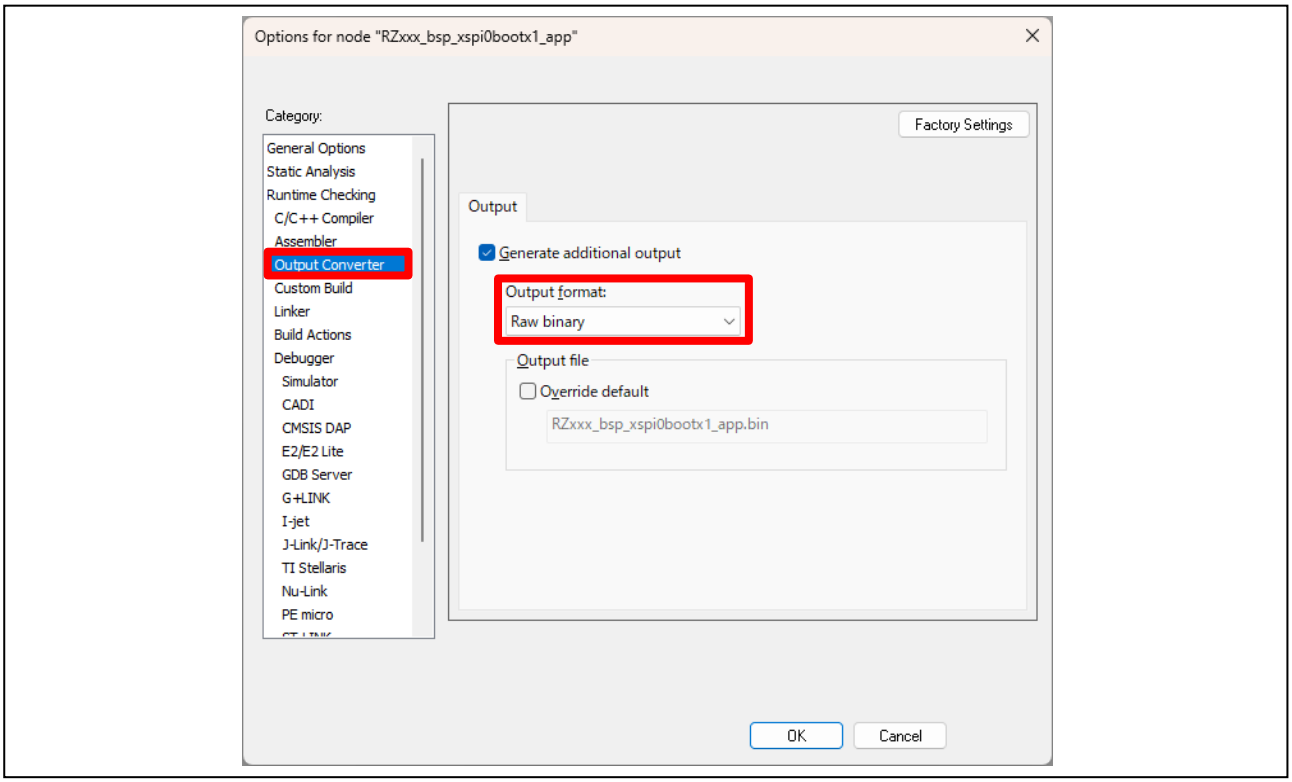

図 5-7 アプリケーションプロジェクトの出力フォーマットの設定

EWARM 環境における補足事項④

以下のオプション設定により、ローダプロジェクトとアプリケーションプロジェクトのコンパイル前に設定されているコード生成用アクションを削除しています。
このアクションはコンパイル前にコード生成を行う機能であり、コード生成の実行によりリンカスクリプトファイルのパスがデフォルト設定に上書きされてしまうため、本設定によりこれを防止しています。

1. [Project]→[Options]を開きます。
2. [Build Actions]を開き、[Build order]が"Pre-compile"のものを選択して、[Remove]をクリックします。

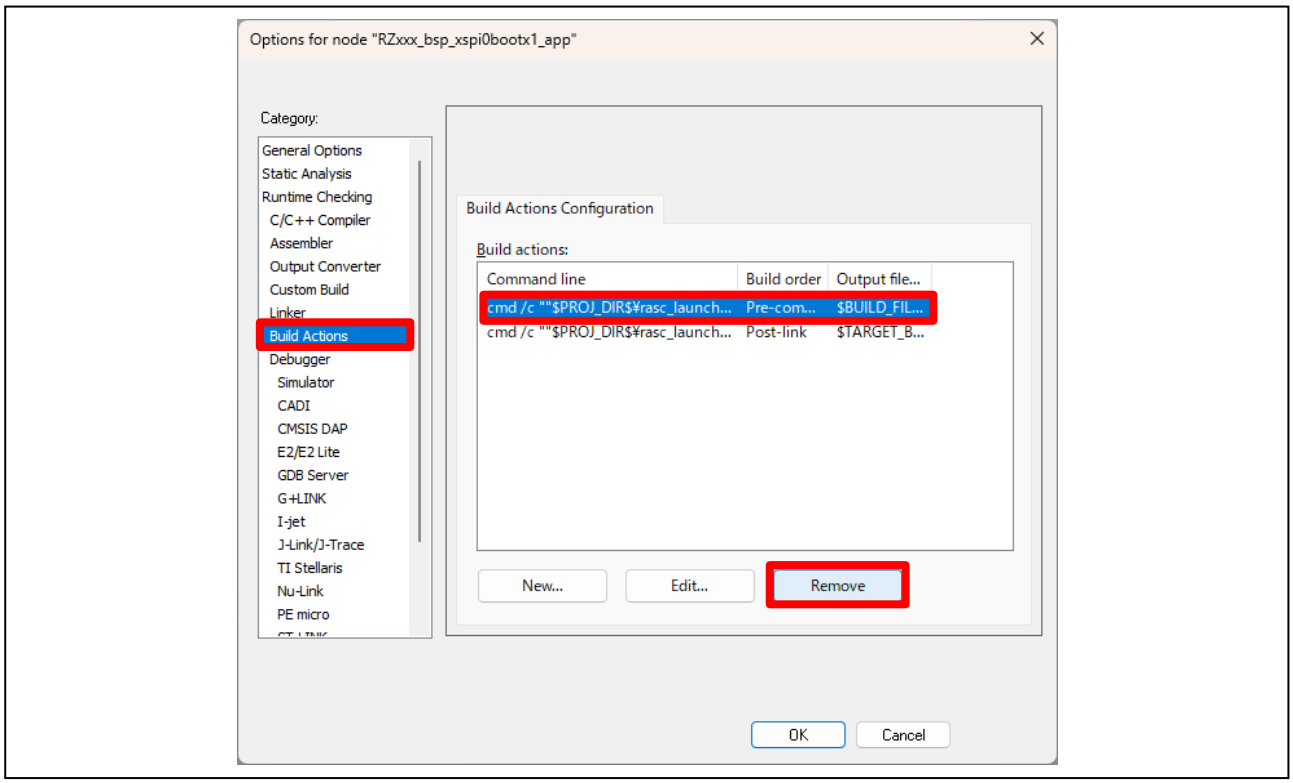

図 5-8 ビルドアクションの消去

EWARM 環境における補足事項⑤

ローダプロジェクトとアプリケーションプロジェクトのデバッグ設定を以下のように変更しています。

[ローダプロジェクト]

1. [Project]→[Options]を開きます。
2. [Debugger]→[Setup]を開き、"Run to"のチェックを外します。
3. [I-jet]→[Setup]を開き、"From the probe"のチェックを外します。

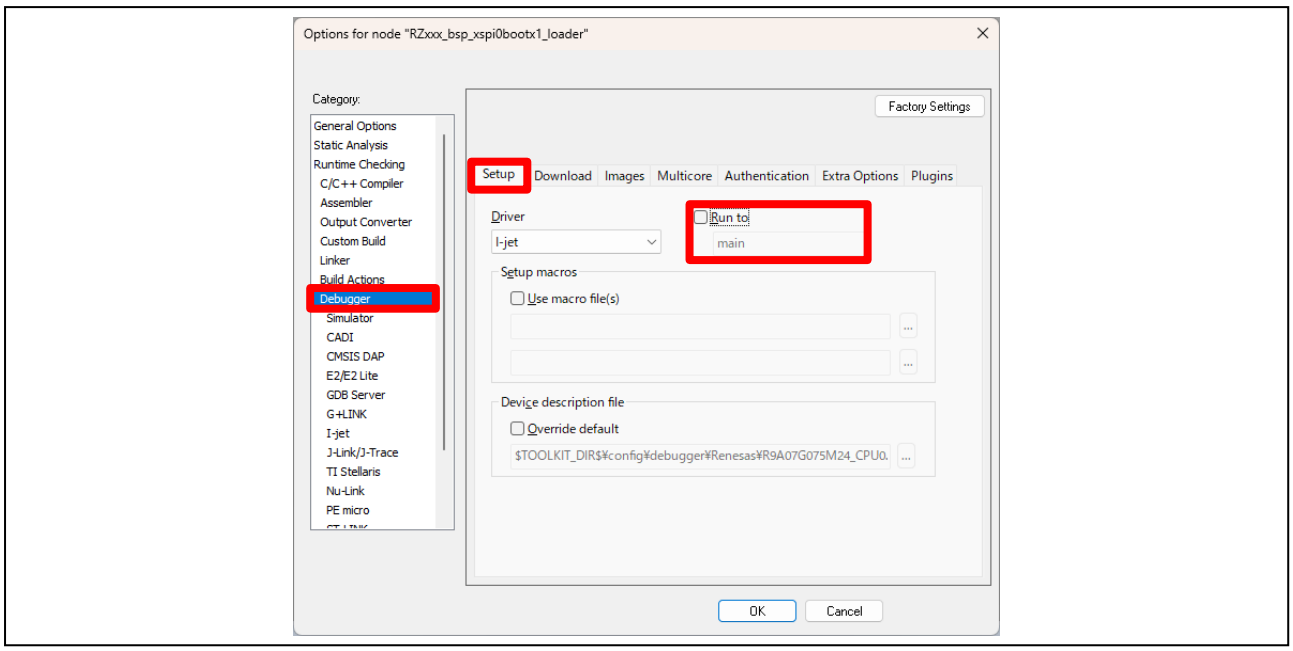

図 5-9 ローダプロジェクトのデバッグ設定①

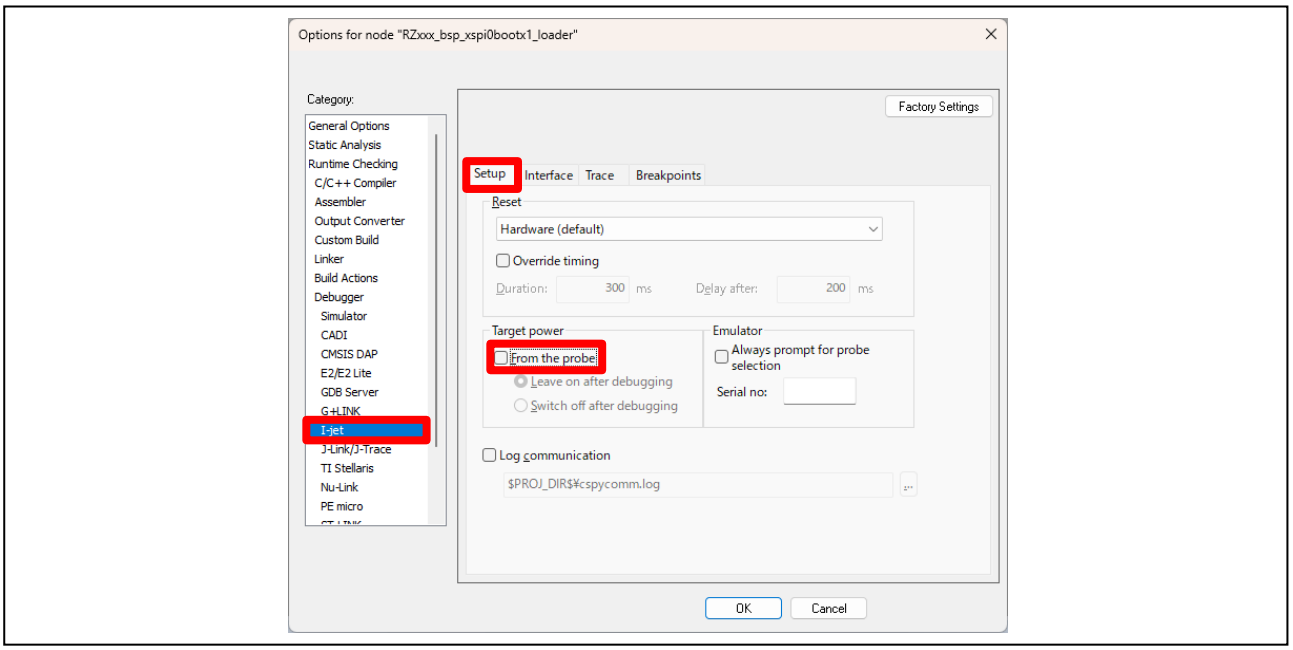

図 5-10 ローダプロジェクトのデバッグ設定②

[アプリケーションプロジェクト]

1. [Project]→[Options]を開きます。
2. [Debugger]→[Setup]を開き、"Run to"のチェックを外します。
3. [I-jet]→[Setup]を開き、"From the probe"のチェックを外し、[Reset]を"Software"に変更します。

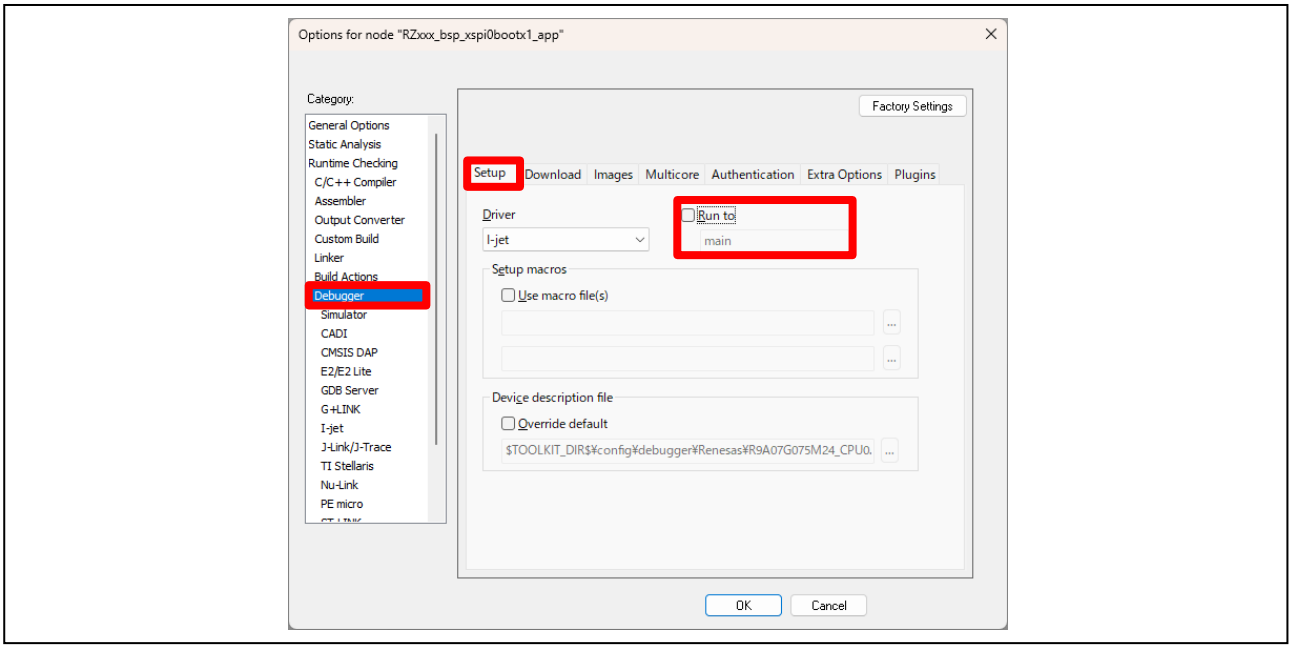

図 5-11 アプリケーションプロジェクトのデバッグ設定①

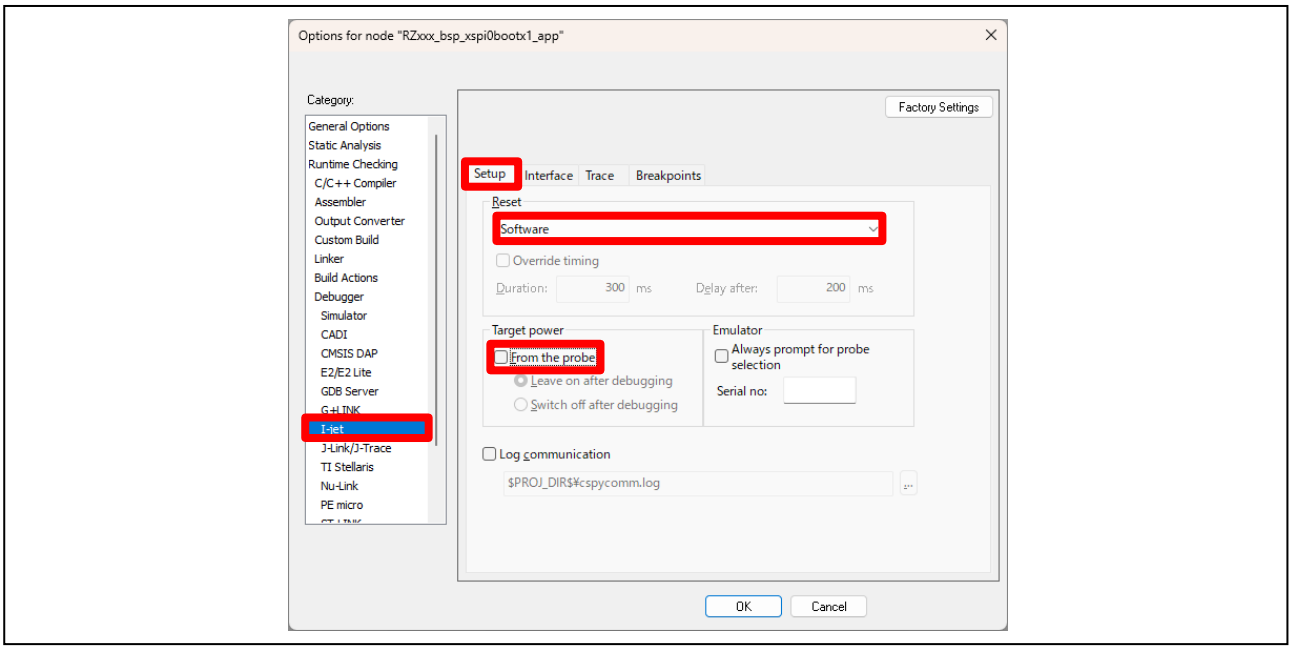

図 5-12 アプリケーションプロジェクトのデバッグ設定②

EWARM 環境における補足事項⑥

デバッグ開始時、次のスタック plug-in に関する警告が表示される場合があります。

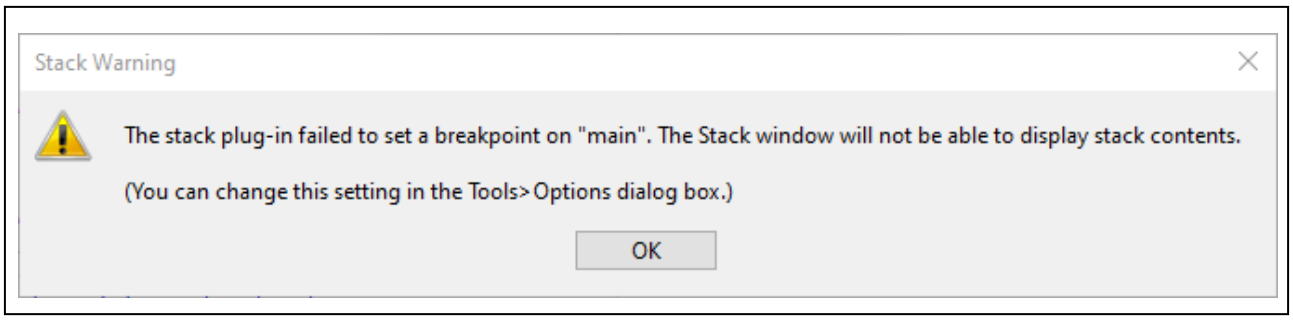

図 5-13　スタック plug-in に関する警告

[ツール]→[オプション]→[Stack]を選択し、[Stack pointer(s) not valid until program reaches]のチェックマークを外すことで、上記の警告ウィンドウは表示されなくなります。

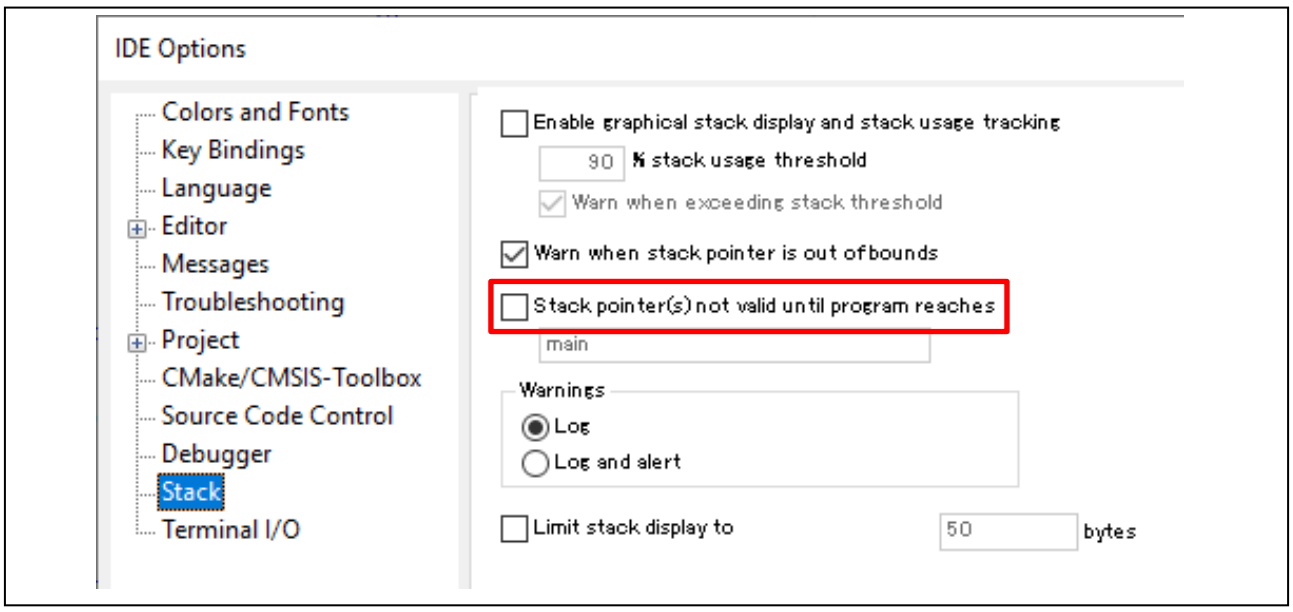

図 5-14　IDE Options の設定

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/T2H_bsp_xspi0bootx1_app”プロジェクトの configuration.xml を開き、[Generate Project Content]ボタンを押してコード生成を実行します。
4. “RZ/T2H_bsp_xspi0bootx1_app”プロジェクトのクリーンを実行した後に、ビルドを実行します。
5. “RZ/T2H_bsp_xspi0bootx1_loader”プロジェクトの configuration.xml を開き、[Generate Project Content]ボタンを押してコード生成を実行します。
6. “RZ/T2H_bsp_xspi0bootx1_loader”プロジェクトのクリーンを実行した後に、ビルドを実行します。
7. “RZ/T2H_bsp_xspi0bootx1_loader”プロジェクトのデバッグ接続設定を選択し、RZ/T2H 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。
8. 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①

以下のオプション設定により、ローダプロジェクトとアプリケーションプロジェクトのリンカスクリプトファイルのパスを変更しています。

1. プロジェクトのノードを右クリックし、[Properties]を開きます。
2. [C/C++ Build]→[Settings]→[Tool Settings]→[Cross ARM C Linker]→[General]を開き、ファイルパスを変更します。
   [ローダプロジェクト] : ”fsp_xspi0_boot_loader.ld”
   [アプリケーションプロジェクト] : ”fsp_xspi0_boot_app.ld”

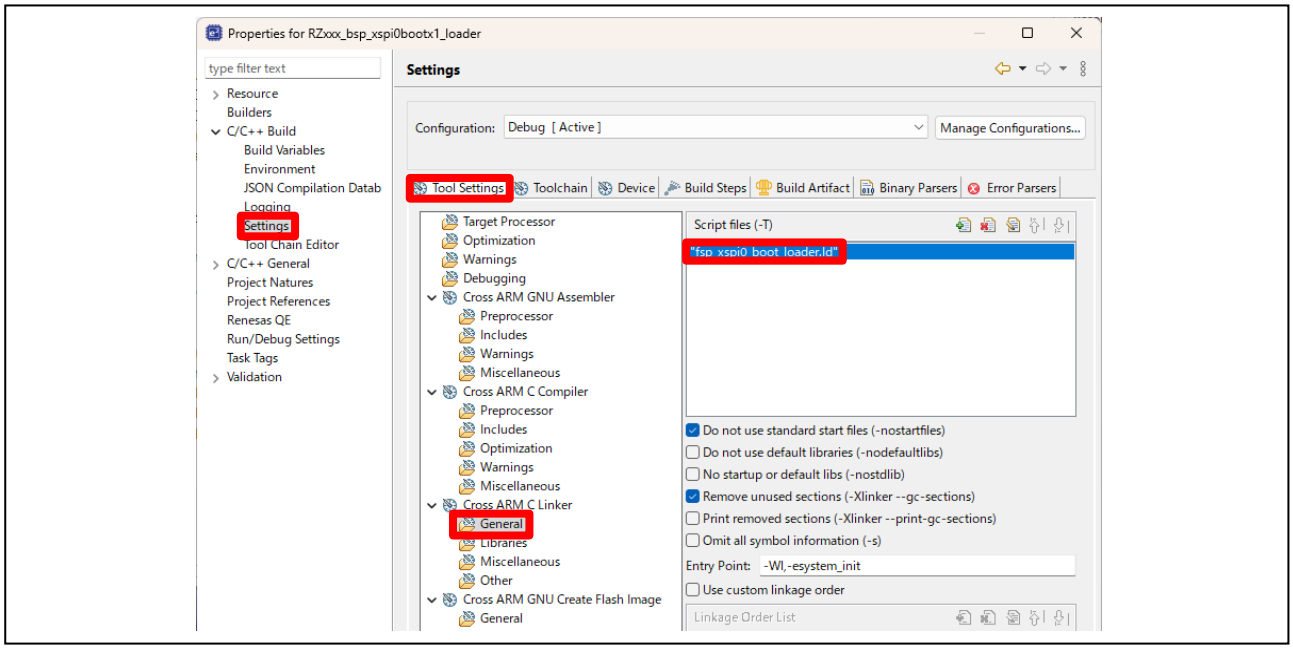

図 5-15 ローダプロジェクトのリンカスクリプトファイルパスの設定

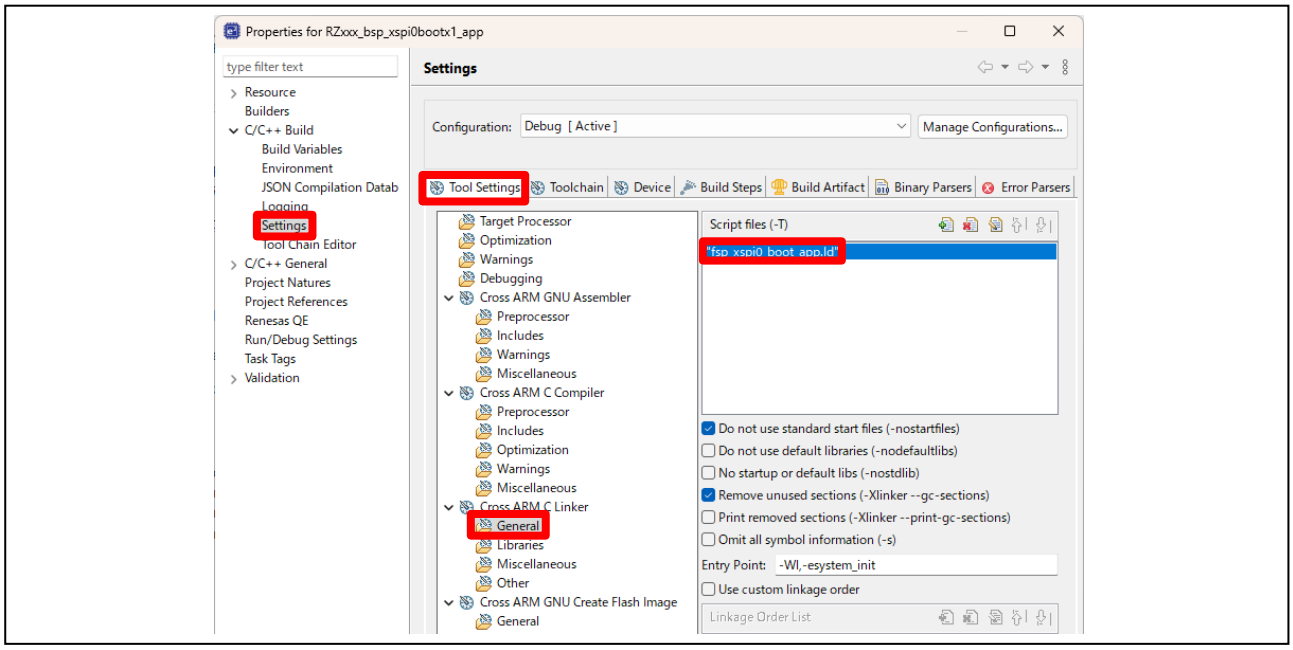

図 5-16 アプリケーションプロジェクトのリンカスクリプトファイルパスの設定

e² studio 環境における補足事項②

以下のオプション設定により、アプリケーションプロジェクトのビルド時にローバイナリでビルド成果物が出力されます。

1. アプリケーションプロジェクトのノードを右クリックし、[Properties]を開きます。
2. [C/C++ Build]→[Settings]→[Tool Settings]→[Cross ARM GNU Create Flash Image]→[General]を開き、[Output file format (-O)]を"Raw binary"に変更します。

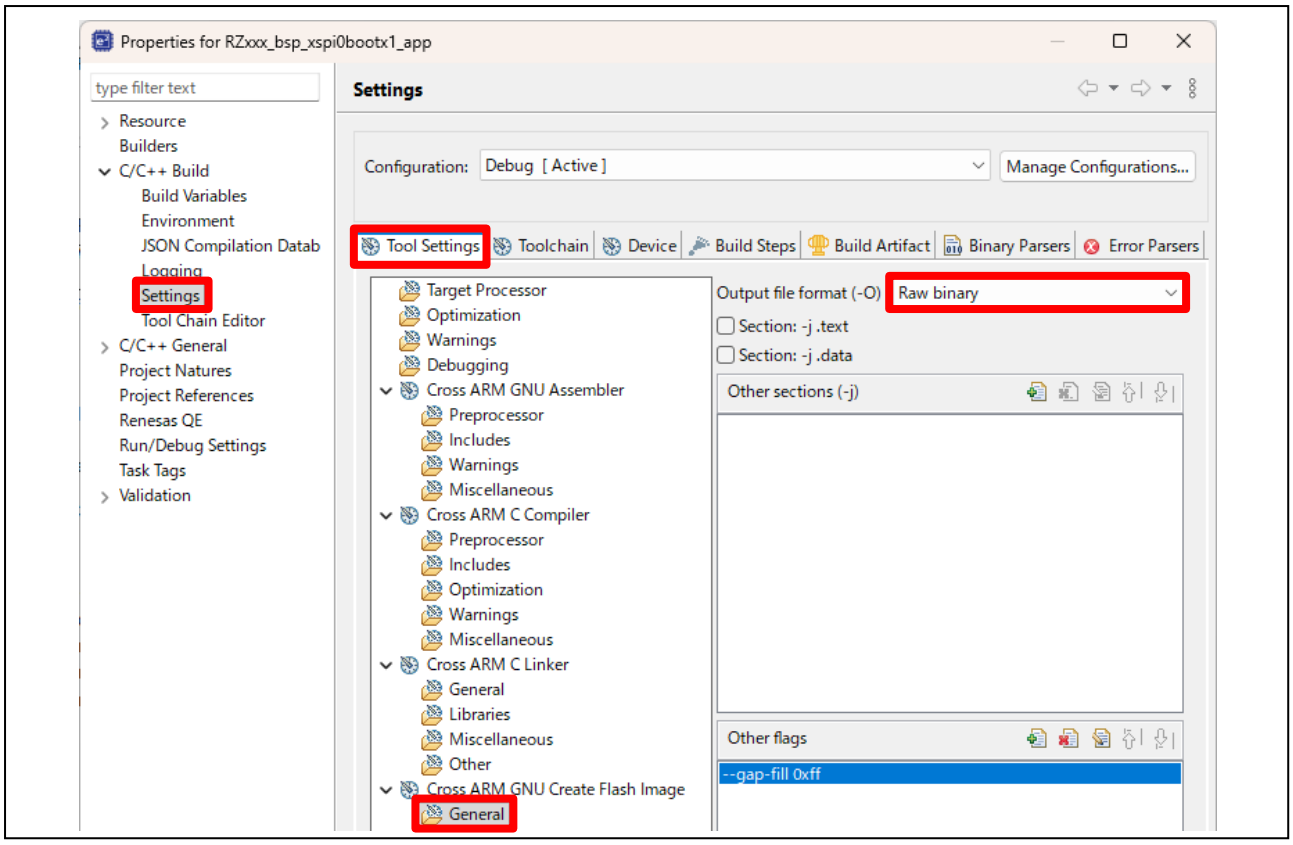

図 5-17 アプリケーションプロジェクトの出力フォーマットの設定

e² studio 環境における補足事項③

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。

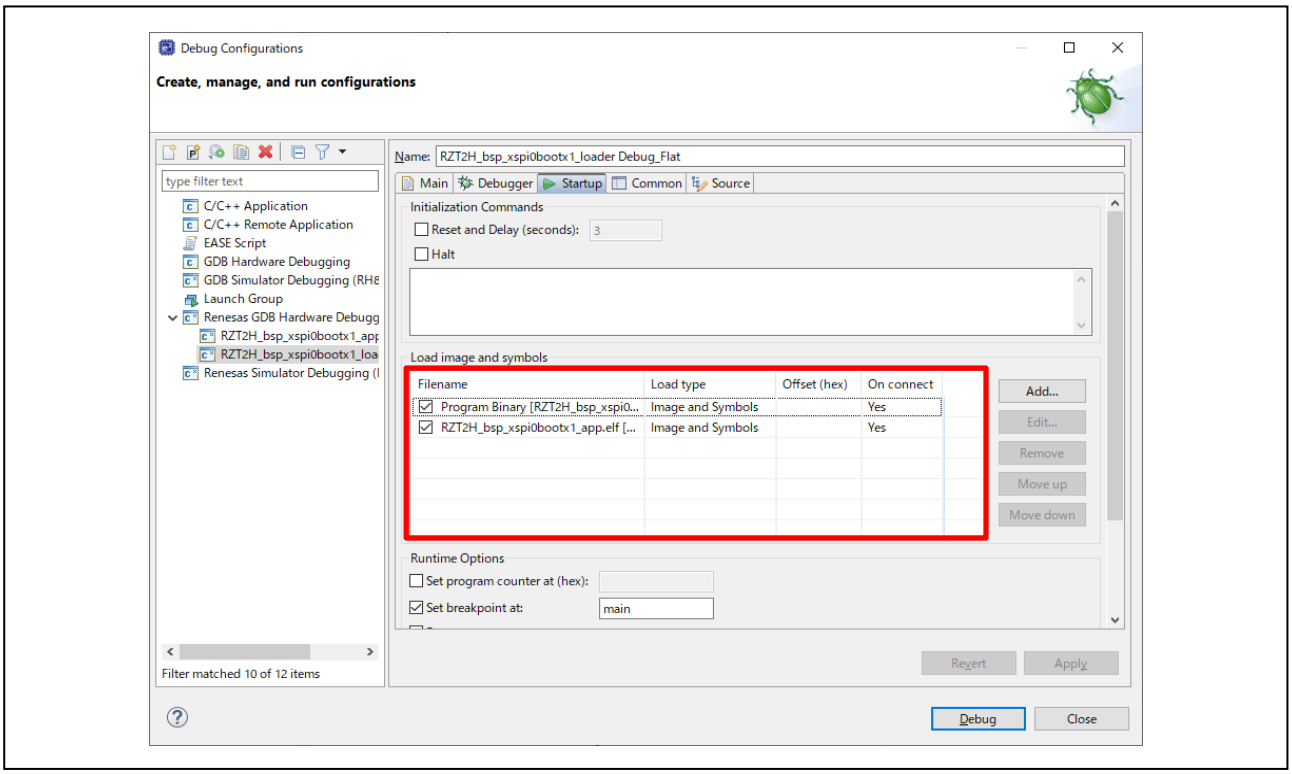

図 5-18　e²studio デバッグ接続設定

5.1.3 アプリケーションプログラムとローダプログラムの補足事項

補足事項①

以下の FSP 設定により、アプリケーションプロジェクトの C ランタイムを無効化しています。
アプリケーションプロジェクトの C ランタイム初期化はローダプロジェクトで行われています。

1. アプリケーションプロジェクトの configuration.xml を開きます。
2. [BSP]→[C Runtime Initialization]を"Disabled"に変更します。

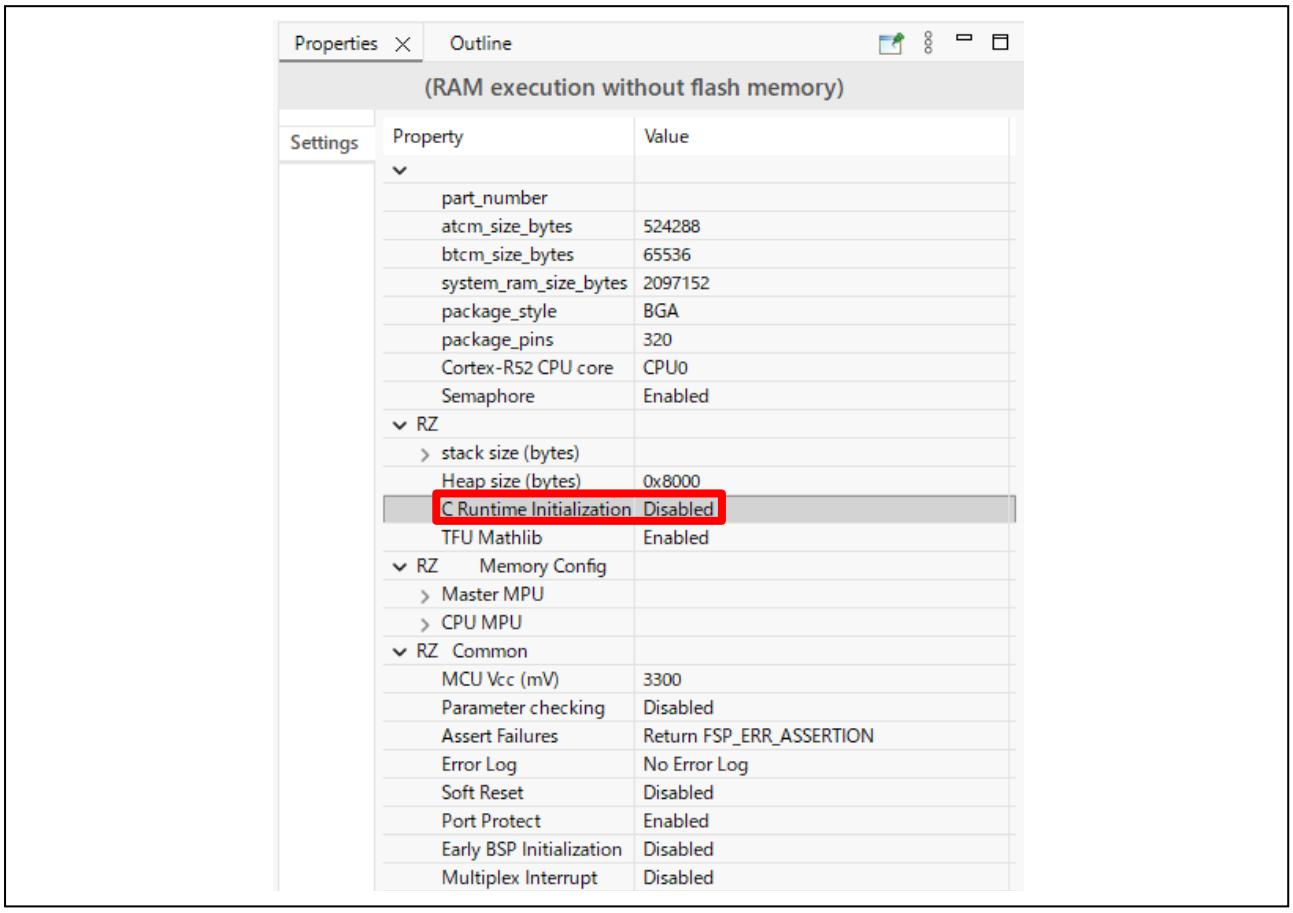

図 5-19 アプリケーションプログラムの BSP 設定

補足事項②

以下の FSP 設定により、アプリケーションプロジェクトの "INTCPU0" 割り込みを設定しています。

1. アプリケーションプロジェクトの configuration.xml を開きます。
2. [Interrupts]→[New User Event]→[ICU]→[INTCPU0 (Software interrupt 0)]を選択し、"intcpu0_handler"を設定し、[OK]をクリックします。

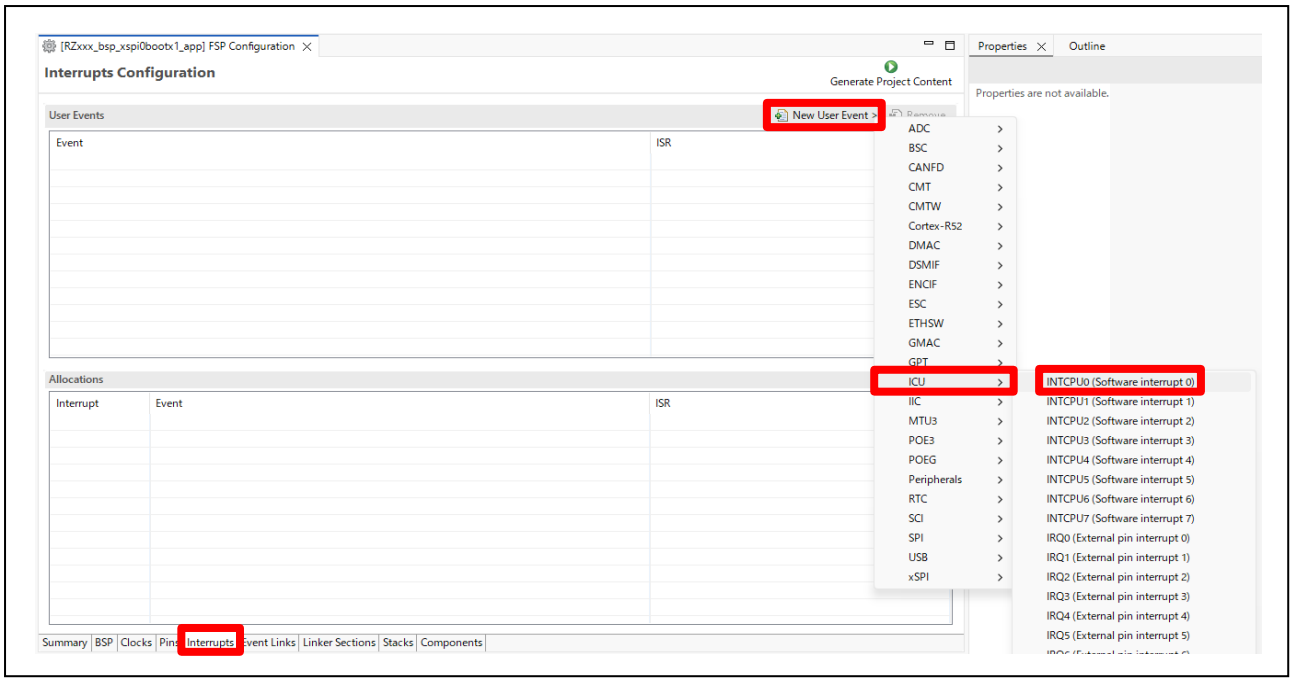

図 5-20 アプリケーションプログラムの割り込み設定①

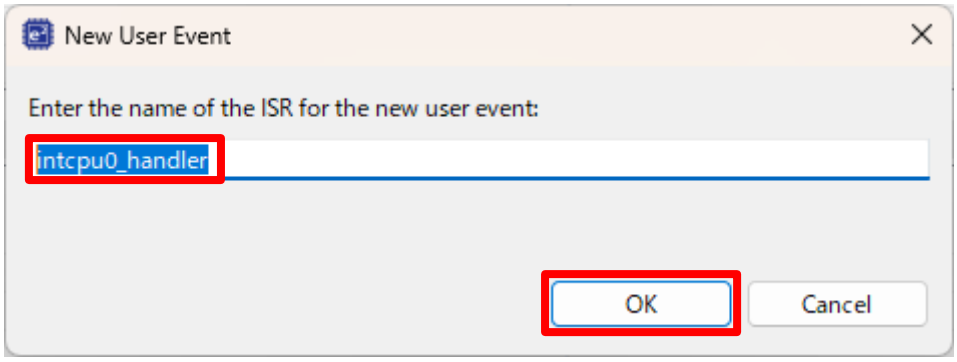

図 5-21 アプリケーションプログラムの割り込み設定②

補足事項③

アプリケーションプログラムの startup_core.c は、FSP コンフィグレータによって生成されたものから下記の修正を加えています。本サンプルパッケージには、修正を加えた startup_core.c がコード生成前から含まれています。FSP バージョンを変更した場合は、コード生成したものに上書きされてしまうため、同様の修正を加えて下さい。

```
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
{
#if 1 // Software loops are only needed when debugging.
    __asm volatile (
        "   mov   r0, #0                        ¥n"
        "   movw  r1, #0x68bf                    ¥n"
        "   movt  r1, #0x478                     ¥n"
        "software_loop:                          ¥n"
        "   adds  r0, #1                        ¥n"
        "   cmp   r0, r1                        ¥n"
        "   bne   software_loop                  ¥n"
        ::: "memory");
#endif

    __asm volatile (
        "set_hactlr:                             ¥n"
        "    MOVW  r0, %[bsp_hactlr_bit_l]       ¥n" /* Set HACTLR bits(L) */
        "    MOVT  r0, #0                        ¥n"
        "    MCR   p15, #4, r0, c1, c0, #1       ¥n" /* Write r0 to HACTLR */
        ::[bsp_hactlr_bit_l] "i" (BSP_HACTLR_BIT_L) : "memory");
```

図 5-22　startup_core.c の修正（ローダプログラム）

```
BSP_TARGET_ARM BSP_ATTRIBUTE_STACKLESS void system_init (void)
{
/* These settings are invalid for application project.
 * The necessary processing has been performed in the loader program. */
#if 0 // Original program
    __asm volatile (
        "set_hactlr:                             ¥n"
        "    MOVW  r0, %[bsp_hactlr_bit_l]       ¥n" /* Set HACTLR bits(L) */
        "    MOVT  r0, #0                        ¥n"
        "    MCR   p15, #4, r0, c1, c0, #1       ¥n" /* Write r0 to HACTLR */
        ::[bsp_hactlr_bit_l] "i" (BSP_HACTLR_BIT_L) : "memory");

~~~ omission ~~~

    __asm volatile (
        "exception_return:                       ¥n"
        "    LDR   r1, =stack_init               ¥n"
        "    MSR   ELR_hyp, r1                   ¥n"
        "    ERET                               ¥n" /* Branch to stack_init and enter EL1 */
        ::: "memory");

#else
    /* Set exception vector offset for application program. */
    __asm volatile (
        "set_vbar:                               ¥n"
        "    LDR   r0, =__Vectors                ¥n"
        "    MCR   p15, #0, r0, c12, c0, #0      ¥n" /* Write r0 to VBAR */
        ::: "memory");

    __asm volatile (
        "jump_stack_init:                        ¥n"
        "    LDR   r0, =stack_init               ¥n"
        "    BLX   r0                           ¥n"
        ::: "memory");
#endif
}
```

図 5-23　startup_core.c の修正（アプリケーションプログラム）

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 128K];

/* Flash memory */
define region LOADER_TABLE_region  =  mem:[from 0x40080000 size 64K];
define region APPLICATION_ROM_region  =  mem:[from 0x40010000 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 128K]; /* Change copy destination
address to ATCM */

/* Flash memory */
define region LOADER_TABLE_region  =  mem:[from 0x40080000 size 64K];
define region APPLICATION_ROM_region  =  mem:[from 0x40010000 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_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 e² studio : 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))
} > SYSTEM_RAM
.IMAGE_APP_FLASH_section 0x40010000 : AT (0x40010000)
{
    IMAGE_APP_FLASH_section_start = .;
    KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
    IMAGE_APP_FLASH_section_end = .;
} > FLASH_CONTENTS
~~~

After changing
~~~
.IMAGE_APP_RAM 0x00000000 : AT (0x00000000) /* Change copy destination address to ATCM */
{
    IMAGE_APP_RAM_start = .;
    KEEP(*(APP_IMAGE_RAM))
} > SYSTEM_RAM
.IMAGE_APP_FLASH_section 0x40010000 : AT (0x40010000)
{
    IMAGE_APP_FLASH_section_start = .;
    KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
    IMAGE_APP_FLASH_section_end = .;
} > FLASH_CONTENTS
~~~
```

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)
{
   ~~~
    _text_end = .;
} > SYSTEM_RAM

   ~~~
    __extab_end = .;
} > SYSTEM_RAM

    ~~~
    __exidx_end = .;
} > SYSTEM_RAM

    ~~~
    _data_end = .;
} > SYSTEM_RAM

    ~~~
    _end = .;
} > SYSTEM_RAM

    ~~~
    __HeapLimit = .;
} > SYSTEM_RAM

    ~~~
    __AArch64StackLimit = .;
} > SYSTEM_RAM
```

```
    ~~~
    __ThreadStackLimit = .;
} > SYSTEM_RAM

    ~~~
    __AbtStackLimit = .;
} > SYSTEM_RAM

~~~
```

After Changing

```
~~~
.text 0x00000000 : AT (TEXT_IMAGE) /* Change code area to ATCM */
{
    ~~~
    _text_end = .;
} > ATCM

    ~~~
    __extab_end = .;
} > ATCM

    ~~~
    __exidx_end = .;
} > ATCM

    ~~~
    _data_end = .;
} > ATCM

    ~~~
    _end = .;
} > ATCM

    ~~~
    __HeapLimit = .;
} > ATCM

    ~~~
    __AArch64StackLimit = .;
} > ATCM

    ~~~
    __ThreadStackLimit = .;
} > ATCM

    ~~~
    __AbtStackLimit = .;
} > ATCM

~~~
```

5.3 Cortex-A55 CPU0 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 Cortex-R52 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_CA55_CPU0」という定義を追加しており、その値を変更することで Cortex-A55 CPU0 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。

Cortex-A55 CPU0 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 Cortex-A55 CPU0 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 Cortex-A55 CPU0 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 Cortex-A55 CPU0 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、Cortex-A55 CPU0 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 Cortex-A55 CPU0 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. Cortex-A55 CPU0 プログラムのビルド

   Loader_application_projects¥RZT2H_bsp_xspi0bootx1_app_CA55_0¥RZT2H_bsp_xspi0bootx1_app_CA55_0.eww を開きます。
   [ツール]から FSP Configurator を開き、[Generate Project Content]ボタンを押してコード生成を実行します。コード生成が完了したら FSP Configurator を閉じます。
   Cortex-A55 CPU0 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。
   Cortex-A55 CPU0 プログラムをビルド後、RZT2H_bsp_xspi0bootx1_app_CA55_0.eww のワークスペースを閉じてください。

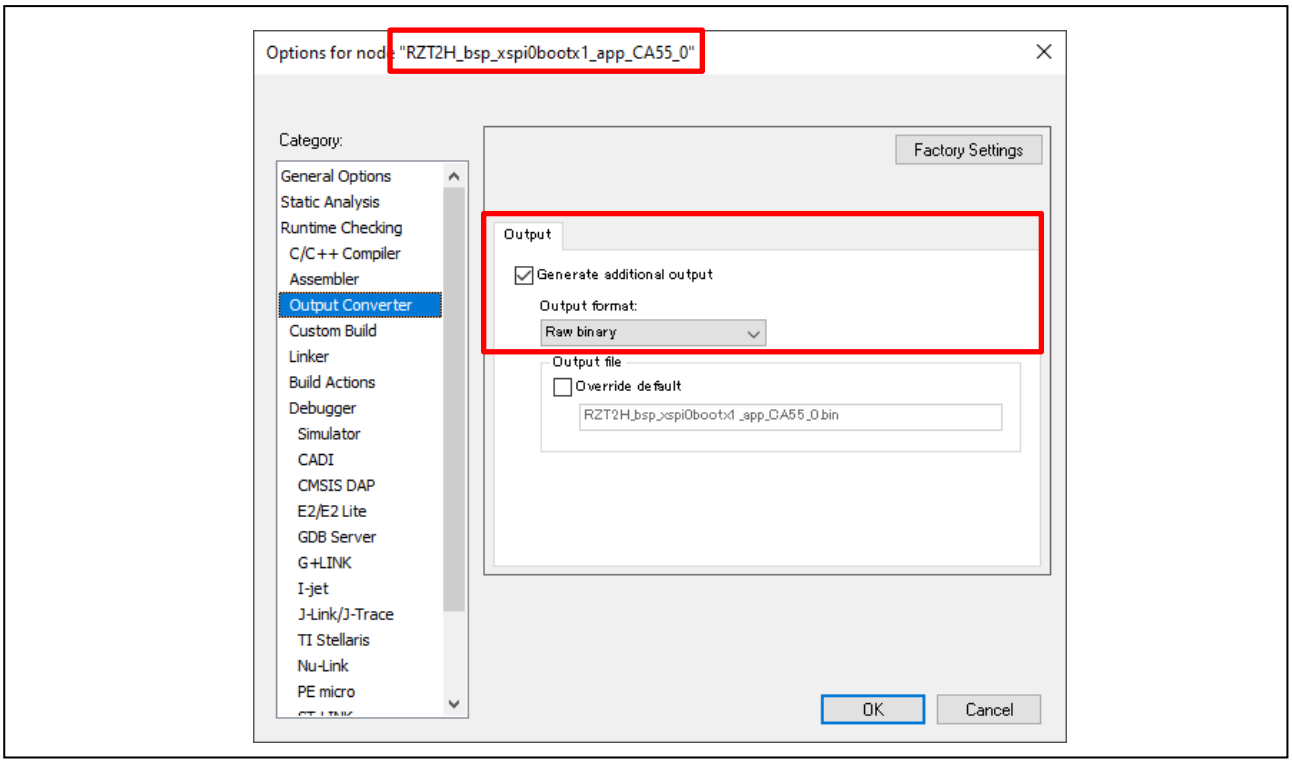

図 5-24 Cortex-A55 CPU0 プログラムのオプション設定

2. Cortex-A55 CPU0 プログラムを Cortex-R52 CPU0 ローダプログラムへリンク

Cortex-A55 CPU0 のバイナリを Cortex-R52 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 : CA55_CPU0_SECTION
ファイル :
$PROJ_DIR$¥..¥RZT2H_bsp_xspi0bootx1_app_CA55_0¥Debug¥Exe¥RZT2H_bsp_xspi0bootx1_app_CA55_0.bin
シンボル : CA55_CPU0_SECTION
セクション : CA55_CPU0_SECTION
アラインメント : 4

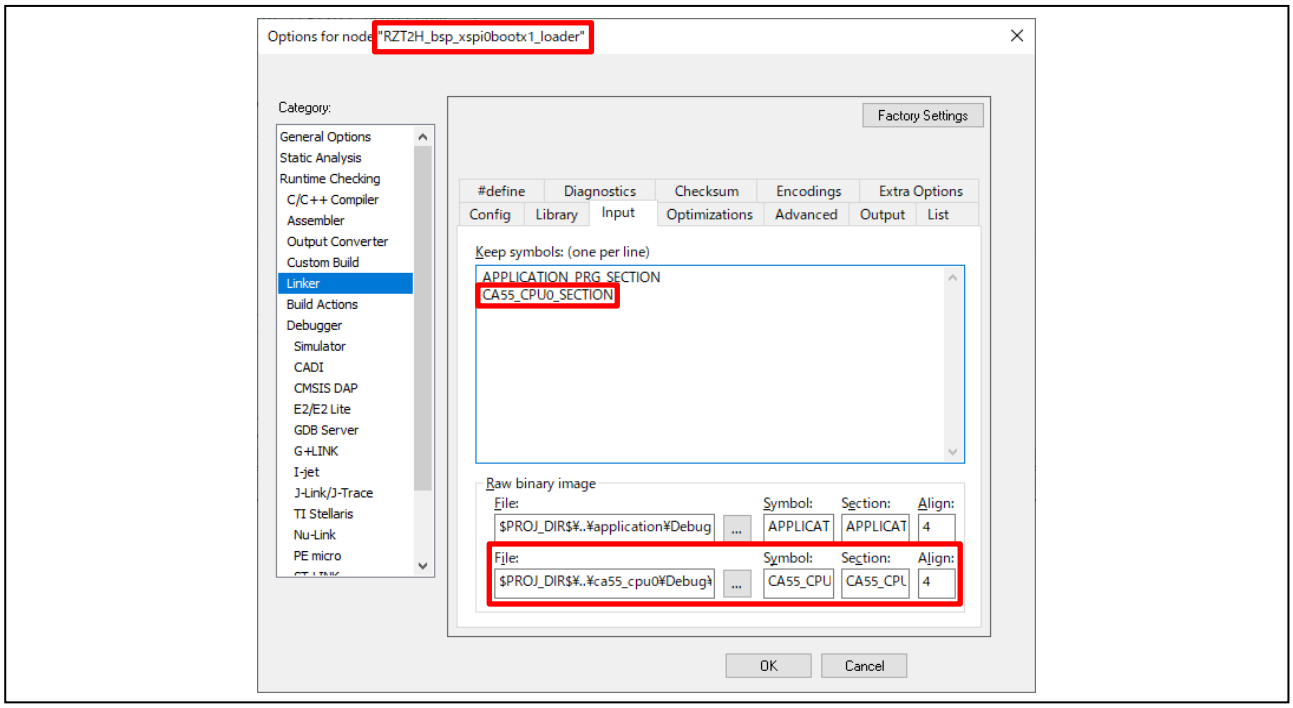

図 5-25 Cortex-R52 CPU0 のローダプログラムのオプション設定

3. USE_CA55_CPU0 定義の有効化

Cortex-A55 CPU0 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_CA55_CPU0」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_CA55_CPU0=1
[オプション] -> [リンカ] -> [設定]タブ: USE_CA55_CPU0=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_CA55_CPU0=1

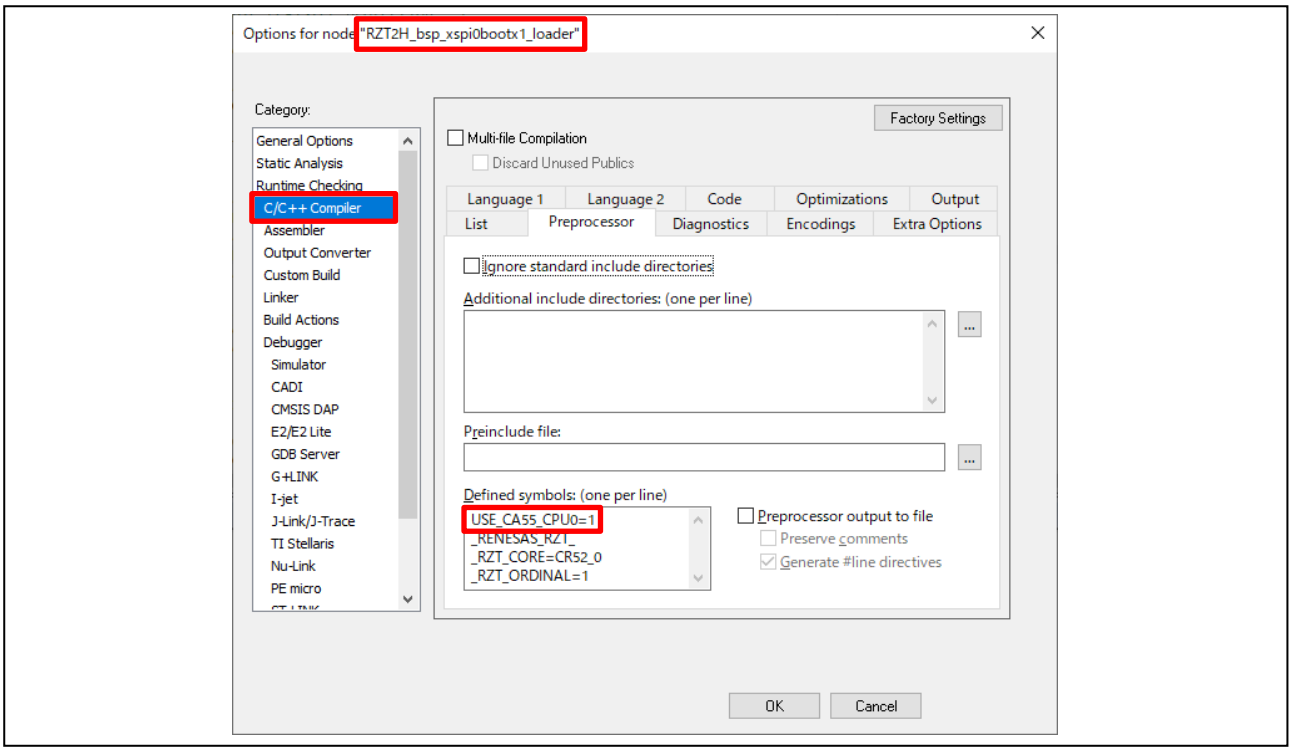

図 5-26 Cortex-R52 CPU0 のローダプログラムの USE_CA55_CPU0 定義有効化 (1/2)

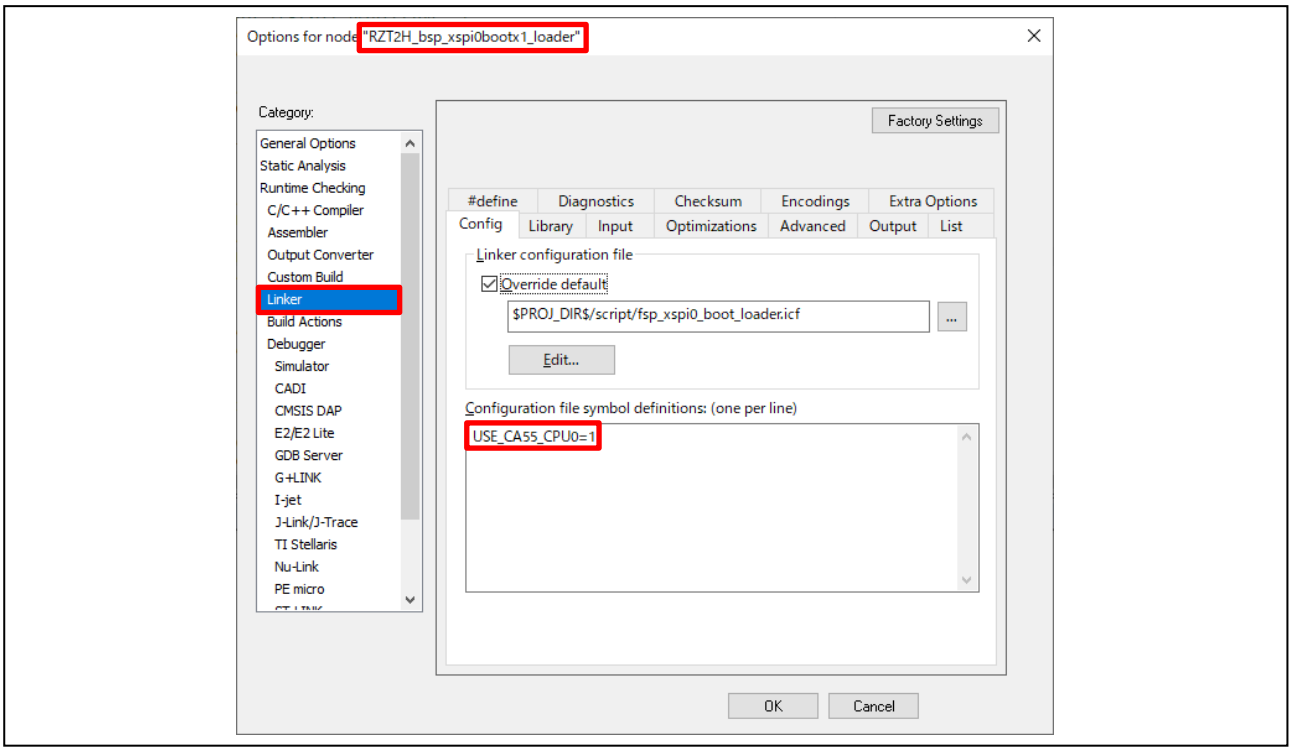

図 5-27　Cortex-R52 CPU0 のローダプログラムの USE_CA55_CPU0 定義有効化（2/2）

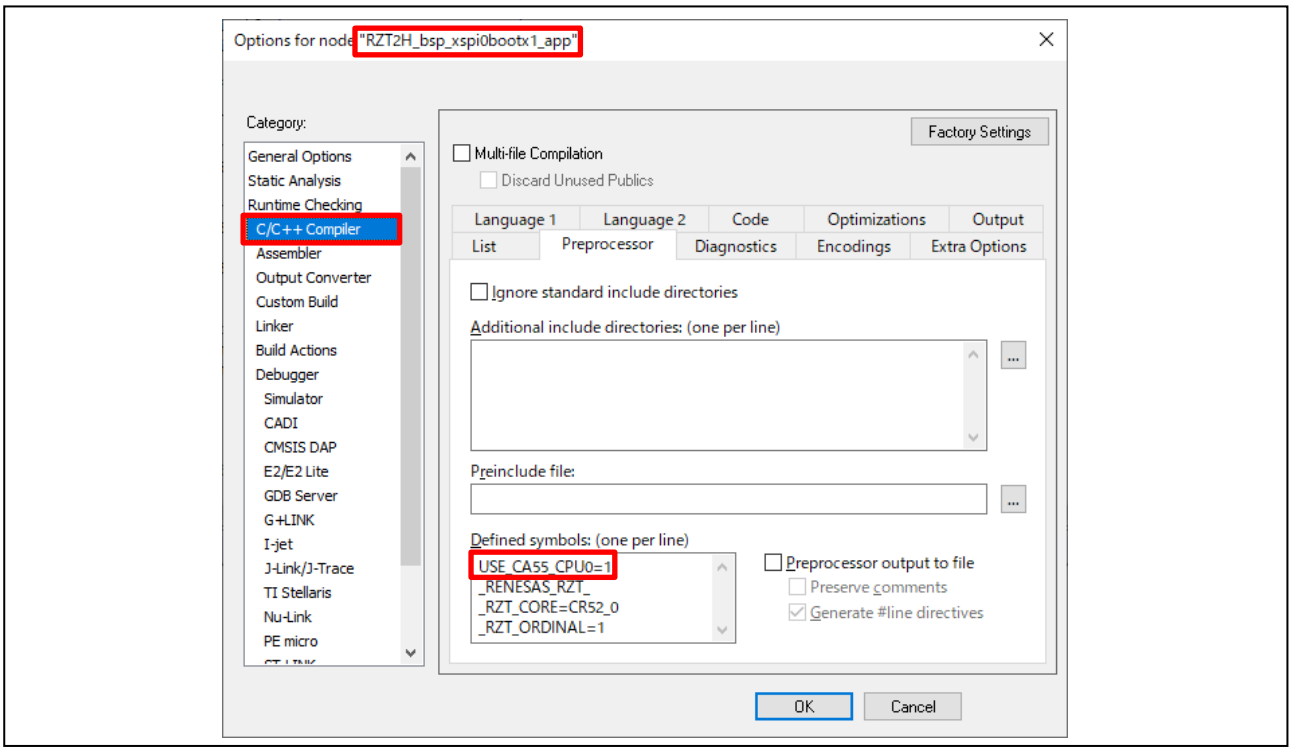

図 5-28　Cortex-R52 CPU0 のアプリケーションプログラムの USE_CA55_CPU0 定義有効化

4. Cortex-A55 CPU0 プロジェクトのデバッグを行う設定

Cortex-A55 CPU0 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、Cortex-A55 CPU0 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 :
$PROJ_DIR$¥..¥RZT2H_bsp_xspi0bootx1_app_CA55_0¥RZT2H_bsp_xspi0bootx1_app_CA55_0.eww
スレーブプロジェクト : RZT2H_bsp_xspi0bootx1_app_CA55_0
スレーブ構成 : Debug

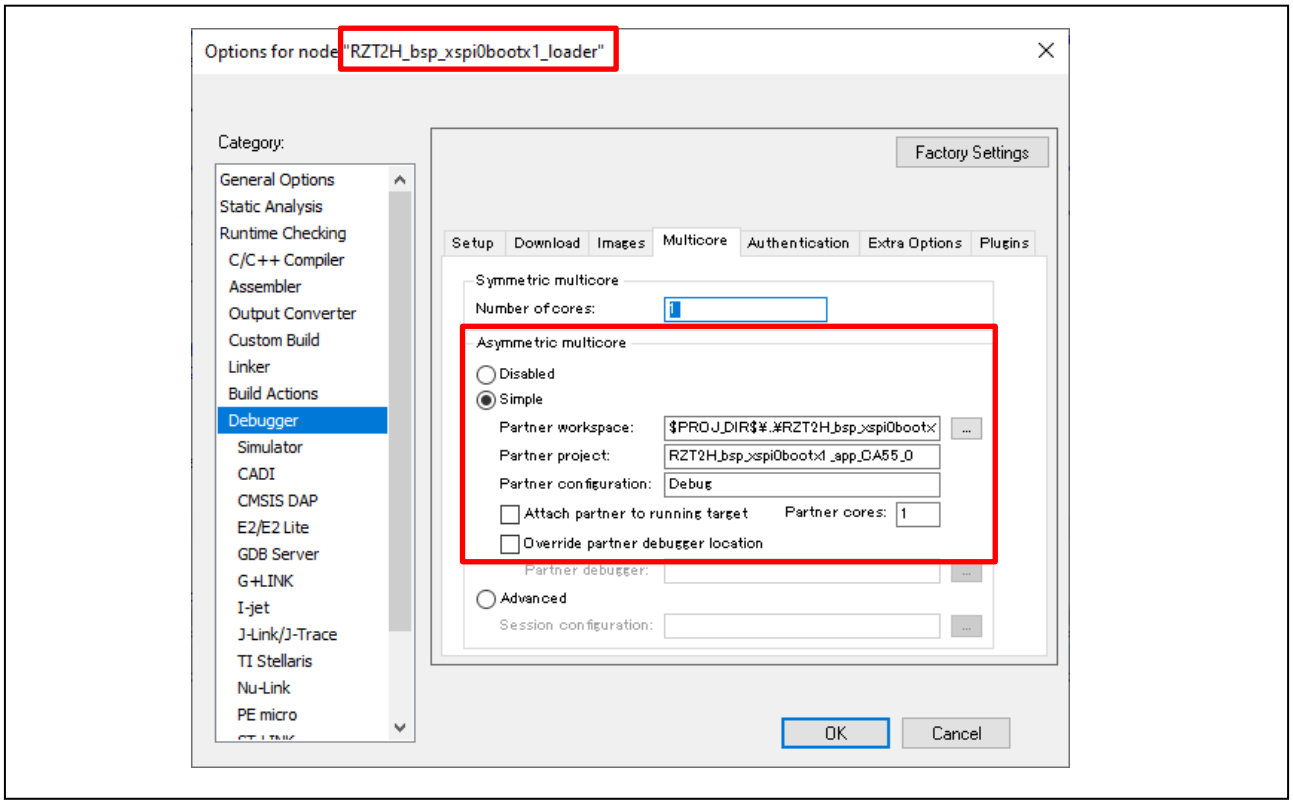

図 5-29 Cortex-R52 CPU0 のローダプログラムのマルチコアデバッグ設定

5. プロジェクトの再ビルドと実行

5.1.1 の手順に従い、ダウンロードとデバッグを開始してください。

デバッグ接続時、Cortex-A55 CPU0 のプロジェクトも自動的に立ち上がります。以降、Cortex-R52 CPU0 と Cortex-A55 CPU0 プロジェクトのデバッグが可能となります。
Cortex-A55 CPU0 プログラムが実行されると、LED2 が点滅します。

EWARM 環境における補足事項①

以下の設定により、Cortex-A55 CPU0 プロジェクトのオプションを変更しています。

1. [Project]→[Options]を開きます。
   [General Options]→[64-bit]を開き、"LP64 (32-bit int, 64-bit long, pointer)" を選択します。

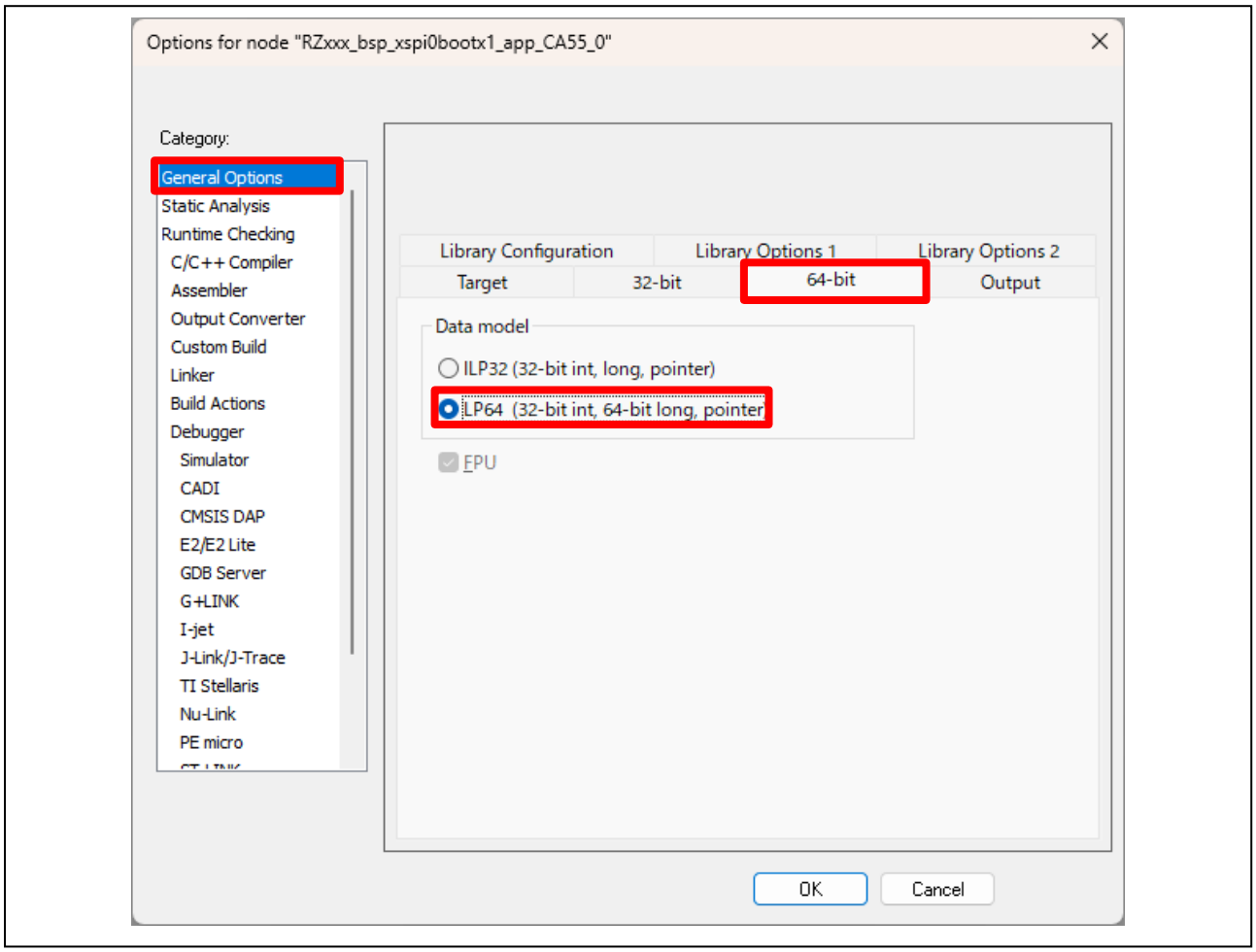

図 5-30 Cortex-A55 CPU0 プロジェクトのオプション設定

EWARM 環境における補足事項②

以下のオプション設定により、Cortex-A55 CPU0 プロジェクトのデバッガ追加オプションにコマンドラインを追加しています。

1. [Project]→[Options]を開きます。
2. [Debugger]→[Extra Options]を開き、[Command line options: (one per line)]に "--macro_param cpu1_enable=1" を追加します。

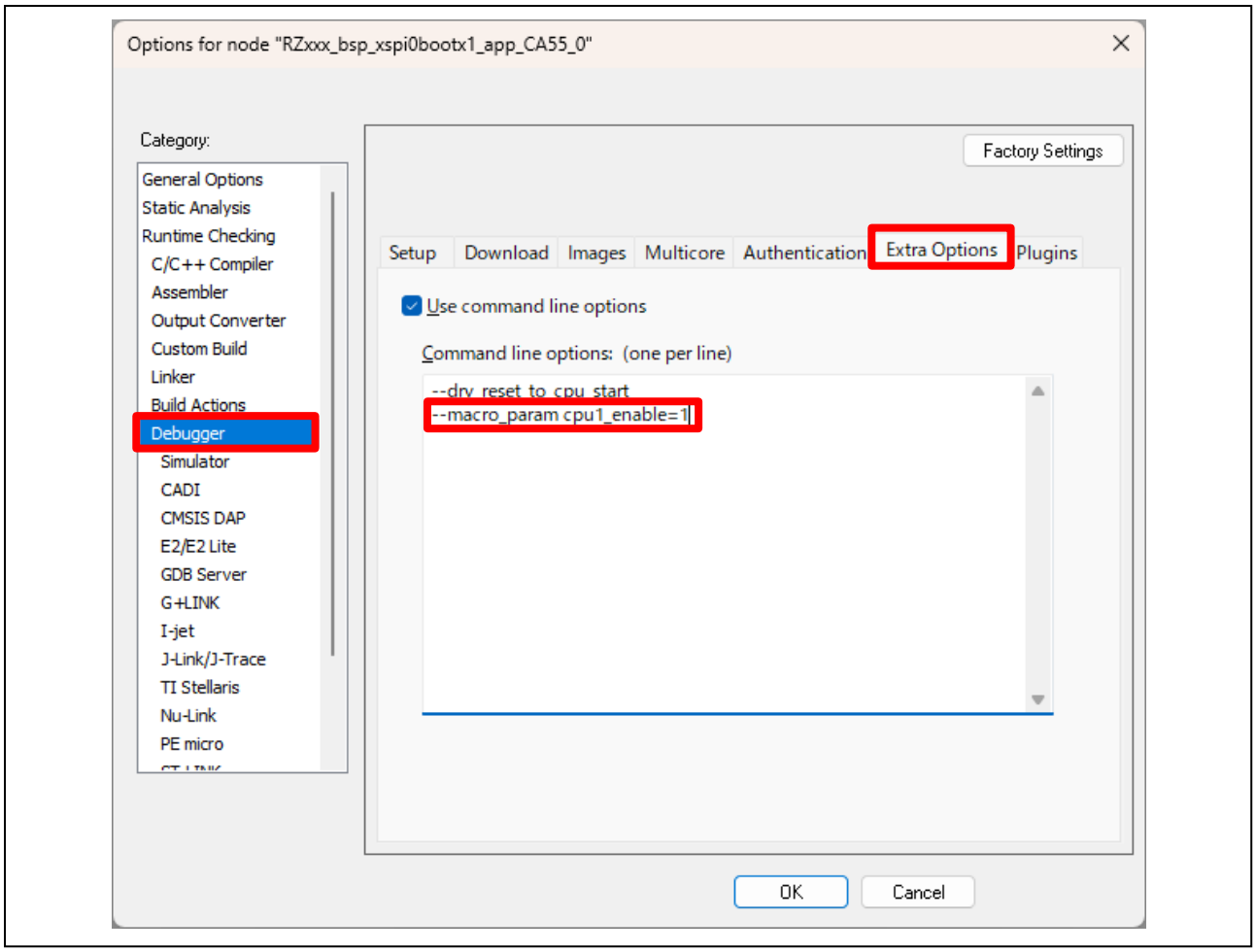

図 5-31 Cortex-A55 CPU0 プロジェクトのコマンドライン設定

EWARM 環境における補足事項③

以下のオプション設定により、コンパイル前に設定されているコード生成用アクションを削除しています。
このアクションはコンパイル前にコード生成を行う機能であり、コード生成の実行によりリンカスクリプトファイルのパスがデフォルト設定に上書きされてしまうため、本設定によりこれを防止しています。

1. [Project]→[Options]を開きます。
2. [Build Actions]を開き、[Build order]が"Pre-compile"のものを選択して、[Remove]をクリックします。

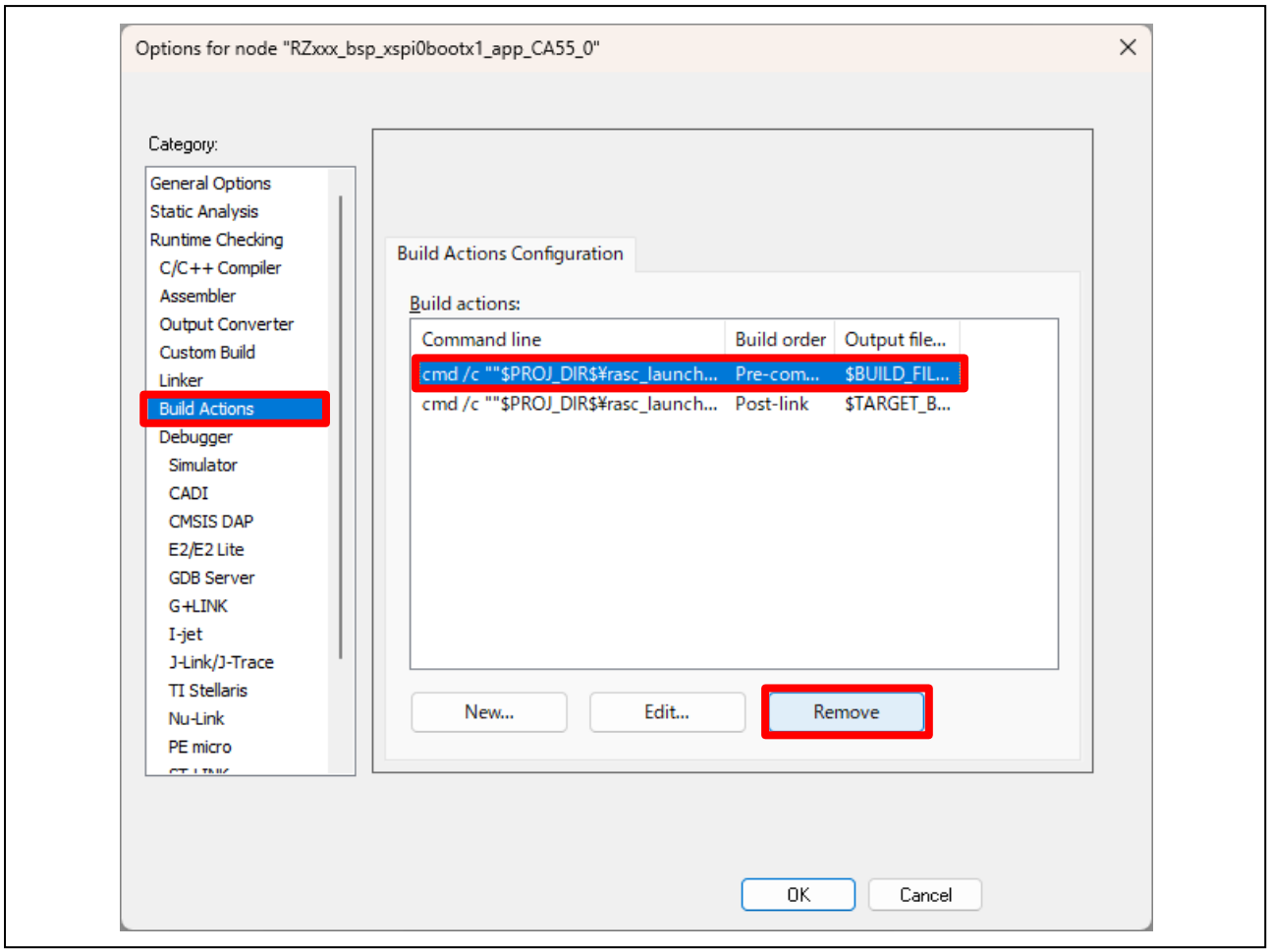

図 5-32 Cortex-A55 CPU0 プロジェクトのビルドアクション消去

EWARM 環境における補足事項④

Cortex-A55 CPU0 プロジェクトのデバッグ設定を以下のように変更しています。

1. [Project]→[Options]を開きます。
2. [Debugger]→[Setup]を開き、"Run to"のチェックを外します。
3. [I-jet]→[Setup]を開き、"From the probe"のチェックを外し、[Reset]を"Software"に変更します。
4. [I-jet]→[Interface]を開き、"From file"を選択し、[CPU]で"CA55_0"を選択します。

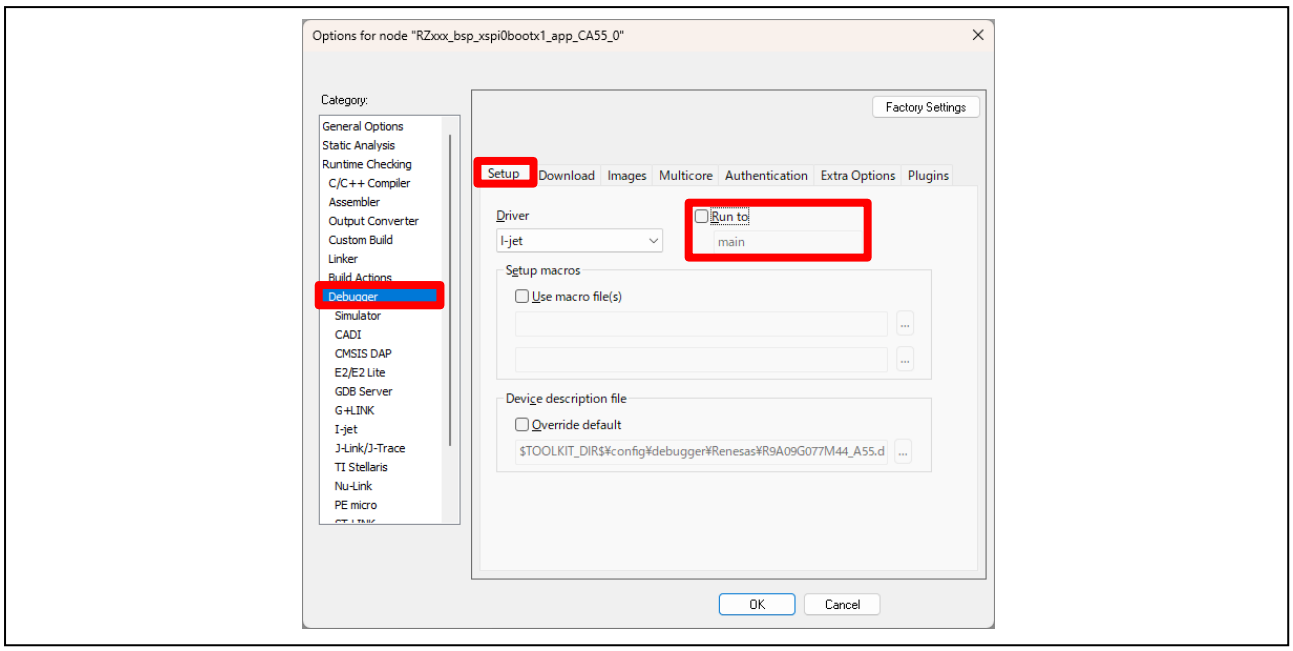

図 5-33 Cortex-A55 CPU0 プロジェクトのデバッグ設定①

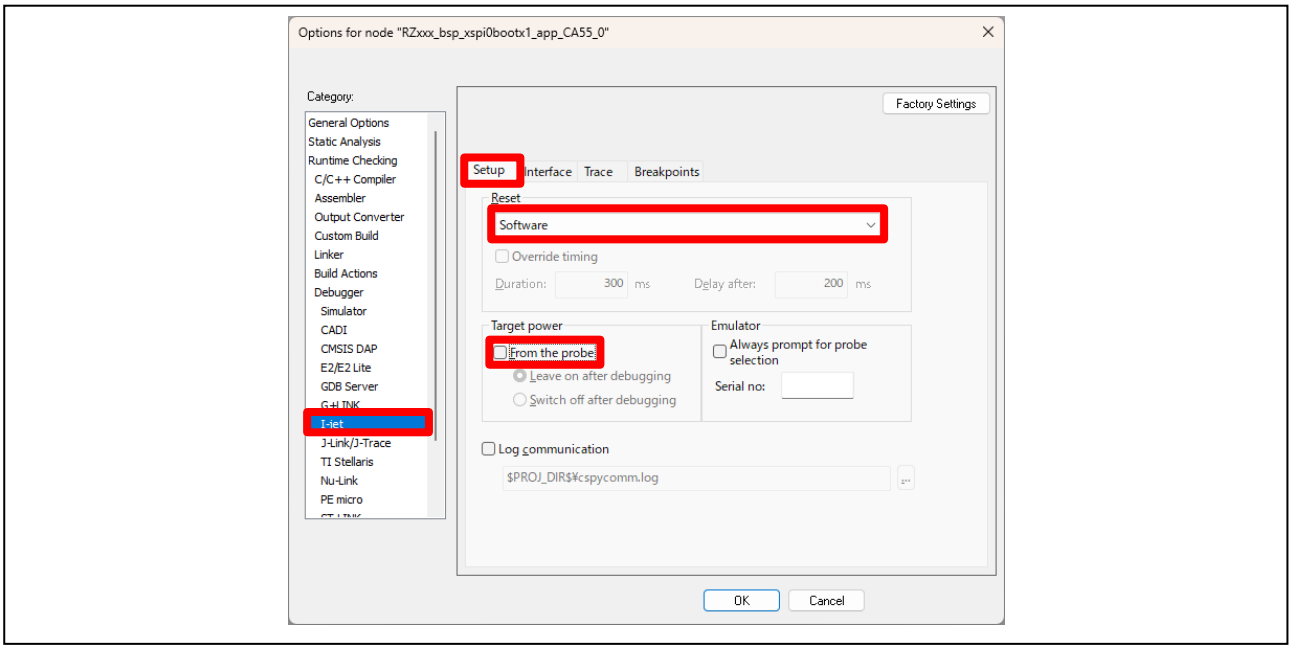

図 5-34 Cortex-A55 CPU0 プロジェクトのデバッグ設定②

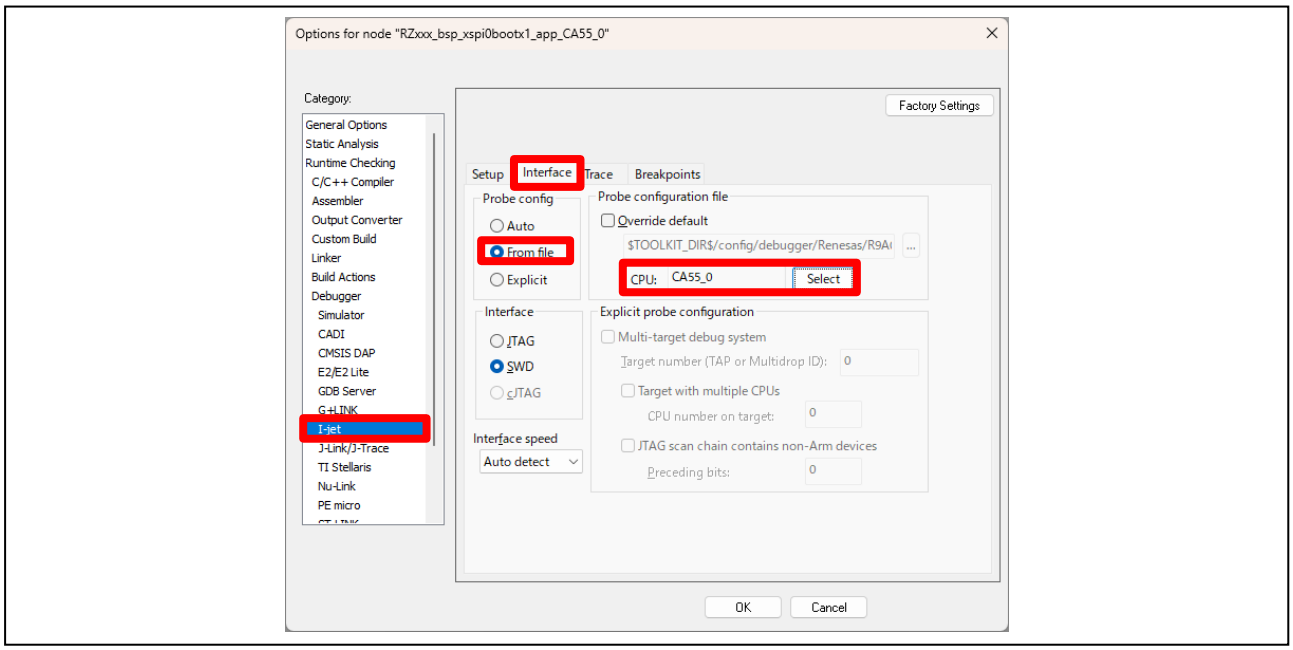

図 5-35 Cortex-A55 CPU0 プロジェクトのデバッグ設定③

5.3.2 e² studio : Renesas 社製

1. Cortex-A55 CPU0 プログラムのビルド

“RZ/T2H_bsp_xspi0bootx1_app_CA55_0”プロジェクトの configuration.xml を開き、[Generate Project Content]ボタンを押してコード生成を実行します。

“RZ/T2H_bsp_xspi0bootx1_app_CA55_0”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。

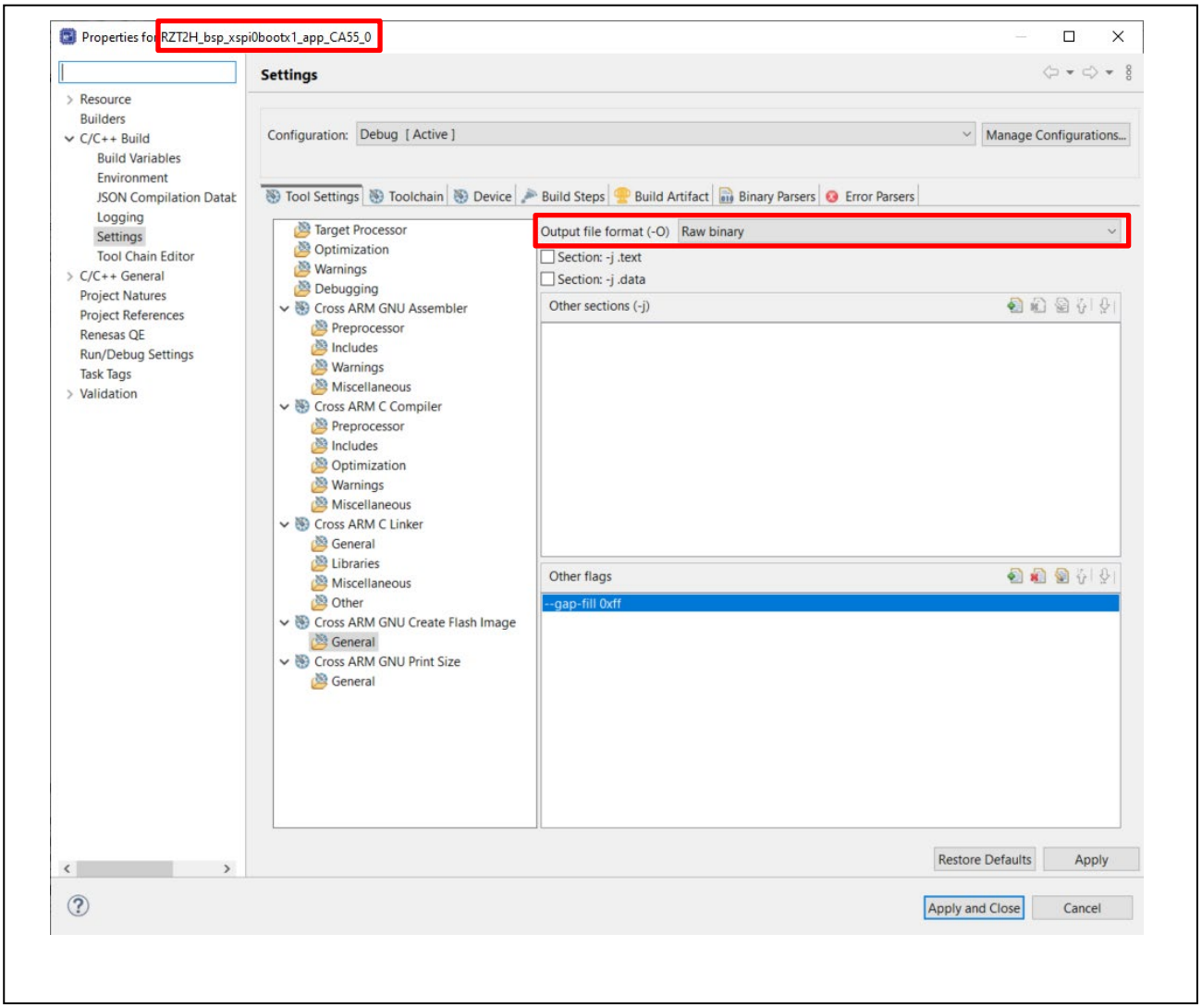

図 5-36 Cortex-A55 CPU0 プログラムのオプション設定

2. Cortex-A55 CPU0 プログラムを Cortex-R52 CPU0 ローダプログラムへリンク

Cortex-A55 CPU0 のバイナリを Cortex-R52 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がリンカスクリプトファイルに追加されています。

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))
    } > SYSTEM_RAM

    .IMAGE_APP_FLASH_section 0x40010000 : AT (0x40010000)
    {
        IMAGE_APP_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_APP_FLASH_section))
        IMAGE_APP_FLASH_section_end = .;
    } > FLASH_CONTENTS

    .IMAGE_CA55_CPU0_RAM 0x10000000 : AT (0x10000000)/* CA55_CPU0 program RAM section for execution */
    {
        IMAGE_CA55_CPU0_RAM_start = .;
        KEEP(*(CA55_CPU0_IMAGE_RAM))
    } > SYSTEM_RAM

    .IMAGE_CA55_CPU0_FLASH_section 0x40040000 : AT (0x40040000) /* CA55_CPU0 program, ROM section. */
    {
        IMAGE_CA55_CPU0_FLASH_section_start = .;
        KEEP(./src/Flash_section.o(.IMAGE_CA55_CPU0_FLASH_section))
        IMAGE_CA55_CPU0_FLASH_section_end = .;
    } > FLASH_CONTENTS

    .loader_param LOADER_PARAM_ADDRESS :
    {
        KEEP(*(.loader_param))
    } > FLASH_CONTENTS

~~~~~~~~
}

IMAGE_APP_FLASH_section_size = SIZEOF(.IMAGE_APP_FLASH_section);
IMAGE_CA55_CPU0_FLASH_section_size = SIZEOF(.IMAGE_CA55_CPU0_FLASH_section);
```

3. USE_CA55_CPU0 定義の有効化

Cortex-A55 CPU0 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_CA55_CPU0」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_CA55_CPU0=1
[Cross ARM C Compiler] -> [Preprocessor]: USE_CA55_CPU0=1

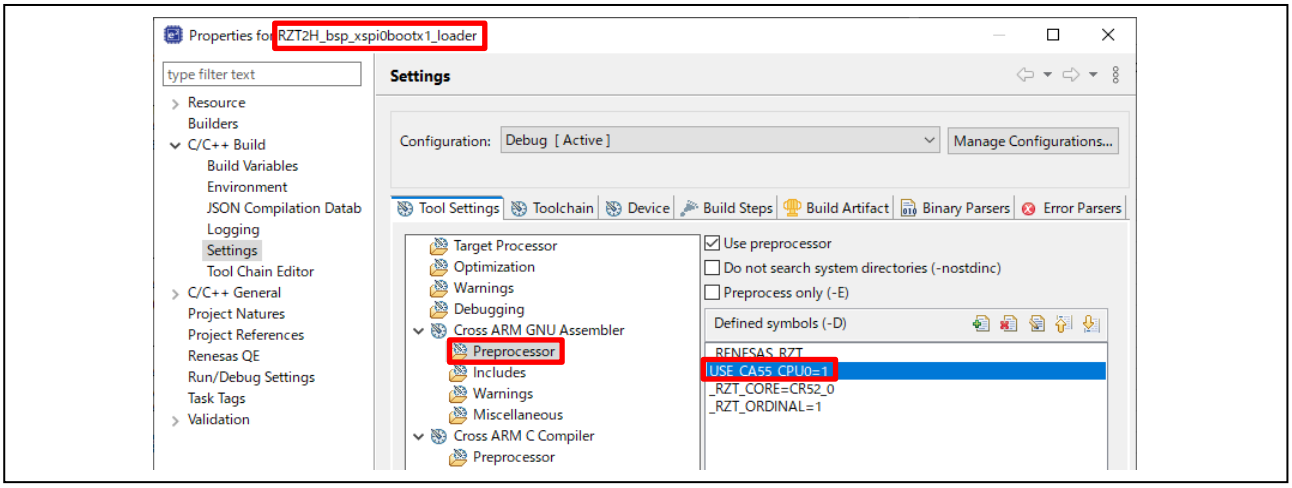

図 5-37 Cortex-R52 CPU0 のローダプログラムの USE_CA55_CPU0 定義有効化 (1/2)

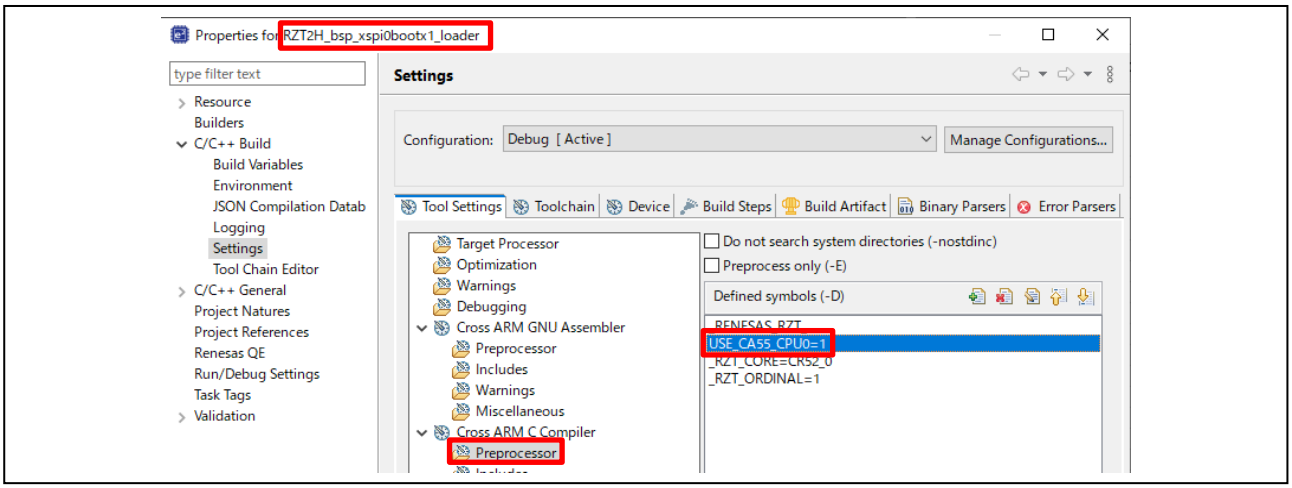

図 5-38 Cortex-R52 CPU0 のローダプログラムの USE_CA55_CPU0 定義有効化 (2/2)

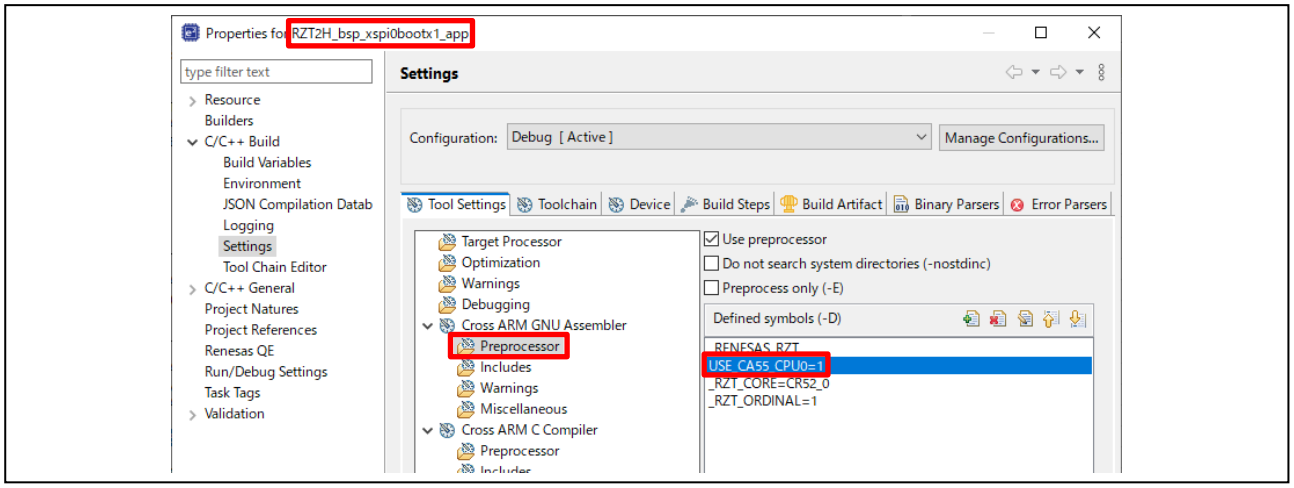

図 5-39　Cortex-R52 CPU0 のアプリケーションプログラムの USE_CA55_CPU0 定義有効化（1/2）

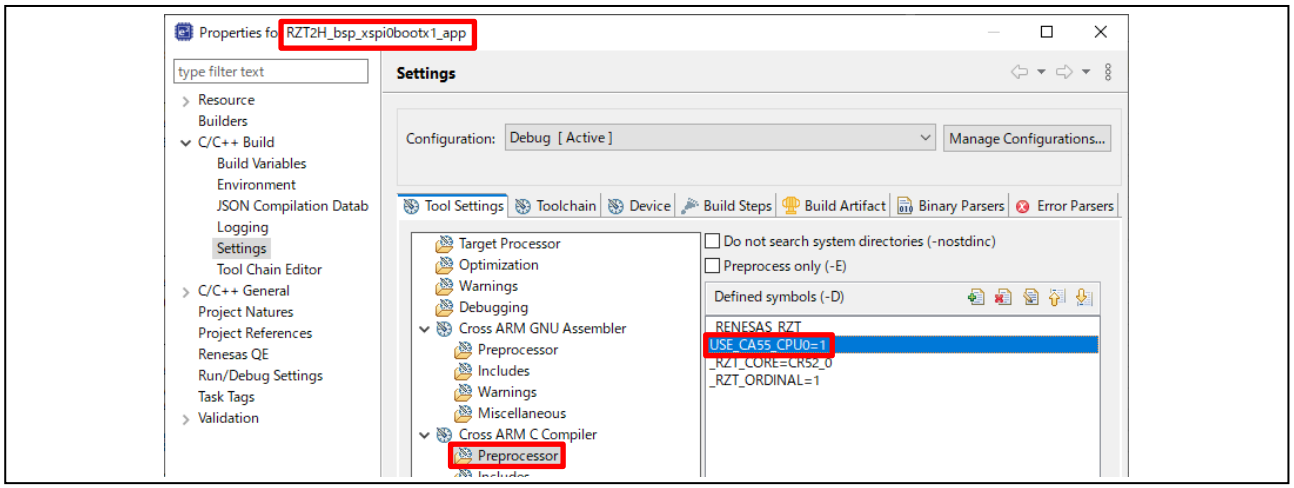

図 5-40　Cortex-R52 CPU0 のアプリケーションプログラムの USE_CA55_CPU0 定義有効化（2/2）

4. プロジェクトの再ビルドと実行

5.1.2 の手順に従い、ダウンロードとデバッグを開始してください。
デバッグ接続時に Cortex-A55 CPU0 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。

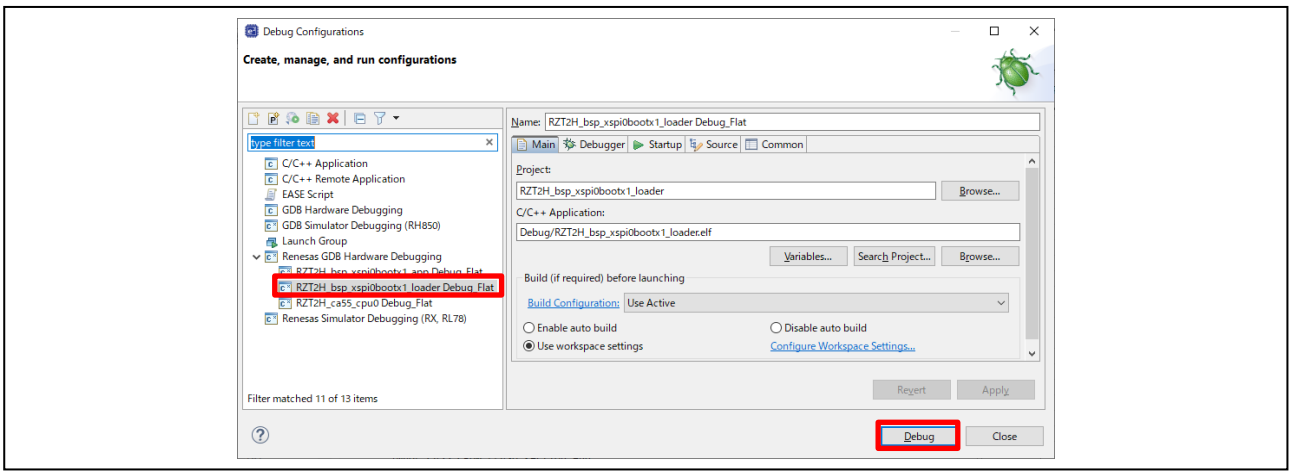

図 5-41　Cortex-R52 CPU0 のローダプログラムのデバッグ開始

Cortex-A55 CPU0 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 Cortex-A55 CPU0 プロジェクトのデバッグ接続を開始してください。

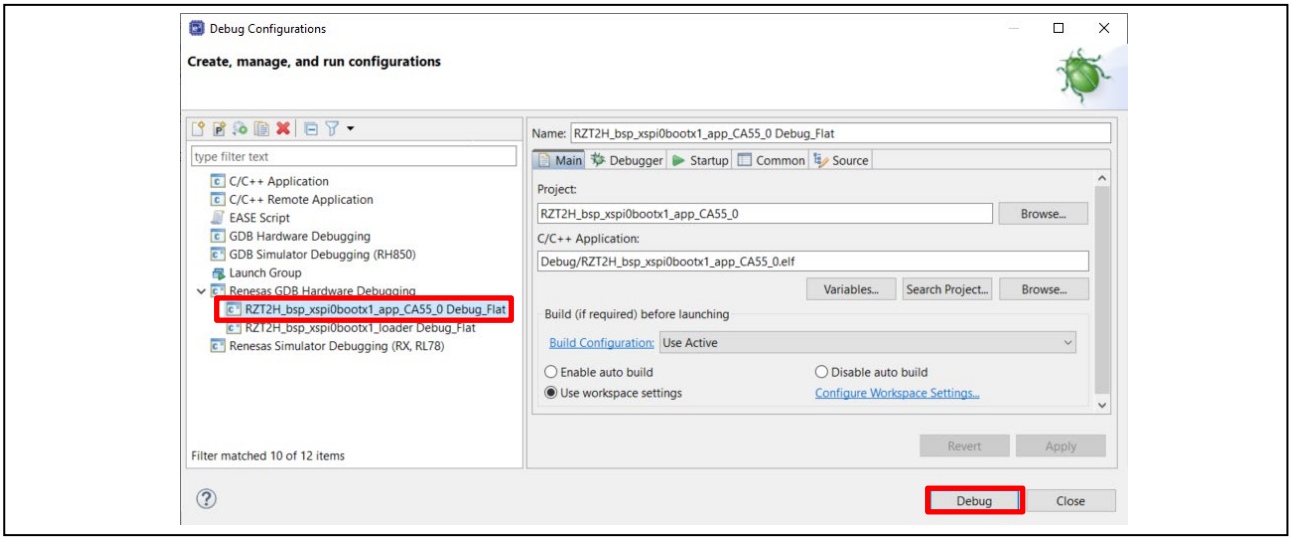

図 5-42　Cortex-A55 CPU0 プロジェクトのデバッグ開始

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。

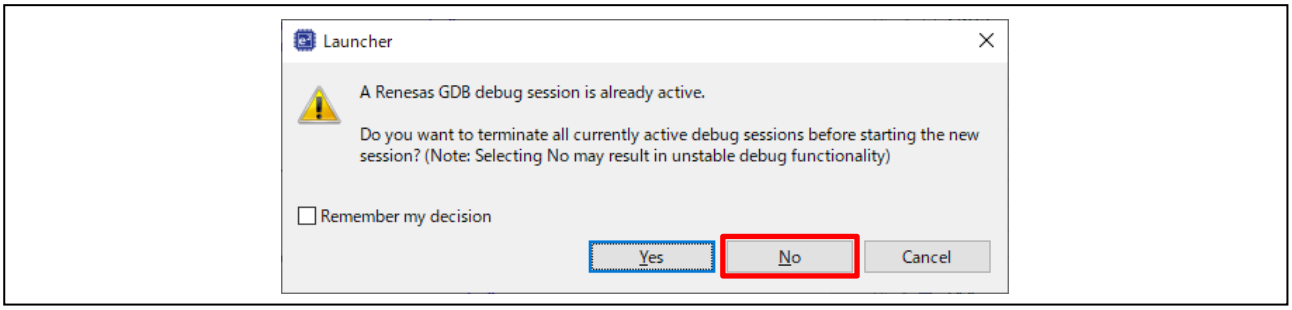

図 5-43　Cortex-R52 CPU0 のローダプログラムのデバッグ継続確認画面

Cortex-A55 CPU0 プロジェクトのデバッグ接続が成功すると Cortex-R52 CPU0 と Cortex-A55 CPU0 がデバッガに接続された状態となります。以降、Cortex-R52 CPU0 と Cortex-A55 CPU0 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 Cortex-R52 CPU0 と Cortex-A55 CPU0 で切り替えることが可能です。

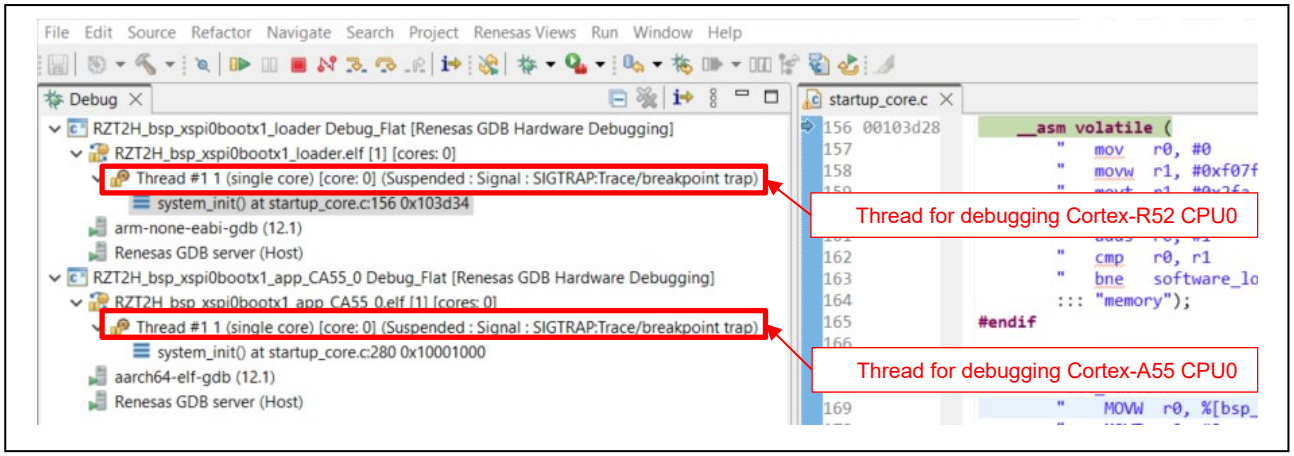

図 5-44　e² studio のデバッグ

Cortex-R52 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 が点滅します。
Cortex-A55 CPU0 プロジェクトの Thread を選択し実行すると、Cortex-A55 CPU0 プログラムが実行され、LED2 が点滅します。

e² studio 環境における補足事項

以下のオプション設定により、Cortex-A55 CPU0 プロジェクトのリンカスクリプトファイルのパスを変更しています。

1. Cortex-A55 CPU0 プロジェクトのノードを右クリックし、[Properties]を開きます。
2. [C/C++ Build]→[Settings]→[Tool Settings]→[Cross ARM C Linker]→[General]を開き、ファイルパスを"fsp_ram_execution_ca55_cpu0.ld"変更します。

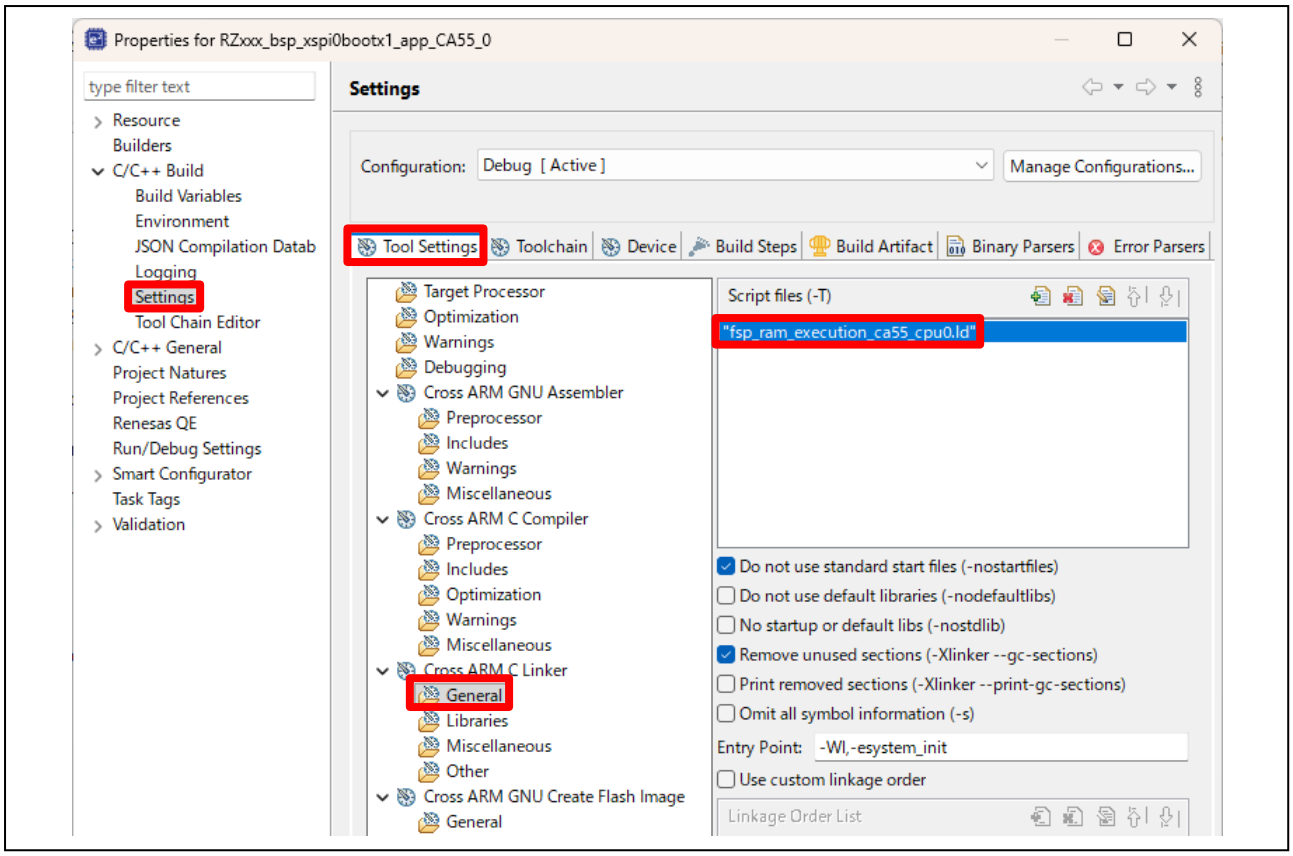

図 5-45 Cortex-A55 CPU0 プロジェクトのリンカスクリプトファイルのパス設定

5.3.3 Cortex-A55 CPU0 プログラムの補足事項

以下の FSP 設定により、Cortex-A55 CPU0 プロジェクトの C ランタイムを無効化しています。
Cortex-A55 CPU0 プロジェクトの C ランタイム初期化はローダプロジェクトで行われています。

1. Cortex-A55 CPU0 プロジェクトの configuration.xml を開きます。
2. [BSP]→[C Runtime Initialization]を"Disabled"に変更します。

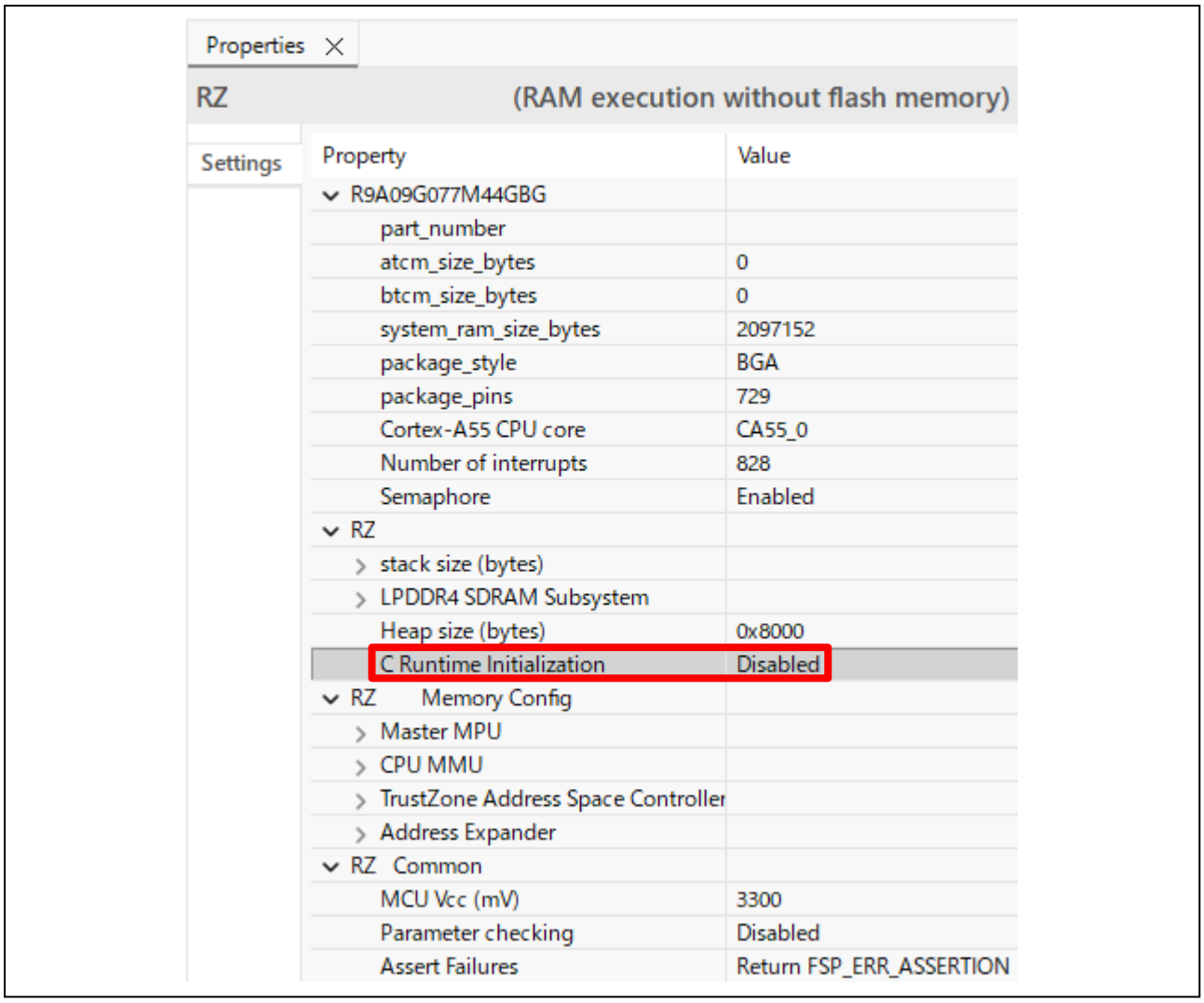

図 5-46 CA55_CPU0 プログラムの BSP 設定

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
2.00	2025.6.15	-	初版
3.00	2025.12.5	-	FSP3.0.0 対応
3.01	2026.4.1	30 ~ 33	サンプルプログラムの EWARM 環境における補足事項の追記
		35	e² studio 環境のデバッグ手順の変更
		36, 37	サンプルプログラムの e² studio 環境における補足事項の追記
		39, 40	サンプルプログラムのアプリケーションプログラムとローダプログラムの補足事項の追記
		51 ~ 55	Cortex-A55 CPU0 プログラムの EWARM 環境における補足事項の追記
		62	Cortex-A55 CPU0 プログラムの e² studio 環境における補足事項の追記
		63	Cortex-A55 CPU0 プログラムの補足事項

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。