RZ/T1 グループ

CMSIS-RTOS RTX for Cortex-R4

RTX サンプルプログラム

(EWARM / ICCARM, e2studio/RenesasGCC, DS-5/ARMCC)

R01AN3538JJ0100
Rev.1.00
2016.12.22

要旨

本アプリケーションノートでは、RZ/T1 上で動作する CMSIS-RTOS RTX サンプルアプリケーションについて説明します。

CMSIS-RTOS RTX サンプルプログラムの特長を以下に示します。

各動作モードに対応した 16 ビットバスブート版、SPI ブート版、および外付けフラッシュを必要とせず内蔵 RAM（TCM）のみ使用する RAM 実行版から構成されます。

本サンプルプログラムはローダプログラム部と CMSIS-RTOS RTX サンプルプログラム部で構成され、ブート後、順に RZ/T1 の初期設定、CMSIS-RTOS 初期化、サンプルアプリケーション実行を行います。

ローダプログラムでは、RZ/T1 のブート後にクロック発生回路、バス・ステート・コントローラなどの初期化やサンプルプログラム自体のコピー処理などを行います。

サンプルプログラムとしては、以下を用意しています（内容は表 5.1 を参照ください）。

FPU_ex1
mail
message
RTX_CMT_ex1
RTX_ex1
RTX_ex2
RTX_MTU3_ex1
RTX_Traffic
Semaphore

対象デバイス

RZ/T1

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

表 1.1 に使用する周辺機能と用途を示します。

表 1.1 使用する周辺機能と用途

周辺機能	用途
CPU(R7X910017)	プログラム動作用
RZ/T1 内蔵バス・ステート・コントローラ	16 ビットバスブートモード(NOR フラッシュ)[注1]起動時のプログラム本体の読み出しと SDRAM のアクセス
RZ/T1 内蔵 SPI マルチ I/O バス・コントローラ	SPI ブート[注2]モード起動時のプログラム本体の読み出し
NOR フラッシュメモリ（MX29GL512FLT2I-10Q）	16 ビットバスブート[注1]モード(NOR フラッシュ)起動時のプログラム本体の格納領域
シリアルフラッシュメモリ(MX25L51245GMI-10G)	SPI ブートモード起動時のプログラム本体の格納領域
RZ/T1 内蔵 RAM(ATCM,BTCM)	RAM ブート[注3]時のプログラム本体の格納領域 プログラム・データ格納領域
RZ/T1 内蔵クロック・パルス発信器	RZ/T1 各種内蔵マクロへのクロック供給制御 各種クロック・ソースの周波数生成（倍率指定）
RZ/T1 内蔵汎用入出力ポート	RZ/T1 兼用端子
RZ/T1 内蔵割り込みコントローラ(ICUA)	割り込み制御

注１） 16 ビットバスブート ：NOR フラッシュからのブート（以後、NOR ブートと記載する）

注２） SPI ブート ：シリアルフラッシュからのブート（以後、SPI ブートと記載する）

注３） RAM ブート ：JTAG-ICE を用いた内蔵 RAM への直接ダウンロードによる起動（以後、RAM ブートと記載する）

2. 動作環境

本アプリケーションノートのサンプルコードは、下記の条件で動作を確認しています。

表 2.1 動作確認条件

項目	内容
使用マイコン	RZ/T1 グループ
動作周波数	CPUCLK = 450MHz
動作電圧	3.3V
統合開発環境	IAR システムズ製 Embedded Workbench for ARM Version 7.80.2 （以下、EWARM と略す） RENESAS 製 e2studio 5.2.0.020（以下、e2studio と略す） ARM 製 DS-5 Version: 5.25.0（以下、DS-5 と略す）
動作モード	SPI ブートモード SW4：ON/ON/ON
	RAM ブートモード、NOR ブートモード SW4：ON/OFF/ON
使用ボード	RZ/T1 評価ボード (RTK7910018C00000BE)
使用デバイス	● NOR フラッシュメモリ (CS0、CS1 空間に接続) メーカ名: Macronix International Co., Ltd. 型名: MX29GL512FLT2I-10Q ● SDRAM (CS2、CS3 空間に接続) メーカ名: Integrated Silicon Solution Inc. 型名: IS42S16320D-7TL ● シリアルフラッシュメモリ メーカ名: Macronix International Co., Ltd. ● 型名: MX25L51245GMI-10G
ICE	IAR システムズ製 I-jet JTAG エミュレータ SEGGER 社製 J-Link JTAG エミュレータ ARM 社製 KEIL ULINK2 エミュレータ

3. ドキュメント

3.1 関連文書一覧

本書に関連する文書を以下に示します。

- CMSIS-RTOS compliant Kernel Version 4.74

  本システムで使用している RTOS の仕様書です。サンプルアプリケーションは、CMSIS-RTOS RTX の機能を使用します。また、各サンプルアプリケーションは CMSIS-RTOS RTX を初期化します。

- RZ/T1 グループ ユーザーズマニュアル ハードウェア編（R01UH0483）

  RZ/T1 のハードウェア仕様を記載しています。

  （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/T1 グループ 初期設定 アプリケーションノート（R01AN2554）

  RZ/T1 の初期設定について記載しています。本書に記載のないものは、この「RZ/T1 グループ 初期設定アプリケーションノート」で設定した値を使用します。

- RZ/T1 グループ CMSIS-RTOS RTX for Cortex-R4 CMT(W) & ELC & ADC サンプルプログラム アプリケーションノート（R01AN3539JJ）

  RZ/T1 の 32 ビットタイマ(CMTW)、およびイベントリンクコントローラ（ELC）を用いて、タイマと ADC 機能を連動させ A/D 変換を行うサンプルプログラムについて記載しています。

- RZ/T1 グループ CMSIS-RTOS RTX for Cortex-R4 MTU3a サンプルプログラム アプリケーションノート（R01AN3540JJ）

  RZ/T1 のマルチファンクションタイマパルスユニット（MTU3a）の MTU3、MTU4 の相補 PWM モードを使用して、正と負の 3 相（6 本）のデッドタイム付き PWM 波形を出力するサンプルプログラムについて記載しています。

- テクニカルアップデート／テクニカルニュース

  （最新の情報をルネサス エレクトロニクスホームページから入手してください。）

- 開発環境に関わるユーザーズマニュアル

  IAR 統合開発環境（IAR Embedded Workbench for ARM）に関しては、最新版を IAR ホームページから入手してください。

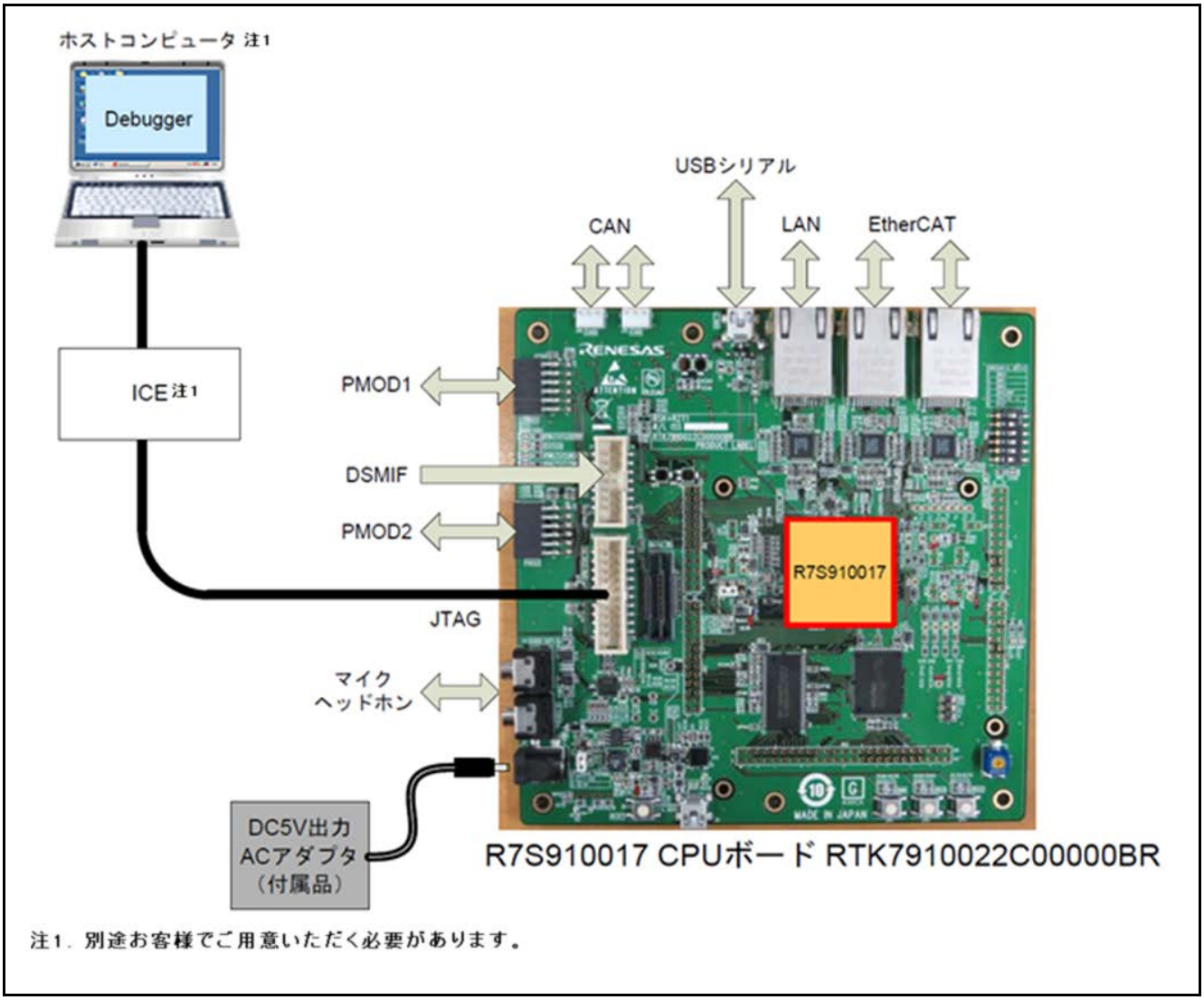
  DS-5 統合開発環境（ARM Development Studio 5）に関しては、最新版を ARM ホームページから入手してください。

  ルネサス エレクトロニクスソフトウェア開発ツール（e2studio 等）に関しては、ルネサスエレクトロニクスホームページから最新版を入手してください。

4. ハードウェア説明

4.1 ハードウェア構成例

以下に接続例を示します。



4.1 接続例

4.2 使用端子一覧

表 4.1 に使用端子と機能を示します。

表 4.1 使用端子と機能

端子名	入出力	内容	RZ/T1 端子設定	ボード接続
MD0	入力	動作モードの選択 MD0="L"、MD1="L"、MD2="L" （SPI ブートモード） MD0="L"、MD1="H"、MD2="L" （NOR ブートモード）	MD0 専用端子	SW4-1 ※MD0～2 の組合せにより動作モードを選択
MD1	入力		MD1 専用端子	SW4-2 ※MD0～2 の組合せにより動作モードを選択
MD2	入力		MD2 専用端子	SW4-3 ※MD0～2 の組合せにより動作モードを選択
A21 ～25	出力	CSn 空間へのアドレス指定	PT6～PT7 PK2～PK3 P97 端子機能選択ビット=35 動作モード=SPI ブートモード	SDRAM
A16 ～20	出力	CSn 空間へのアドレス指定	PH7 P20、P25～P27 端子機能選択ビット=34 動作モード=SPI ブートモード、NOR ブートモード(NOR ブートモード時は自動設定)	NOR フラッシュ SDRAM
A1 ～A15	出力	CSn 空間へのアドレス指定	PG0～PD7 PH0～PH6 端子機能選択ビット=34 動作モード=SPI ブートモード、NOR ブートモード(NOR ブートモード時は自動設定)	NOR フラッシュ SDRAM
A0	出力	CSn 空間へのアドレス指定	P23 端子機能選択ビット=34 動作モード=SPI ブートモード	NOR フラッシュ SDRAM
D0 ～D15	入出力	CSn 空間との入出力データ	P00～P07 PE0～PE7 端子機能選択ビット=34 動作モード=SPI ブートモード、NOR ブートモード(NOR ブートモード時は自動設定)	NOR フラッシュ SDRAM
CS0#	出力	CS0 空間へのチップセレクト	P21 端子機能選択ビット=34 動作モード=NOR ブートモード (NOR ブートモード時は自動設定)	NOR フラッシュ
CS1#	出	CS1 空間へのチップセレ	PD1	SDRAM

	力	クト	端子機能選択ビット=35 動作モード=SPI ブートモード、 NOR ブートモード	
RD/ WR#	出力	CS0, CS1 空間へのリード・データ出力許可	P24 端子機能選択ビット=34 動作モード=SPI ブートモード、 NOR ブートモード(NOR ブートモード時は自動設定)	NOR フラッシュ SDRAM
CS2#	出力	CS2 空間へのチップセレクト	P45 端子機能選択ビット=34 動作モード=SPI ブートモード、 NOR ブートモード	SDRAM
CS3#	出力	CS3 空間へのチップセレクト	PT4 端子機能選択ビット=34 動作モード=SPI ブートモード、 NOR ブートモード	SDRAM
RAS#	出力	SDRAM RAS#端子に接続	P90 端子機能選択ビット=35 動作モード=SPI ブートモード、 NOR ブートモード	SDRAM
CAS#	出力	SDRAM CAS#端子に接続	PK0 端子機能選択ビット=35 動作モード=SPI ブートモード、 NOR ブートモード	SDRAM
CKE	出力	SDRAM CKE 端子に接続	P46 端子機能選択ビット=34 動作モード=SPI ブートモード、 NOR ブートモード	SDRAM
WE0/ DQMLL	出力	バイト指定	P36 端子機能選択ビット=34 動作モード=SPI ブートモード、 NOR ブートモード	SDRAM
WE1/ DQMLU	出力	バイト指定	P37 端子機能選択ビット=34 動作モード=SPI ブートモード、 NOR ブートモード	SDRAM
CKIO	出力	システム・クロック出力	P10 端子機能選択ビット=34 動作モード=SPI ブートモード、 NOR ブートモード	NOR フラッシュ SDRAM
SPBCLK	出力	SPI マルチ I/O バス・コントローラ クロック出力	P62 端子機能選択ビット=27 動作モード=SPI ブートモード (SPI ブートモード時自動設定)	シリアルフラッシュメモリ
SPBSSL	出力	SPI マルチ I/O バス・コントローラ スレーブセレクト	P60 端子機能選択ビット=27 動作モード=SPI ブートモード (SPI ブートモード時自動設定)	シリアルフラッシュメモリ
SPBMO/	入	SPI マルチ I/O バス・コン	P63	シリアルフラッシュメ

SPBIO0	出力	トローラ チャネル 0 のデータ入出力	端子機能選択ビット=27 動作モード=SPI ブートモード (SPI ブートモード時自動設定)	モリ
SPBMI/ SPBIO1	入出力	SPI マルチ I/O バス・コントローラ チャネル 0 のデータ入出力	P64 端子機能選択ビット=27 動作モード=SPI ブートモード (SPI ブートモード時自動設定)	シリアルフラッシュメモリ
SPBIO2	入出力	SPI マルチ I/O バス・コントローラ チャネル 0 のデータ入出力	P65 端子機能選択ビット=27 動作モード=SPI ブートモード (SPI ブートモード時自動設定)	シリアルフラッシュメモリ
SPBIO3	入出力	SPI マルチ I/O バス・コントローラ チャネル 0 のデータ入出力	P61 端子機能選択ビット=27 動作モード=SPI ブートモード (SPI ブートモード時自動設定)	シリアルフラッシュメモリ
RxD2	入力	SCIF チャネル 2 受信データ入力	P91 端子機能選択ビット=11	RZ/T1 評価ボード USB/シリアルポート
TxD2	出力	SCIF チャネル 2 送信データ出力	P92 端子機能選択ビット=11	RZ/T1 評価ボード USB/シリアルポート

※　リセット時にハードウェアによってに設定される端子は、初期化処理で再設定はしません。
※　CMSIS-RTOS RTX サンプルプログラムの SCIF へのデバッグ・メッセージ出力は、SCIF ドライバの機能を利用せず、直接出力しています。

5. ソフトウェア説明

5.1 動作概要

CMSIS-RTOS RTX サンプルプログラムの動作概要を以下に示します。

RZ/T1 はリセット解除後のブート処理で、各動作モード（NOR ブート／SPI ブート）に対応する外付けフラッシュ（NOR／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内蔵 RAM（BTCM）へコピーします。ローダプログラムでは、ブート処理後にリセット判定、クロック設定、バス設定などを行った後、外付けフラッシュ（NOR／シリアルフラッシュメモリ）に格納されたユーザーアプリケーションプログラムを内蔵 RAM（ATCM）にコピーします。次に MPU 設定、キャッシュ設定を行い、CMSIS-RTOS RTX ユーザーアプリケーションプログラムの先頭に分岐し、それぞれのサンプルプログラム動作を行います。

- 対象ハードウェア
  RZ/T1 を搭載した RZ/T1 評価ボード
- IAR Embedded Workbench for ARM Version 7.80.1 対応
  IAR Embedded Workbench for ARM Version 7.80.1 のプロジェクト・ファイル形式でサンプルアプリケーション環境を提供
- RENESAS e2studio 5.2.0.020 対応
  RENESAS e2studio 5.2.0.020 のプロジェクト・ファイル形式でサンプルアプリケーション環境を提供
- ARM DS-5 Version: 5.25.0 対応
  ARM DS-5 Version: 5.25.0 のプロジェクト・ファイル形式でサンプルアプリケーション環境を提供
- エンディアン
  リトル専用
- ブート
  以下のいずれかを選択可能
  - シリアルフラッシュメモリから起動（SPI ブートモード）
  - NOR フラッシュメモリから起動（NOR ブートモード）
  - JTAG-ICE を用いた内蔵 RAM(ATCM,BTCM)への直接ダウンロードによる起動（RAM ブートモード）
- メモリ領域
  - コード、データ領域は内蔵 RAM(ATCM,BTCM)を使用
- メモリ管理
  - MPU を使用したメモリ管理
- ハードウェア初期化
  プラットフォーム初期化で以下のモジュールを初期化(初期化順)
  - クロック・パルス発振器(システム・クロック制御)
  - 汎用入出力ポート(端子機能設定)
  - バス・ステート・コントローラ(CS0 CS1 NOR フラッシュ、CS2 CS3 SDRAM)
  - CMT4(CMSIS-RTOS RTX システム・タイマ)
- CMSIS-RTOS RTX コンフィグレーション
  CMSIS-RTOS RTX のコンフィギュレーション項目は以下の設定で実装する
  - OS タイマとして CMT4 を使用(割り込み周期は 1ms)
  - PendSV 割り込みとして CMT5 を使用
  - idle 状態は WFI スリープ・モード

- スレッドのデフォルト・スタック・サイズ：1024byte
- スタック・オーバフロー・チェック機能：有効
- スレッド動作モード：特権モード
- ラウンドロビン動作：無効
- ISR イベント FIFO キュー数：16
- スレッド最大生成数のデフォルト値：50
- ユーザー・タイマ機能：有効
- タイマスレッドへのメッセージキューサイズ：5

各サンプルアプリケーションの概要を以下に示します。

表 5.1　サンプルアプリケーション概要

サンプルアプリケーション名	概要
FPU_ex1	浮動小数点の演算サンプルアプリケーションです。
Mail	RTX Kernel Mail Queue Management のサンプルアプリケーションです。
Message	RTX Kernel Message Management のサンプルアプリケーションです。
RTX_CMT_ex1 ※詳細は 3.1 章に記載の R01AN3539JJ を参照	RZ/T1 の 32 ビットタイマ(CMTW)、およびイベントリンクコントローラ(ELC)を用いて、タイマと ADC 機能を連動させ A/D 変換を行うサンプルアプリケーションです。
RTX_ex1	RTX Kernel Signal Management を使用して 2 つのスレッドを制御するサンプルアプリケーションです。
RTX_ex2	RTX Kernel Signal Management を使用して 4 つのスレッドを制御するサンプルアプリケーションです。
RTX_MTU3_ex1 ※詳細は 3.1 章に記載の R01AN3540JJ を参照	マルチファンクションタイマパルスユニット（MTU3a）の MTU3、MTU4 の相補 PWM モードを使用して、正と負の 3 相（6 本）のデッドタイム付き PWM 波形を出力するサンプルアプリケーションです。
RTX_Traffic	RTX Kernel Timer Management 及び Signal Flag Management のサンプルアプリケーションです。
Semaphore	RTX Kernel Semaphore Management のサンプルアプリケーションです。

5.2 初期化シーケンス

5.2.1 EWARM 環境

以下に、EWARM 初期化シーケンスを示します。

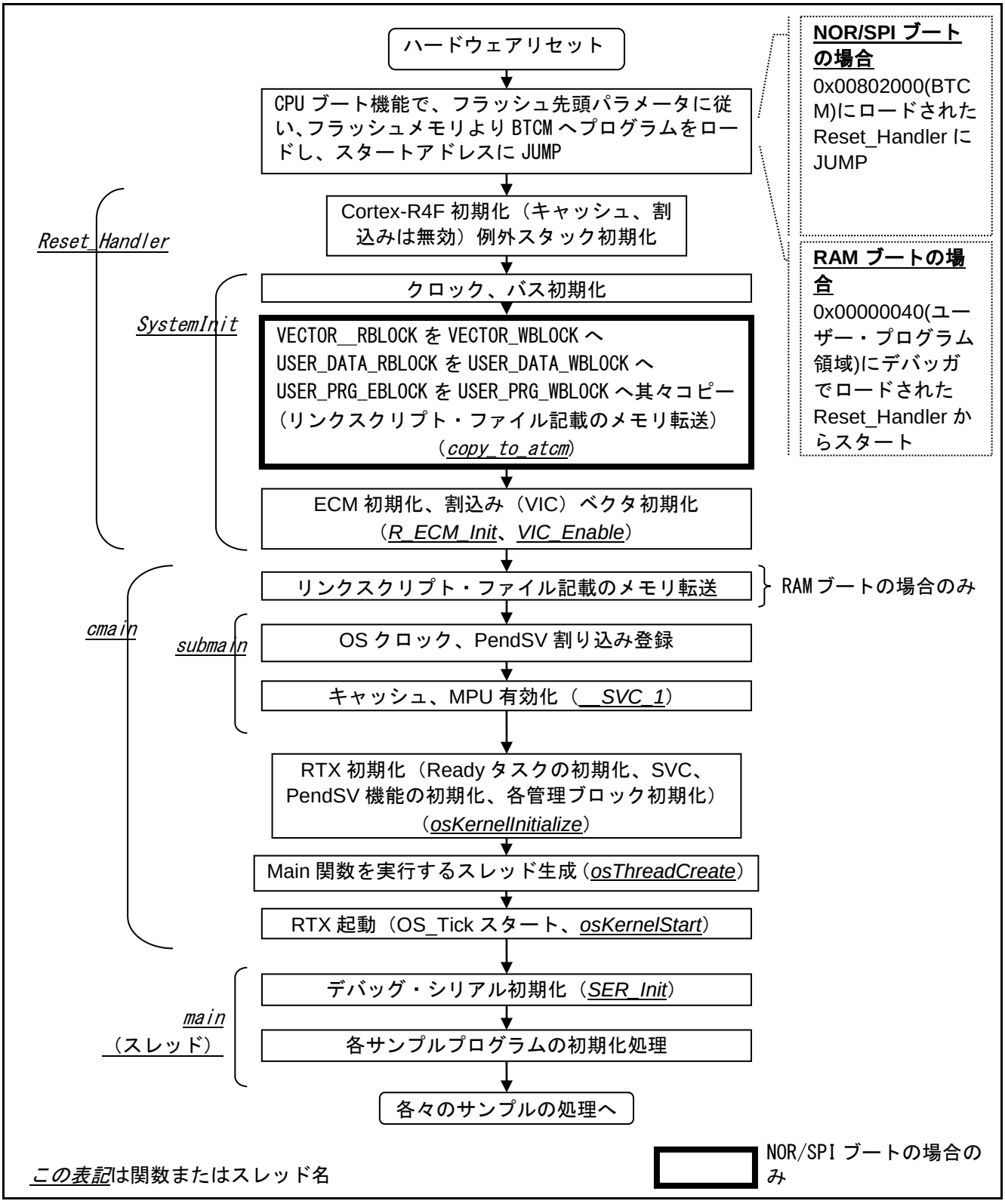


図 5.1 EWARM 初期化シーケンス

5.2.2 e2Studio 環境

以下に e2studio 初期化シーケンスを示します。

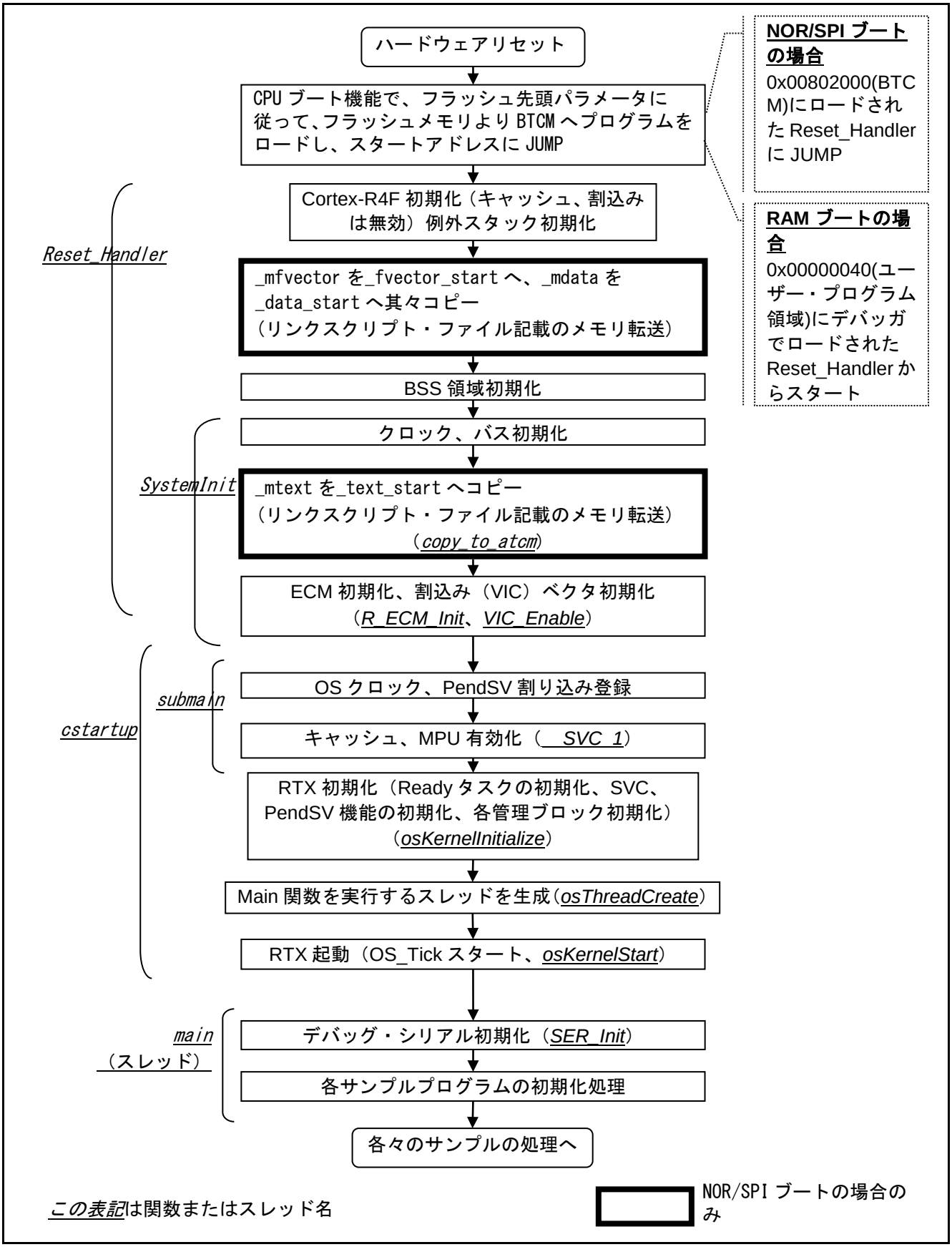


図 5.2 e2studio 初期化シーケンス

5.2.3 DS-5 環境

以下に DS-5 初期化シーケンスを示します。

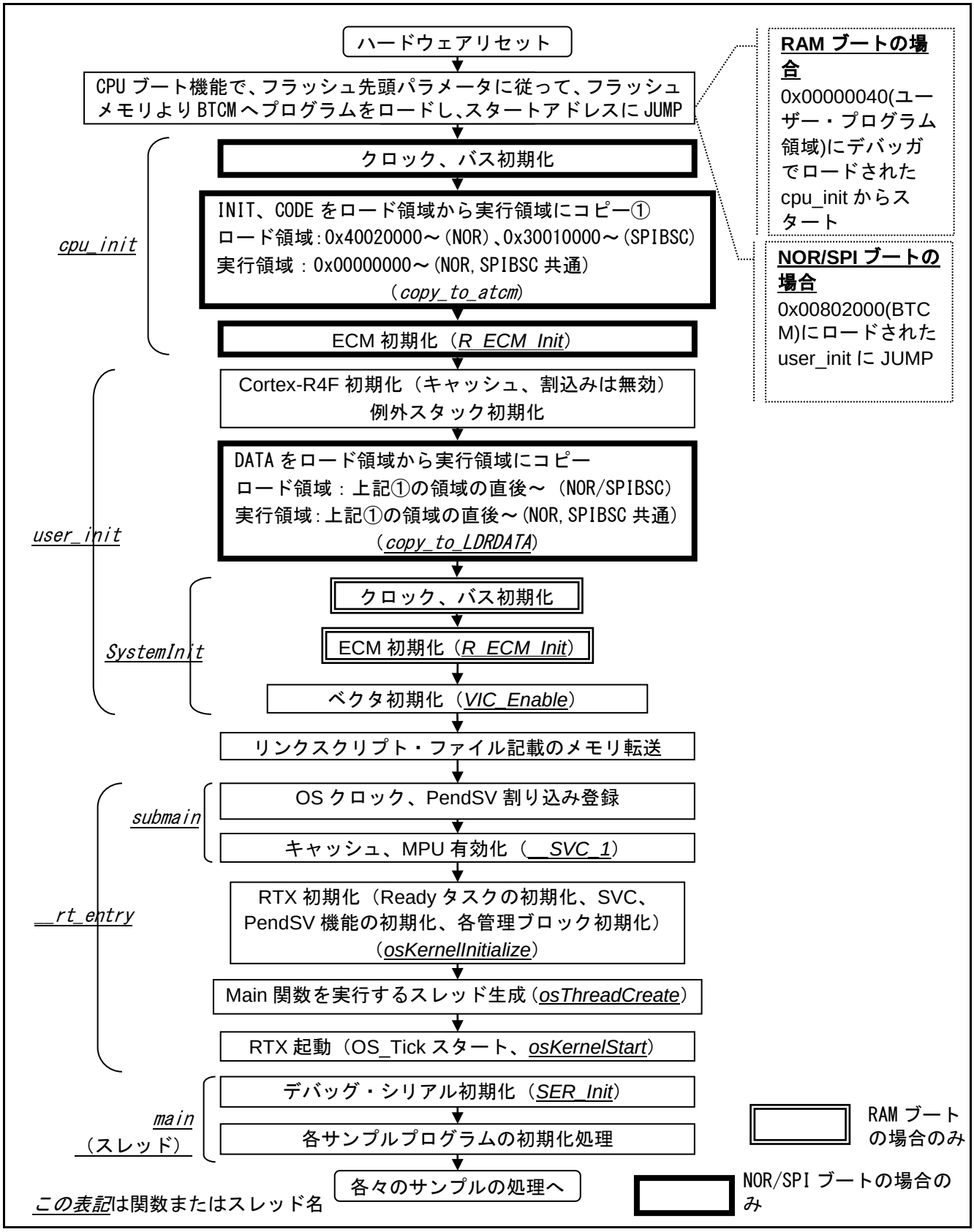


図 5.3 DS-5 初期化シーケンス

5.3 プロジェクト構成

CMSIS-RTOS RTX サンプルプログラムのワーキング・セットに含まれるプロジェクトの一覧を以下に示します。各プロジェクト（EWARM/e2studio/DS-5）の設定は実際のプロジェクトにて確認してください。

表 5.2 RZ/T1 ワーキング・セット構成

プロジェクト名	プロジェクト概要	ブートモード
Bootloader NorFlash	NOR ブートモード（NOR フラッシュ）ブートローダーライブラリ（Bootloader NorFlash.a）	NOR
Bootloader SerialFlash	SPI ブートモード（シリアルフラッシュ）ブートローダーライブラリ（Bootloader SerialFlash.a）	SPI
RTX_CMT_ex1_NOR	RZ/T1 の 32 ビットタイマ(CMTW)、およびイベントリンクコントローラ(ELC)を用いて、タイマと ADC 機能を連動させ A/D 変換を行うサンプルアプリケーション	NOR
RTX_CMT_ex1_RAM		RAM
RTX_CMT_ex1_SPIBSC		SPI
RTX_MTU3_ex1_NOR	マルチファンクションタイマパルスユニット（MTU3a）の MTU3、MTU4 の相補 PWM モードを使用して、正と負の 3 相（6 本）のデッドタイム付き PWM 波形を出力するサンプルアプリケーション	NOR
RTX_MTU3_ex1_RAM		RAM
RTX_MTU3_ex1_SPIBSC		SPI
FPU_ex1_NOR	浮動小数点の演算サンプルアプリケーション	NOR
FPU_ex1_RAM		RAM
FPU_ex1_SPIBSC		SPI
Mail_NOR	RTX Kernel Mail Queue Management のサンプルアプリケーション	NOR
Mail_RAM		RAM
Mail_SPIBSC		SPI
Message_NOR	RTX Kernel Message Management のサンプルアプリケーション	NOR
Message_RAM		RAM
Message_SPIBSC		SPI
RTX_ex1_NOR	RTX Kernel Signal Management を使用して 2 つのスレッドを制御するサンプルアプリケーション	NOR
RTX_ex1_RAM		RAM
RTX_ex1_SPIBSC		SPI
RTX_ex2_NOR	RTX Kernel Signal Management を使用して 4 つのスレッドを制御するサンプルアプリケーション	NOR
RTX_ex2_RAM		RAM
RTX_ex2_SPIBSC		SPI
RTX_Traffic_NOR	RTX Kernel Timer Management 及び Signal Flag Management のサンプルアプリケーション	NOR
RTX_Traffic_RAM		RAM
RTX_Traffic_ SPIBSC		SPI
Semaphore_NOR	RTX Kernel Semaphore Management のサンプルアプリケーション	NOR
Semaphore_RAM		RAM
Semaphore_SPIBSC		SPI

5.4 メモリ・マップ

RZ/T1 グループのアドレス空間と RZ/T1 評価ボードのメモリマッピングについては、関連文書一覧の「RZ/T1 グループ初期設定アプリケーションノート」に記載しています。

5.4.1 サンプルプログラムのセクション配置

CMSIS-RTOS RTX サンプルプログラムのセクション配置について次に示します。

(1) EWARM サンプルプログラムのセクション配置

表 5.3 EWARM のセクション配置

領域の名前	内容	タイプ	ロード領域	実行領域
VECTOR_WBLOCK	リセット、例外ベクタテーブル	Code	-	ATCM
USER_PRG_WBLOCK	サンプルアプリケーションプログラム領域（実行用）	Code	-	ATCM
USER_DATA_WBLOCK	サンプルアプリケーションプログラム変数領域（実行用）	Data	-	ATCM
LDR_DATA_WBLOCK 注3	ローダプログラム用変数領域（実行用）	Data	-	BTCM
LDR_PRG_WBLOCK 注3	ローダプログラム領域（実行用）	Code	-	BTCM
ldr_param	ローダ用パラメータ 注1	Data	FLASH 注2	-
LDR_PRG_RBLOCK 注3	ローダプログラム領域（格納用）注1	Code	FLASH 注2	-
LDR_DATA_RBLOCK 注3	ローダプログラム用変数領域（格納用）注1	Data	FLASH 注2	-
VECTOR_RBLOCK	リセット、例外ベクタテーブル領域（格納用）注1	Code	FLASH 注2	-
USER_PRG_RBLOCK	サンプルアプリケーションプログラム領域（格納用）注1	Code	FLASH 注2	-
USER_DATA_RBLOCK	サンプルアプリケーションプログラム用変数領域（格納用）注1	Data	FLASH 注2	-

注1. RAM 実行版では存在しません。

注2. NOR ブートモード版では、NOR フラッシュメモリ、SPI ブートモード版では、シリアルフラッシュメモリとなります。

注3. CMSIS-RTOS RTX サンプルプログラムでは、ローダプログラムで使用する関数（オブジェクト）はローダプログラム用セクション（LDR_PRG_xxx、LDR_DATA_xxx）で指定する必要があります。ローダプログラムで使用する関数を追加する場合は、リンカ設定ファイル（RZ_T1_init_xxx.icf）内でオブジェクトの指定をしてください。

例）ローダプログラムで使用する関数 r_func()（r_func.c）を追加する場合、例えば次頁のように該当セクションで指定します。

```
define block LDR_PRG_RBLOCK with fixed order
                                    {
                                      ro code object startup_Renesas_RZ_T1.o,
                                      ro code object system_Renesas_RZ_T1.o,
                                      ro code object RZ_T1_RSK_Init.o,
                                      ro code object vic.o,
                                      ro code object r_atcm_init.o,
                                      ro code object r_cpg.o,
                                      ro code object r_ram_init.o,
                                      ro code object r_mpc.o,
                                      ro code object bus_init_nor_boot.o,
                                      ro code object r_reset.o,          // 左記行右端に「,」（カンマ）を追加
                                      ro code object r_func.o            // 追加した行
};
```

(2) e2studio サンプルプログラムのセクション配置

ブートモード毎のリンカスクリプト・ファイルのセクション割り当てを、表 5.4～表 5.6 セクション・マップ（RAM ブート設定）に示します。

プログラムからは「__attribute__」で以下のセクション名を指定する事で、配置先を指定する事ができます。

表 5.4 セクション・マップ（SPI ブート設定）

アドレス範囲	領域の名前	内容	ロード領域	実行領域
0x3000004C ～	.flash_contents	ラベル：_mloader_text ラベル：.=.+ (_loader_text_end - _loader_text_start) ・ローダーイメージの格納用 ＜拡張＞ KEEP（グループは参照されなくても削除されない）：チェック	FLASH	-
0x3000604C	.flash_contents_user	ラベル：_mfvector ラベル：.=. + (_fvector_end - _fvector_start) ・ベクターイメージの格納用 ラベル：_mtext ラベル：.=. + (_text_end - _text_start) ・ユーザーコードイメージの格納用 ラベル：_mdata ラベル；.=. + (_data_end - _data_start) ・ユーザーデータイメージの格納用	FLASH	-
0x00000000~	.fvectors	ラベル：_fvectors_start 式：.fvectors ラベル；_fvectors_end ＜拡張＞ セクションマッピングのためにメモリを予約する：チェック 開始アドレス：ラベル-_mfvector KEEP（グループは参照されなくても削除されない）：チェック	-	ATCM
.fvectors から連続	.text	ラベル：_text_start 式：.text 式：.text.* ラベル；_text_end ＜拡張＞ セクションマッピングのためにメモリを予約する：チェック 開始アドレス：ラベル-_mtext	-	ATCM
.text から連続	.init	式：.init キーワード：PROVIDE_HIDDEN (__exidx_start = .) キーワード：PROVIDE_HIDDEN (__exidx_end = .)	-	ATCM
.init から連続	.fini	式：.fini	-	ATCM
.fini から連続	.got	式：.got 式：.got.plt	-	ATCM
.got	.rodata	式；.rodata	-	ATCM

から連続		式：.rodata.* ラベル：_erodata		
.rodata から連続	.eh_frame_hdr	式；.eh_fram_hdr	-	ATCM
eh_frame_hdr から連続	.eh_frame	式：.eh_frame	-	ATCM
.eh_frame から連続	.jcr	式：.jcr	-	ATCM
.jcr から連続	.tors	ラベル；__CTOR_LIST__ キーワード：. = ALIGN(2) ラベル：__ctors 式：.ctors ラベル：__ctors_end ラベル：__CTOR_END__ ラベル：__DTOR_LIST__ ラベル：___dtors 式：.dtors ラベル：___dtors_end ラベル：__DTOR_END__ キーワード：. = ALIGN(2)	-	ATCM
0x00800000～	.loader_param	ラベル：.loader_param ＜拡張＞ セクションマッピングのためにメモリを予約する： チェック 開始アドレス：固定アドレス-0x40000000	-	BTCM
0x30000000～	.loader_param の格納用		FLASH	-
0x00802000～	.loader_text	ラベル：_loader_text_start 式：.loader_text ＜拡張＞ セクションマッピングのためにメモリを予約する： チェック 開始アドレス：ラベル-_mloader_text	-	BTCM
loader_text から連続	.loader_text2	式：.loader_text2 ラベル：. = . + (512 - ((. - _loader_text_start) % 512)) ラベル：_loader_text_end	-	BTCM
0x00700000～	.data	ラベル：_data_start 式：.data ラベル：_data_end ＜拡張＞ セクションマッピングのためにメモリを予約する： チェック 開始アドレス：ラベル-_mdata	-	ACTM
.data から連続	.bss	キーワード：PROVIDE(__bss_start__ = .) ラベル：_bss 式：.bss 式：.bss.** 式：COMMON キーワード：PROVIDE(__bss_end__ = .) ラベル：_ebss ラベル：_end キーワード：PROVIDE(end = .)	-	ACTM
0x00807000～	.STACK		-	BTCM

表 5.5 セクション・マップ（NOR ブート設定）

アドレス範囲	領域の名前	内容	ロード領域	実行領域
0x4000004C ～	.flash_contents	ラベル：_mloader_text ラベル：.=.+ (_loader_text_end - _loader_text_start) ・ローダーイメージの格納用 ＜拡張＞ KEEP（グループは参照されなくても削除されない）：チェック	FLASH	-
0x4000604C~	.flash_contents_user	ラベル：_mfvector ラベル：.=. + (_fvector_end - _fvector_start) ・ベクターイメージの格納用 ラベル：_mtext ラベル：.=. + (_text_end - _text_start) ・ユーザーコードイメージの格納用 ラベル：_mdata ラベル；.=. + (_data_end - _data_start) ・ユーザーデータイメージの格納用	FLASH	-
0x00000000~	.fvectors	ラベル：_fvectors_start 式：.fvectors ラベル；_fvectors_end ＜拡張＞ セクションマッピングのためにメモリを予約する：チェック 開始アドレス：ラベル-_mfvector KEEP（グループは参照されなくても削除されない）：チェック	-	ATCM
.fvectors から連続	.text	ラベル：_text_start 式：.text 式：.text.* ラベル；_text_end ＜拡張＞ セクションマッピングのためにメモリを予約する：チェック 開始アドレス：ラベル-_mtext	-	ATCM
.text から連続	.init	式：.init キーワード：PROVIDE_HIDDEN (__exidx_start = .) キーワード：PROVIDE_HIDDEN (__exidx_end = .)	-	ATCM
.init から連続	.fini	式：.fini	-	ATCM
.fini から連続	.got	式：.got 式：.got.plt	-	ATCM
.got から連続	.rodata	式；.rodata 式：.rodata.* ラベル：_erodata	-	ATCM
.rodata から連続	.eh_frame_hdr	式；.eh_fram_hdr	-	ATCM
eh_frame_hdr から連続	.eh_frame	式：.eh_frame	-	ATCM

.eh_frame から連続	.jcr	式：.jcr	-	ATCM
.jcr から連続	.tors	ラベル；__CTOR_LIST__ キーワード：. = ALIGN(2) ラベル：__ctors 式：.ctors ラベル：__ctors_end ラベル：__CTOR_END__ ラベル：__DTOR_LIST__ ラベル：___dtors 式：.dtors ラベル：___dtors_end ラベル：__DTOR_END__ キーワード：. = ALIGN(2)	-	ATCM
0x00800000～	.loader_param	ラベル：.loader_param ＜拡張＞ セクションマッピングのためにメモリを予約する： チェック 開始アドレス：固定アドレス-0x40000000	-	BTCM
0x04000000～	.loader_param の格納用		FLASH	-
0x00802000～	.loader_text	ラベル：_loader_text_start 式：.loader_text ＜拡張＞ セクションマッピングのためにメモリを予約する： チェック 開始アドレス：ラベル-_mloader_text	-	BTCM
loader_text から連続	.loader_text2	式：.loader_text2 ラベル：. = . + (512 - ((. - _loader_text_start) % 512)) ラベル：_loader_text_end	-	BTCM
0x00700000～	.data	ラベル：_data_start 式：.data ラベル：_data_end ＜拡張＞ セクションマッピングのためにメモリを予約する： チェック 開始アドレス：ラベル-_mdata	-	ACTM
.data から連続	.bss	キーワード：PROVIDE(__bss_start__ = .) ラベル：_bss 式：.bss 式：.bss.** 式：COMMON キーワード：PROVIDE(__bss_end__ = .) ラベル：_ebss ラベル：_end キーワード：PROVIDE(end = .)	-	ACTM
0x00807000～	.STACK		-	BTCM

表 5.6 セクション・マップ（RAM ブート設定）

アドレス範囲	領域の名前	内容	ロード領域	実行領域
0x00000000~	.fvectors	ラベル：_text_start 式：.fvectors ラベル；_fvectors_end ＜拡張＞ KEEP（グループは参照されなくても削除されない）：チェック	ATCM	同左
0x00000040～	.text	ラベル：_text_start 式：.text 式：.text.*	ATCM	同左
.text から連続	.init	式：.init キーワード：PROVIDE_HIDDEN (__exidx_start = .) キーワード：PROVIDE_HIDDEN (__exidx_end = .)	ATCM	同左
.init から連続	.fini	式：.fini	ATCM	同左
.fini から連続	.got	式：.got 式：.got.plt	ATCM	同左
.got から連続	.rodata	式；.rodata 式：.rodata.* ラベル：_erodata	ATCM	同左
.rodata から連続	.eh_frame_hdr	式；.eh_fram_hdr	ATCM	同左
eh_frame_hdr から連続	.eh_frame	式：.eh_frame	ATCM	同左
.eh_frame から連続	.jcr	式：.jcr	ATCM	同左
.jcr から連続	.tors	ラベル；__CTOR_LIST__ キーワード：. = ALIGN(2) ラベル：__ctors 式：.ctors ラベル：__ctors_end ラベル：__CTOR_END__ ラベル：__DTOR_LIST__ ラベル：___dtors 式：.dtors ラベル：___dtors_end ラベル：__DTOR_END__ キーワード：. = ALIGN(2)	ATCM	同左
.tors から連続	.loader_text	ラベル：_loader_text_start 式：.loader_text ＜拡張＞ セクションマッピングのためにメモリを予約する：チェック 開始アドレス：ラベル-_mloader_text	ATCM	同左
loader_text から連続	.loader_text2	式：.loader_text2 ラベル：. = . + (512 - ((. - _loader_text_start) % 512)) ラベル：_loader_text_end	ATCM	同左
loader_text2 から連続	.data	ラベル：_data_start 式：.data	ATCM	同左

		ラベル：_data_end ＜拡張＞ セクションマッピングのためにメモリを予約する： チェック 開始アドレス：ラベル-_mdata		
.data から連続	.bss	キーワード：PROVIDE(__bss_start__ = .) ラベル：_bss 式：.bss 式：.bss.** 式：COMMON キーワード：PROVIDE(__bss_end__ = .) ラベル：_ebss ラベル：_end キーワード：PROVIDE(end = .)	-	ATCM
.bss から連続	.SVC_TABLE		-	ATCM
0x00807000～	.STACK		-	BTCM

(3) DS-5 サンプルプログラムのセクション配置

ブートモード毎のリンカスクリプト・ファイルのセクション割り当てを、表 5.7～表 5.8 に示します。

プログラムからは「#pragma arm section code」で以下のセクション名を指定することで、配置先を指定する事ができます。

表 5.7 DS-5 のセクション配置（SPI/NOR ブート設定）

領域の名前	内容	タイプ	ロード領域	実行領域
CONST_LOADER_TABLE	ローダ用パラメータ注1	Data	FLASH 注1	FLASH
LOADER_RESET_HANDLER	リセットハンドラ用領域	Code	FLASH 注1	FLASH
LOADER_CODE 注2	ローダ用のコード領域	Code	FLASH 注1	BTCM
LOADER_CONST 注2	ローダ用の読み出し専用データ領域	Data	FLASH 注1	BTCM
LOADER_DATA 注2	ローダ用の読み書きデータ領域	Data	FLASH 注1	BTCM
LOADER_BSS 注2	ローダ用の BSS 領域	Zero	FLASH 注1	BTCM
LOADER_IN_ROOT	ARM ライブラリ用のルート領域	Code Ven Data	FLASH 注1	FLASH
INIT	リセット、例外ベクタテーブル領域（格納用）	Code	FLASH 注1	ATCM
CODE	プログラムコード領域	Code	FLASH 注1	ATCM
DATA	読み出し専用、読み書き用のデータ領域	Data	FLASH 注1	ATCM

注1. NOR ブートモード版では、NOR フラッシュメモリ、SPI ブートモード版では、シリアルフラッシュメモリとなります。

注2. CMSIS-RTOS RTX サンプルプログラムでは、ローダプログラムで使用する関数（オブジェクト）はローダプログラム用セクション（LOADER_CODE、LOADER_CONST、LOADER_DATA、LOADER_BSS）で指定する必要があります。ローダプログラムで使用する関数を追加する場合は、以下の例のようにプログラムに追加して下さい。

例）R_ATCM_WaitSet()関数を該当セクションに追加する。

```
#pragma arm section code = "LOADER_CODE"
#pragma arm section rodata = "LOADER_CONST"
#pragma arm section rwdata = "LOADER_DATA"
#pragma arm section zidata = "LOADER_BSS"
void R_ATCM_WaitSet()
{
}
```

表 5.8 DS-5 のセクション配置（RAM ブート設定）

領域の名前	内容	タイプ	ロード領域	実行領域
VECTORS	リセット、例外ベクタテーブル領域（格納用）	Code	ATCM	同左
RO_CODE	プログラムコード領域	Code	ATCM	同左
RO_DATA	読み出し専用のデータ領域	Data	ATCM	同左
LOADER_CODE 注1	プログラムコード領域	Code	ATCM	同左
LOADER_CONST 注1	読み出し専用のデータ領域	Data	ATCM	同左
LOADER_DATA 注1	読み書き用のデータ領域	Data	ATCM	同左
LOADER_BSS 注1	BSS 領域	Zero	ATCM	同左
RW_DATA	読み書き用のデータ領域	Data	ATCM	同左
ZI_DATA	BSS 領域	Zero	-	ATCM
STACK	STACK 領域	Zero	-	BTCM
INIT	dummy 領域（ビルドエラー対策）	-	ビルド対策のため領域なし（サイズゼロ）	

注1.「LOADER～」セクションは RAM ブートモードでは不要であるが、SPI ブートモード、NOR ブートモード用の共通ソースである為、セクションを定義している。

5.4.2 MPU の設定

MPU の設定は、RZ/T1 グループ初期設定アプリケーションノートを参照してください。

5.4.3 例外ベクタテーブル

RZ/T1 には 7 種類の例外処理（リセット、未定義命令、ソフトウェア割り込み、プリフェッチアボート、データアボート、IRQ、FIQ）があり、0000 0000H 番地から 32 バイトの領域（0000 0000H 番地～0000 001F 番地）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。

以下に、CMSIS-RTOS RTX 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。

表 5.9 RZ/T1 例外処理ベクタテーブル

例外	ハンドラアドレス	備考
RESET 例外	0000 0000H	ローダプログラム先頭へ分岐
未定義命令例外	0000 0004H	CMSIS-RTOS RTX の未定義命令例外ハンドラ
ソフトウェア例外	0000 0008H	CMSIS-RTOS RTX の SVC 例外ハンドラ
プリフェッチアボート例外	0000 000CH	CMSIS-RTOS RTX のプリフェッチアボートハンドラ
データアボート例外	0000 0010H	CMSIS-RTOS RTX のデータアボードハンドラ
Reserved	0000 0014H	NOP
IRQ 例外	0000 0018H	CMSIS-RTOS RTX の IRQ 例外ハンドラ（未使用）
FIQ 例外	0000 001CH	ダミールーチン（無限ループ）

割り込みについて、RZ/T1 では、Cortex-R4F に対する割り込み制御としてベクタ割り込みコントローラ（VIC）を採用しており、VIC ドライバが割り込みを管理しています。

使用する割り込みにつきましては、ベクタ番号と割り込みサービスルーチンを指定して InterruptHandlerRegister 関数をコールし、発生した割り込みに対する割り込みサービスルーチンを登録してください。

5.4.4 使用メモリサイズ

(1) EWARM サンプルプログラムの使用メモリ

以下に参考として、EWARM の RTX_ex1 サンプルプログラムが使用するメモリサイズを記載します。

表 5.10 EWARM の使用メモリサイズ（RTX_ex1_NOR / RTX_ex1_SPIBSC）

ラベル名	概要	メモリサイズ（byte）（概算）	
		NOR	SPIBSC
VECTOR_WBLOCK	リセット、例外ベクタテーブル（実行用）	60	60
USER_PRG_WBLOCK	サンプルアプリケーションプログラム領域（実行用）	16K	11K
.noinit	スタック領域	1280	1280
USER_DATA_WBLOCK	サンプルアプリケーションプログラム変数領域（実行用）	16	16
USER_DATA_ZBLOCK	未初期化データ領域	59K	59K
LDR_DATA_WBLOCK	ローダプログラム用変数領域（実行用）	16	16
LDR_PRG_WBLOCK	ローダプログラム領域（実行用）	4K	16K
ldr_param	ローダ用パラメータ	76	76
LDR_PRG_RBLOCK	ローダプログラム領域（格納用）	4K	16K
LDR_DATA_RBLOCK	ローダプログラム用変数領域（格納用）	16	16
VECTOR_RBLOCK	リセット、例外ベクタテーブル（格納用）	60	60
USER_PRG_RBLOCK	サンプルアプリケーションプログラム領域（格納用）	16K	11K
USER_DATA_RBLOCK	サンプルアプリケーションプログラム変数領域（格納用）	16	16

※各サンプルにより値が異なります。

表 5.11 EWARM の使用メモリサイズ（RTX_ex1_RAM）

ラベル名	概要	メモリサイズ（byte）（概算）
intvec	リセット、例外ベクタテーブル	60
ROM_region	プログラム、定数領域	19K
RAM_region	未初期化データ領域、データ領域	60K

※各サンプルにより値が異なります。

(2) e2studio サンプルプログラムの使用メモリサイズ

以下に参考として、e2studio の RTX_e1 サンプルプログラムが使用するメモリサイズを記載します。

表 5.12 e2studio の使用メモリサイズ（RTX_ex1_RAM / RTX_ex1_NOR / RTX_ex1_SPIBSC）

ラベル名	概要	メモリサイズ（byte）（概算）		
		NOR	SPIBSC	RAM
.flash_contents	ローダ前処理部コード領域（格納用）	6K	19K	-
.flash_content_user	サンプルアプリケーションプログラム領域、サンプルアプリケーションプデータ領域、サンプルアプリケーションプ定数領域、リセット・例外ベクタテーブル（格納用）	48K	48K	-
.fvectors	リセット、例外ベクタテーブル（実行用）	60	60	60
.text	サンプルアプリケーションプログラム領域（実行用）	48K	48K	49K
.rodata	サンプルアプリケーション定数領域（実行用）	168	168	168
.tors		0	0	0
.loader_param	ローダ用パラメータ	76	76	-
.loader_text	ローダ前処理部コード領域（実行用）	1104	1104	-
.loader_text2	ローダ後処理部コード領域（実行用）	5K	17K	2K
.data	サンプルアプリケーションデータ領域（実行用）	32	32	32
.bss	未初期化データ領域	59K	59K	59K
.SVC_TABLE	SVC テーブル	8	8	8
.STACK	各モードのスタック領域	1280	1280	1280

※各サンプルにより値が異なります。

(3) DS-5 サンプルプログラムの使用メモリサイズ

以下に参考として、DS-5 の RTX_ex1 サンプルプログラムが使用するメモリサイズを記載します。

表 5.13 DS-5 の使用メモリサイズ（RTX_ex1_NOR / RTX_ex1_SPIBSC）

ラベル名	概要	メモリサイズ（byte）（概算）	
		NOR	SPIBSC
CONST_LOADER_TABLE	ローダ用パラメータ[注1]	76	76
LOADER_RESET_HANDLER	リセットハンドラ用領域	28	28
LOADER_CODE	ローダ用のコード領域	236	236
LOADER_CONST	ローダ用の読み出し専用データ領域	0	0
LOADER_DATA	ローダ用の読み書きデータ領域	20	20
LOADER_BSS	ローダ用の BSS 領域	1204	1308
LOADER_IN_ROOT	ARM ライブラリ用のルート領域	236	236
INIT	リセット、例外ベクタテーブル領域（格納用）	60	60
CODE	プログラムコード領域	25K	25K
DATA	読み出し専用、読み書き用のデータ領域	68K	68K
SVC_TABLE	SVC テーブル	8	8
STACK	各モードのスタック領域	1280	1280

※各サンプルにより値が異なります。

表 5.14 DS-5 の使用メモリサイズ（RTX_ex1_RAM）

ラベル名	概要	メモリサイズ（byte）（概算）
		RAM
LOADER_CODE	ローダ用のコード領域	1260
LOADER_CONST	ローダ用の読み出し専用データ領域	0
LOADER_DATA	ローダ用の読み書きデータ領域	12
LOADER_BSS	ローダ用の BSS 領域	0
VECTORS	リセット、例外ベクタテーブル領域（格納用）	60
RO_CODE	プログラムコード領域	22K
RO_DATA	読み出し専用データ領域	108
RW_DATA	初期値あり読み書き用のデータ領域	128
ZI_DATA	初期値なし読み書き用のデータ領域	69K
SVC_TABLE	SVC テーブル	8
STACK	各モードのスタック領域	1280

※各サンプルにより値が異なります。

5.5 フォルダ・ファイル構成

TOP フォルダの構成は下記の通りとなります。[ ] はフォルダを示します。各章で詳細を説明します。

```
[RZ_T1_RTX]    TOP フォルダ
    │
    ├──[IAR]          IAR プロジェクト管理フォルダ（5.5.1 章参照）
    │
    ├──[KPITGCC]      e2studio プロジェクト管理フォルダ（5.5.2 章参照）
    │
    ├──[ARMCC]        DS-5 プロジェクト管理フォルダ（5.5.3 章参照）
    │
    ├──[Include]      ターゲット CPU 別ハードウェア制御関数ヘッダ（5.5.1 章参照）
    │
    └──[RTOS]         ソース管理フォルダ（DS-5/IAR/e2stuido 共通）（5.5.1 章参照）
```

5.5.1 EWARM サンプルアプリケーションのフォルダ・ファイル構成

(1) フォルダ構成

EWARM サンプルアプリケーションのフォルダ構成は、[RZ_T1_RTX]の TOP フォルダから、[IAR]、[Include]、[RTOS]の 3 つのフォルダで構成されています。

図 5.4 に[IAR]フォルダについて説明します。[IAR]フォルダ以下には、プロジェクト管理ファイルのみ存在し、全て EWARM のそれぞれのプロジェクト管理フォルダとなります（ソースは保管されていません）。

なお下記の[…_***]のフォルダの部分については、[…_RAM] / […_NOR] / […_SPIBSC]と 3 つのフォルダが存在し、RAM ブート用 / NOR ブート用 / SPI ブート用と、フォルダが 3 つに分かれています。

```
[RZ_T1_RTX]          TOP フォルダ
  ┃
  ┣━[IAR]            以下、全て IAR 用プロジェクト管理ファイルフォルダ
  :   ┃
      ┣━[Bootloader]                    それぞれのブートローダー
      ┃   ┃
      ┃   ┣━[Bootloader_NorFlash]       NOR ブートのブートローダー
      ┃   ┃
      ┃   ┗━[Bootloader_SerialFlash]    SPI ブートのブートローダー
      ┃
      ┣━[Renesas_example]                サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[Drivers]                    それぞれのドライバ
      ┃   ┃   ┃
      ┃   ┃   ┣━[RTX_ADC_drv]           ADC ドライバ
      ┃   ┃   ┃
      ┃   ┃   ┣━[RTX_CMT_drv]           CMT ドライバ
      ┃   ┃   ┃
      ┃   ┃   ┣━[RTX_ELC_drv]           ELC ドライバ
      ┃   ┃   ┃
      ┃   ┃   ┗━[RTX_MTU3_drv]          MTU3 ドライバ
      ┃   ┃
      ┃   ┣━[FPU_ex1_***]                FPU_ex1 サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[Mail_***]                   Mail サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[Message_***]                Message サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[RTX_CMT_ex1_***]            RTX_CMT_ex1 サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[RTX_ex1_***]                RTX_ex1 サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[RTX_ex2_***]                RTX_ex2 サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[RTX_MTU3_ex1_***]           RTX_MTU3_ex1 サンプルアプリケーション
      ┃   ┃
      ┃   ┣━[RTX_Traffic_***]            RTX Traffic サンプルアプリケーション
      ┃   ┃
      ┃   ┗━[Semaphore_***]              Semaphore サンプルアプリケーション
      ┃
      ┗━[RTOS]
          ┗━[RTX]
              ┗━[RTX_CR4_Library]        CMSIS-RTOS RTX
```

図 5.4 [IAR]フォルダ構成

続いて、図 5.5 に[include]、[RTOS]フォルダについて説明します。この 2 つのフォルダはプログラムが保存されています。以下に説明します。

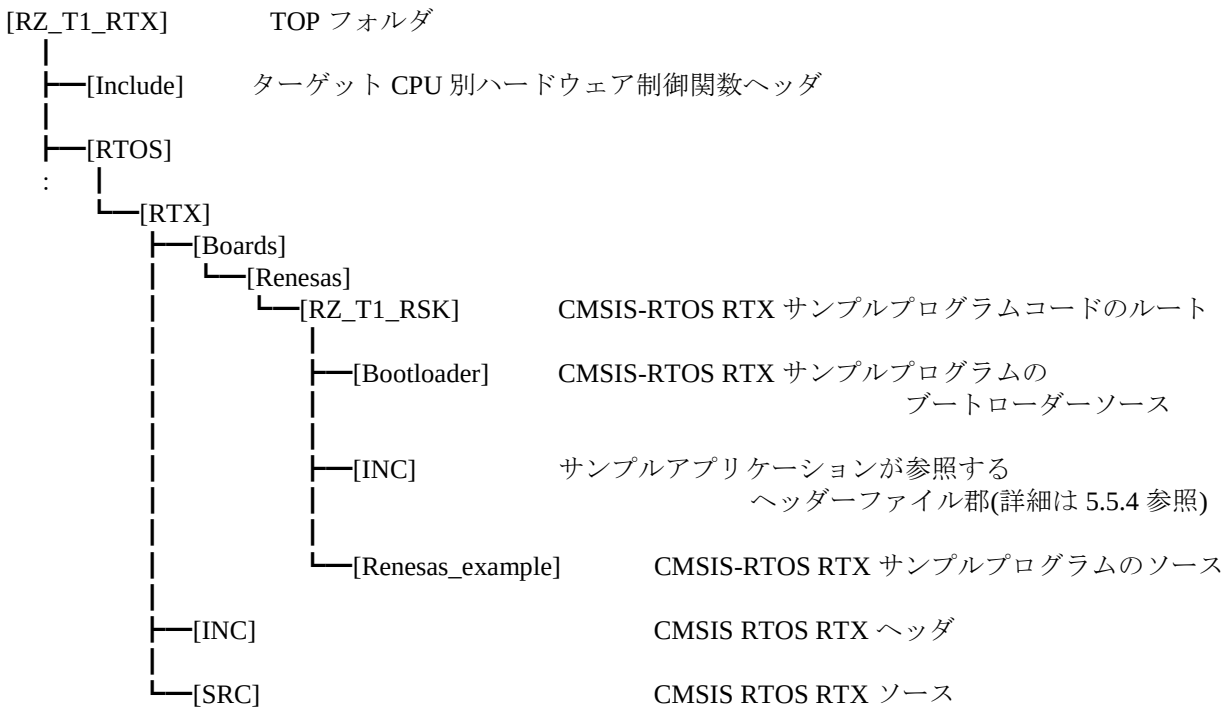


図 5.5 [Include],[RTOS]フォルダ

(2) ファイル構成

CMSIS-RTOS RTX サンプルプログラムがリンクするライブラリの一覧を以下に記します。

表 5.15 サンプルアプリケーションがリンクするライブラリ

ファイル名	概要	備考
RTX_CR4_Library.a	CMSIS-RTOS RTX ライブラリ	Cortex-R4F 用リトルエンディンのライブラリ
Bootloader_NorFlash.a	NOR ブート用 ライブラリ	NOR ブートモード（NOR フラッシュ）により、サンプルアプリケーションを起動する為のライブラリ _NOR プロジェクトでリンクします。
Bootloader_SerialFlash.a	SPI ブート用 ライブラリ	SPI ブートモード（シリアルフラッシュ）により、サンプルアプリケーションを起動する為のライブラリ _SPIBSC プロジェクトでリンクします。
RTX_ADC_drv.a	RTX_ADC ドライバ ライブラリ	RTX_ADC ドライバのライブラリ。RTX_CMT_ex1 サンプルプロジェクトでリンクします。
RTX_CMT_drv.a	RTX_CMT ドライバ ライブラリ	RTX_CMT のドライバライブラリ。 RTX_CMT_ex1 サンプルプロジェクトでリンクします。
RTX_ELC_drv.a	RTX_ELC ドライバ ライブラリ	RTX_ELC ドライバのライブラリ。 RTX_CMT_ex1 サンプルプロジェクトでリンクします。
RTX_MTU3_drv.a	RTX_MTU3 ドライバ ライブラリ	RTX_MTU3 ドライバのライブラリ。 RTX_MTU3_ex1 サンプルプロジェクトでリンクします。

CMSIS-RTOS RTX サンプルプログラム共通のソースファイル一覧を以下に記します。

表 5.16　サンプルアプリケーション共通のソースファイル一覧

ファイル名	概要	備考
close.c	ファイルを閉じます	IAR C/C++ コンパイラ ファイル I/O
lseek.c	ファイル位置インジケータを設定します	IAR C/C++ コンパイラ ファイル I/O
read.c	文字バッファをリードします	IAR C/C++ コンパイラ ファイル I/O
write.c	文字バッファをライトします	IAR C/C++ コンパイラ ファイル I/O
fpuenable.s	浮動小数点ユニット（FPU）設定ソース	
icu_vic.s	Cortex-R4F ベクタ割り込みコントローラ（VIC）制御ソース	
startup_Renesas_RZ_T1.s	例外処理ベクタ(リセットハンドラの実体)	
vector.s	ベクタ・アドレスの実体	
RAM¥system_Renesas_RZ_T1.c	ハードウェア初期化ソース	

上記ファイルは、以下のパスの何れにも含まれています

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example¥(*Dir1)¥IAR

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example¥Drivers¥drv_example¥(*Dir2)¥IAR

*Dir1：FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore

*Dir2：RTX_CMT_ex1 / RTX_MTU3_ex1

5.5.2 e2studio サンプルアプリケーションのフォルダ・ファイル構成

(1) フォルダ構成

e2studio サンプルアプリケーションのフォルダ構成は、[RZ_T1_RTX]の TOP フォルダから、[KPITGCC]、[Include]、[RTOS]の 3 つのフォルダで構成されています。

各々のフォルダ構成は、5.5.1 章の図 5.4、図 5.5 と同じです。IAR を KPITGCC、EWARM を e2studio と読み替えて参照ください。

(2) ファイル構成

CMSIS-RTOS RTX サンプルプログラムがリンクするライブラリについては、表 5.15 と同じです。そちらを参照ください。

CMSIS-RTOS RTX サンプルプログラム共通のソースファイル一覧を以下に記します。

表 5.17 サンプルアプリケーション共通のソースファイル一覧

ファイル名	概要	備考
syscalls.c	_read、_write、_exit 関数	GCC コンパイラファイル I/O
exclusion_func.c	排他制御関数	GCC コンパイラファイル I/O
siorw.c	シリアルポート入出力関数	GCC コンパイラファイル I/O
newheap.c	ヒープ終端アドレス取得関数	GCC コンパイラファイル I/O
fpuenable.S	浮動小数点ユニット（FPU）設定ソース	
icu_vic.S	Cortex-R4F ベクタ割り込みコントローラ（VIC） 制御ソース	
startup_Renesas_RZ_T1.S	例外処理ベクタ (リセットハンドラの実体)	
vector.S	ベクタ・アドレスの実体	
RAM¥system_Renesas_RZ_T1.c	ハードウェア初期化ソース	

上記ファイルは、以下のパスの何れにも含まれています

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example¥(*Dir1)¥GCC

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example¥Drivers¥drv_example¥(*Dir2)¥GCC

*Dir1：FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore

*Dir2：RTX_CMT_ex1 / RTX_MTU3_ex1

5.5.3 DS-5 サンプルアプリケーションのフォルダ・ファイル構成

(1) フォルダ構成

DS-5 サンプルアプリケーションのフォルダ構成は、[RZ_T1_RTX]の TOP フォルダから、[ARMCC]、[Include]、[RTOS]の 3 つのフォルダで構成されています。

各々のフォルダ構成は、5.5.1 章の図 5.4、図 5.5 と同じです。IAR を ARMCC、EWARM を DS-5 と読み替えて、そちらをご参照ください。

注意）以下の２つのフォルダと、それ以下の下位フォルダの構成については、同梱されている fmtool がこのフォルダ構成でのみ動作する為、変更しないで下さい。

¥ARMCC¥Bootloader¥Bootloader_NorFlash

¥ ARMCC¥Bootloader¥Bootloader_SerialFlash

なお、フラッシュ書き込み用ファイルについて以下に補足説明します。

表 5.18 NOR ブートでのフラッシュ書き込み用ファイル

ファイル名	概要	備考
¥ARMCC¥Bootloader¥Bootloader_NorFlash¥fmtool	サンプルの fmtool NOR フラッシュ書き込みツール	本フォルダ、ファイルは、同梱されている fmtool がこのフォルダ構成でのみ動作する為、変更しないで下さい。
¥ARMCC¥Bootloader¥Bootloader_NorFlash¥script_nor	サンプルダウンロード用スクリプトファイル	

表 5.19 SPI ブートでのフラッシュ書き込み用ファイル

ファイル名	概要	備考
¥ ARMCC¥Bootloader¥Bootloader_SerialFlash¥fmtool	サンプルの fmtool シリアルフラッシュ書き込みツール	本フォルダ、ファイルは、同梱されている fmtool がこのフォルダ構成でのみ動作する為、変更しないで下さい。
¥ ARMCC¥Bootloader¥Bootloader_SerialFlash¥script_nor	サンプルダウンロード用スクリプトファイル	

(2) ファイル構成

CMSIS-RTOS RTX サンプルプログラムがリンクするライブラリについては、表 5.15 と同じです。そちらを参照ください。

CMSIS-RTOS RTX サンプルプログラム共通のソースファイル一覧を以下に記します。

表 5.20 サンプルアプリケーション共通のソースファイル一覧

ファイル名	概要	備考
fpuenable.s	浮動小数点ユニット（FPU）設定ソース	ARM C/C++ コンパイラ
		ファイル I/O
read.c	文字バッファリード	ARM C/C++ コンパイラ
write.c	文字バッファライト	ARM C/C++ コンパイラ
retarget.c	ARM C ライブラリ・ターゲット 依存 I/O サポート関数	ARM C/C++ コンパイラ
sio_char.h	文字関数関連のヘッダ	ARM C/C++ コンパイラ
icu_vic.s	Cortex-R4F ベクタ割り込みコントローラ（VIC） 制御ソース	
startup_Renesas_RZ_T1.s	例外処理ベクタ(リセットハンドラの実体)	
vector.s	ベクタ・アドレスの実体	
RAM¥system_Renesas_RZ_T1.c	ハードウェア初期化ソース	

上記ファイルは、以下のパスの何れにも含まれています

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example¥(*Dir1)¥ARM

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example¥Drivers¥drv_example¥(*Dir2)¥ARM

*Dir1：FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore

*Dir2：RTX_CMT_ex1 / RTX_MTU3_ex1

5.5.4 サンプルアプリケーション共通のフォルダ構成

CMSIS-RTOS RTX サンプルプログラムが参照するヘッダーファイルの一覧を以下に記します。

表 5.21 サンプルアプリケーションが参照するヘッダーファイル一覧

ファイル名	概要	備考
iodefine.h	RZ/T1 レジスタ定義ヘッダ (全体インクルード用)	
r_typedefs.h	基本データ型ヘッダ	
Renesas_RZ_T1.h	RZ_T1CPU 依存定義	
RZ_T1_RSK_Init.h	RZ/T1 評価ボード初期化ヘッダ	
system_Renesas_RZ_T1.h	CPU 依存部初期化関数ヘッダ	
vic.h	割り込みコントローラ設定 API ヘッダ	
r_atcm_init.h	ATCM ウェイト設定 API ヘッダ	
r_bsc.h	バス・コントローラ設定 API ヘッダ	
r_cpg.h	クロック・パルス発信器(CPG)設定 API ヘッダ	
r_ecm.h	エラーコントロールモジュール（ECM）API ヘッダ	
r_icu_init.h	割り込みコントローラ初期化及び API ヘッダ	
r_mpc.h	マルチファンクションピンコントローラ（MPC）設定 API	
r_port.h	I/O ポート設定 API ヘッダ	
r_ram_init.h	内蔵拡張 SRAM(領域 1，領域 2)設定 API ヘッダ	
r_reset.h	リセット API ヘッダ	
r_spibsc_flash_api.h	シリアルフラッシュ設定 API ヘッダ	
r_spibsc_ioset_api.h	シリアルフラッシュ IO 設定 API ヘッダ	
r_system.h	ハードウェア初期化ヘッダ	

上記ファイルは以下のディレクトリに置かれています。各プロジェクト共通で使用されます。

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥INC

CMSIS-RTOS RTX サンプルプログラム共通のソースファイル一覧を以下に記します。

表 5.22　サンプルアプリケーション共通のソースファイル一覧

ファイル名	概要	備考
low_level_init.c	低レベル初期化関数ソース	
loader_reset_handler.s	NOR/SPI ブート用のリセットハンドラ。	RZ/T1、RZ/T1 評価ボード
r_atcm_init.c	ATCM ウェイト設定ソース	RZ/T1、RZ/T1 評価ボード
r_cpg.c	クロック・パルス発信器（CPG）設定ソース	RZ/T1、RZ/T1 評価ボード
r_ecm.c	エラーコントロールモジュール設定ソース	RZ/T1、RZ/T1 評価ボード
r_mpc.c	マルチピンファンクション設定ソース	RZ/T1、RZ/T1 評価ボード
r_reset.c	リセット制御ソース	RZ/T1、RZ/T1 評価ボード
RTX_Conf_CM.c	CMSIS-RTOS RTX コンフィギュレーション	RZ/T1、RZ/T1 評価ボード
RZ_T1_RSK_Init.c	RZ/T1 評価ボード初期化ソース	RZ/T1、RZ/T1 評価ボード
Serial.h	デバッグ・シリアル入出力ヘッダ	RZ/T1、RZ/T1 評価ボード
Serial.c	デバッグ・シリアル入出力ソース	RZ/T1、RZ/T1 評価ボード
vic.c	Cortex-R4F ベクタ割り込みコントローラ（VIC）設定 API ソース	RZ/T1（Cortex-R4F）、RZ/T1 評価ボード

上記ファイルは、以下のパスの何れにも含まれています

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example¥(*Dir1)

*Dir1：FPU_ex1 / Mail / Message / RTX_ex1 / RTX_ex2 / RTX_Traffic / Semaphore

CMSIS-RTOS RTX サンプルプログラム本体の共通のファイル一覧を以下に記します。

表 5.23 サンプルアプリケーション本体のファイル一覧

ファイル名	概要
FPU_ex1¥FPU_ex1.c	FPU ex1 サンプルアプリケーションの本体
Mail¥Mail.c	Mail サンプルアプリケーションの本体
Message¥Message.c	Message サンプルアプリケーションの本体
RTX_ex1¥RTX_ex1.c	RTX ex1 サンプルアプリケーションの本体
RTX_ex2¥RTX_ex2.c	RTX ex2 サンプルアプリケーションの本体
Semaphore¥Semaphore.c	Semaphore サンプルアプリケーションの本体
Drivers¥drv_example¥ RTX_CMT_ex1¥CMT_ex1.c	RTX_CMT ex1 サンプルアプリケーションの本体
Drivers¥drv_example¥ RTX_MTU3_ex1 ¥MTU_ex1.c	RTX_MTU3 ex1 サンプルアプリケーションの本体

上記ディレクトリのパスは、以下となります。

RZ_T1_RTX¥RTOS¥RTX¥Boards¥Renesas¥RZ_T1_RSK¥Renesas_example

5.6 基本データ型

CMSIS-RTOS RTX サンプルプログラムで使用している基本データ型を以下に示します。

本 CMSIS-RTOS RTX サンプルプログラムでは、基本データ型には C 言語の型を直接使用せず、以下の型のみを使用します。

表 5.24 基本データ型

typedef char	char_t
typedef int	int_t
typedef unsigned int	bool_t
typedef signed char	int8_t
typedef unsigned char	uint8_t
typedef signed short	int16_t
typedef unsigned short	uint16_t
typedef signed int	int32_t
typedef unsigned int	uint32_t
typedef signed long long	int64_t
typedef unsigned long long	uint64_t

5.7 各種値定義(コンフィギュレーション・マクロ)

コンフィギュレーション用定数マクロを以下に示します。

表 5.25 コンフィギュレーション用定数マクロ一覧

マクロ名	値	内容	定義ファイル
__FPU_PRESENT	値なしで定義	FPU の使用 定義時：使用、未定義時：未使用	プロジェクト設定
OS_TASKCNT	50	CMSIS-RTOS RTX の最大スレッド生成数 ※システム上限値は 250	RTX_Conf_CM.c
OS_STKSIZE	256	CMSIS-RTOS RTX スレッドのデフォルト・スタック・サイズ(単位：4byte)	RTX_Conf_CM.c
OS_MAINSTKSIZE	256	CMSIS-RTOS RTX 初期スレッドのスタック・サイズ(単位：4byte)	RTX_Conf_CM.c
OS_PRIVCNT	0	CMSIS-RTOS RTX のユーザー定義スタック使用スレッド数	RTX_Conf_CM.c
OS_PRIVSTKSIZE	500	CMSIS-RTOS RTX のユーザー定義スタック合計サイズ	RTX_Conf_CM.c
OS_STKCHECK	1	CMSIS-RTOS RTX のスタック・オーバフロー・チェック機能 1：有効、0：無効	RTX_Conf_CM.c
OS_RUNPRIV	1	CMSIS-RTOS RTX のスレッド実行時の CPU 動作モード 1：特権モード、0：非特権モード	RTX_Conf_CM.c
OS_SYSTICK	0	CMSIS-RTOS RTX のシステム・タイマ選択 0：周辺タイマ、1： CPU コア内蔵タイマ ※RZ/T1 は CPU コア内蔵タイマを持たないので 0 固定。	RTX_Conf_CM.c
OS_CLOCK	2343750 (75000000u/32u)	CMSIS-RTOS RTX のシステム・タイマ入力クロック(単位：Hz) ※OS_SYSTICK が 1 の時は参照されない	RTX_Conf_CM.c
OS_TICK	1000	CMSIS-RTOS RTX のシステム・タイマ周期(単位：μsec) ※設定例では 1ms となる	RTX_Conf_CM.c
OS_ROBIN	0	CMSIS-RTOS RTX のラウンドロビン機能 1：有効、0：無効	RTX_Conf_CM.c
OS_ROBINTOUT	5	CMSIS-RTOS RTX のラウンドロビン・タイムアウト(単位：システム・タイマ・ティック)	RTX_Conf_CM.c
OS_TIMERS	1	CMSIS-RTOS RTX のユーザー・タイマ機能 1：有効、0：無効 ※有効にすると「ユーザー・タイマ・コールバック関数実行スレッド」が生成される	RTX_Conf_CM.c
OS_TIMERPRIO	5	CMSIS-RTOS RTX のユーザー･タイマ･コールバック関数実行スレッドの優先度(最小 1～最大 6)	RTX_Conf_CM.c
OS_TIMERSTKSZ	256	CMSIS-RTOS RTX のユーザー･タイマ･コールバック関数実行スレッドのスタック・サイズ(単位：4byte)	RTX_Conf_CM.c
OS_TIMERCBQS	5	CMSIS-RTOS RTX のユーザー･タイマ･コールバック関数実行スレッドへのタイムアウト通知メッセー	RTX_Conf_CM.c

		ジキューイング可能数	
OS_FIFOSZ	16	CMSIS-RTOS RTX の isr_*系 API イベント用 FIFO キュー要素数	RTX_Conf_CM.c
OS_MUTEXCNT	8	スタンダードＣライブラリマルチスレッド対応排他用ミューテックス定義数	RTX_Conf_CM.c
OS_TRV	右記に記載	CPU コア内蔵タイマのコンペア・レジスタ設定値 ※OS_SYSTICK が 1 の時は参照されない 以下で定義 ((uint32_t)(((double)OS_CLOCK*(double)OS_TICK)/1E6)-1)	RTX_Conf_CM.c

5.8 エラーコード

CMSIS-RTOS RTX は、API のエラーコードとして以下の値を返します。

表 5.26 CMSIS-RTOS RTX エラーコード一覧

列挙子名	値	用例	定義ファイル
osOK	0x00	API 正常終了	cmsis_os.h
osEventSignal	0x08	シグナル・イベント発生	cmsis_os.h
osEventMessage	0x10	メッセージ・イベント発生	cmsis_os.h
osEventMail	0x20	メール・イベント発生	cmsis_os.h
osEventTimeout	0x40	タイムアウト発生	cmsis_os.h
osErrorParameter	0x80	パラメータ・エラー	cmsis_os.h
osErrorResource	0x81	リソース獲得失敗	cmsis_os.h
osErrorTimeoutResource	0xC1	リソース獲得失敗（タイムアウト）	cmsis_os.h
osErrorISR	0x82	ISR 内で利用不可能な API を実行	cmsis_os.h
osErrorISRRecursive	0x83	ISR から同じオブジェクトに対して複数回 API を実行	cmsis_os.h
osErrorPriority	0x84	システムが優先度を決定できない 不正な優先度を指定	cmsis_os.h
osErrorNoMemory	0x85	メモリアロケーション失敗	cmsis_os.h
osErrorValue	0x86	パラメータ設定値が設定可能な値の範囲外	cmsis_os.h
osErrorOS	0xFF	未定義 CMSIS-RTOS RTX エラー 他のエラーに該当しない全てのエラー	cmsis_os.h

5.9 関数一覧

以下に、RZ/T1 評価ボード依存の処理を含む内部関数を示します。

表 5.27 RZ/T1 評価ボード依存内部関数

関数名	概要	実装ファイル
Reset_Handler	ICCARM/RenesasGCC 用のリセットベクタ割り当て用のアセンブラ関数です。SystemInit()の実行後、以下の関数にジャンプします。 **・GCC コンパイラ：cstartup()** **・IAR コンパイラ：__cmain()**	startup_Renesas_RZ_T1.s
user_init	ARMCC 用のリセットベクタ割り当て用のアセンブラ関数です。SystemInit()の実行後、__main 関数にジャンプします。	startup_Renesas_RZ_T1.s
cpu_init	ARMCC 用、かつ NOR/SPI ブート用の関数です。クロック、バスの初期化などを行い、user_init 関数にジャンプします。	loader_reset_handler.s
Undef_Handler	未定義命令例外ハンドラ	startup_Renesas_RZ_T1.s
PAbt_Handler	ソフトウェア例外ハンドラ	startup_Renesas_RZ_T1.s
DAbt_Handler	データアボート例外ハンドラ	startup_Renesas_RZ_T1.s
CUndefHandler	データアボート例外ハンドラ	system_Renesas_RZ_T1.c
CPAbtHandler	プリフェッチアボート例外ハンドラ	system_Renesas_RZ_T1.c
CDAbtHandler	データアボート例外 C 言語ハンドラ	system_Renesas_RZ_T1.c
__SVC_1	キャッシュを初期化します。	system_Renesas_RZ_T1.c
__cmain **※ICCARM コンパイラ専用**	Reset_Handler()からコールされる C 言語の関数です。 C ライブラリのための初期化と OS 起動を行います。	RTX_CM_lib.h
cstartup **※RenesasGCC コンパイラ専用**	Reset_Handler()からコールされる C 言語の関数です。 C ライブラリのための初期化と OS 起動を行います。	RTX_CM_lib.h
SystemInit	Reset_Handler()からコールされる C 言語の関数です。 ボード依存のハードウェア初期化を行います。	system_Renesas_RZ_T1.c
submain	使用割り込みの登録とキャッシュ初期化を行います。	system_Renesas_RZ_T1.c
__low_level_init **※ICCARM コンパイラ専用**	低レベル初期化関数	low_level_init.c
R_ATCM_WaitSet	ATCM ウェイト設定関数	r_atcm_init.c
R_CPG_WriteEnable	クロック発生回路関連レジスタへの書き込みを許可にします。	r_cpg.c
R_CPG_WriteDisable	クロック発生回路関連レジスタへの書き込みを無効にします。	r_cpg.c
R_CPG_PLL_Wait	PLL 安定待ち処理	r_cpg.c
R_ECM_Init	エラーコントロールモジュールの初期化	r_ecm.c
R_ECM_CompareError_Wait	15us の間コンペアマッチタイマ割り込みの発生待ち処理	r_ecm.c
R_ECM_Write_Reg8	エラーコントロールモジュールレジスタへの 8bit 書き込み	r_ecm.c
R_ECM_Write_Reg32	エラーコントロールモジュールレジスタへの 32bit 書き込み	r_ecm.c
R_MPC_WriteEnable	マルチピンファンクション関連レジスタへの書き込	r_mpc.c

	みを許可にします。	
R_MPC_WriteDisable	マルチピンファンクション関連レジスタへの書き込みを無効にします。	r_mpc.c
R_RST_WriteEnable	リセット関連レジスタへの書き込みを許可にします。	r_reset.c
R_RST_WriteDisable	リセット関連レジスタへの書き込みを無効にします。	r_reset.c
reset_check	リセットフラグ判定処理	RZ_T1_RSK_Init.c
cpg_init	クロック・パルス発信器（CPG）の初期化	RZ_T1_RSK_Init.c
copy_to_atcm	外部フラッシュから ATCM へ AP をコピーします。	RZ_T1_RSK_Init.c
SER_Init	デバッグ・シリアルの初期化	Serial.c
SER_Enable	デバッグ・シリアルの有効化	Serial.c
SER_Disable	デバッグ・シリアルの無効化	Serial.c
SER_Set_baud_rate	デバッグ・シリアル・ボーレート設定	Serial.c
SER_PutChar	デバッグ・シリアル 1 文字出力	Serial.c
SER_GetChar	デバッグ・シリアル 1 文字入力	Serial.c
interrupt_SER	デバッグ・シリアル割り込みハンドラ(未使用)	Serial.c
FPUEnable	浮動小数点ユニットを許可にします。	fpuenable.s
vic_isr_001～ vic_isr_300	各ベクタ割り込み（VIC1～VIC300）	icu_vic.s
vic_isr	ベクタ割り込み判定処理	icu_vic.s
act_irq	割り込み実処理	icu_vic.s
ret_irq	割り込みの後処理	icu_vic.s

以下に、IAR C/C++コンパイラ依存内部関数を記します。

表 5.28　IAR C/C++コンパイラ依存内部関数

関数名	概要	実装ファイル
__read	文字バッファのリード	read.c
__write	文字バッファのライト	write.c
__lseek	ファイル・シーク	lseek.c
__close	ファイル・クローズ	close.c

以下に、RenesasGCC コンパイラ依存内部関数を記します。

表 5.29　GNU C/C++コンパイラ依存内部関数

関数名	概要	実装ファイル
_read	文字バッファのリード	syscalls.c
_write	文字バッファのライト	syscalls.c
_exit	プログラム終了	syscalls.c
SioRead	SCIF への文字出力	siorw.c
SioWrite	SCIF からの文字入力	siorw.c
_top_of_heap	ヒープ終端アドレス取得関数	newheap.c
__enable_irq	IRQ 割り込みの有効化	exclusion_func.c
__disable_irq	IRQ 割り込みの無効化	exclusion_func.c
__strex	排他的レジスタストア命令	exclusion_func.c
__clrex	排他をクリアする命令	exclusion_func.c
__ldrex	排他的レジスタロード命令	exclusion_func.c

以下に、ARM C/C++コンパイラ依存内部関数を記します。

表 5.30　ARM C/C++コンパイラ依存内部関数

関数名	概要	実装ファイル
__rt_entry	ARM C ライブラリライブラリ初期化 ※この関数は CMSIS-RTOS RTX ヘッダ内にあり、カスタマイズ不要	RTX_CM_lib.h
_user_perthread_libspace	ARM C ライブラリ・スレッドのローカルデータ領域の取得 ※この関数は CMSIS-RTOS RTX ヘッダ内にあり、カスタマイズ不要	RTX_CM_lib.h
_mutex_initialize	ARM C ライブラリ排他用ミューテックスの資源生成 ※この関数は CMSIS-RTOS RTX ヘッダ内にあり、カスタマイズ不要	RTX_CM_lib.h
_mutex_acquire	ARM C ライブラリ排他用ミューテックスのロックを取得 ※この関数は CMSIS-RTOS RTX ヘッダ内にあり、カスタマイズ不要	RTX_CM_lib.h
_mutex_release	ARM C ライブラリ排他用ミューテックスのロックを解除 ※この関数は CMSIS-RTOS RTX ヘッダ内にあり、カスタマイズ不要	RTX_CM_lib.h
fgetc	ファイルから 1 文字読み込み	retarget.c
fputc	ファイルへ 1 文字書き出し	retarget.c
ferror	ファイルのエラー状態の検知を行います。	retarget.c
SioRead	SCIF への文字出力	read.c
SioWrite	SCIF からの文字入力	write.c
_ttywrch	ARM C ライブラリ　端末への 1 文字出力	retarget.c
_sys_exit	ARM C ライブラリライブラリ終了	retarget.c

以下に、CMSIS-RTOS RTX 依存の処理を含む内部関数を示します(RZ/T1 評価ボード依存の処理を含む関数があります。

表 5.31　CMSIS-RTOS RTX 依存内部関数

関数名	概要	実装ファイル
os_idle_demon	CMSIS-RTOS RTX の idle スレッド	RTX_Conf_CM.c
os_tick_init	CMSIS-RTOS RTX 周辺機能タイマの初期化	RTX_Conf_CM.c
os_tick_val	CMSIS-RTOS RTX ハードウエアタイマーの現在値の取得	RTX_Conf_CM.c
os_tick_ovf	CMSIS-RTOS RTX ハードウエアタイマーのオーバーフローの有無	RTX_Conf_CM.c
os_tick_irqack	CMSIS-RTOS RTX 周辺機能タイマの割り込みステータス・クリア	RTX_Conf_CM.c
os_error	CMSIS-RTOS RTX のランタイム・エラー発生時のハングアップ	RTX_Conf_CM.c
OS_Tick_Handler	CMSIS-RTOS RTX 周辺機能タイマの ISR ハンドラ	HAL_CR4_asm.s
os_pendsv_init	CMSIS-RTOS RTX　PendSV 割込みの初期化	RTX_Conf_CM.c
os_pendsv_irqack	CMSIS-RTOS RTX PendSV 割り込みステータス・クリア	RTX_Conf_CM.c
os_pendsv	CMSIS-RTOS RTX PendSV 割り込みハンドラの処理	RTX_Conf_CM.c

以下に、システム初期化後ドライバや、ユーザー・アプリケーションから利用可能な関数を示します。

表 5.32　ユーティリティ関数

関数名	概要	実装ファイル
RZ_T1_RSK_InitClock	内部クロック倍率設定	RZ_T1_RSK_Init.c
InterruptHandlerRegister	IRQ ハンドラ登録関数	system_Renesas_RZ_T1.c
InterruptHandlerUnregister	IRQ ハンドラ登録解除関数	system_Renesas_RZ_T1.c

以下に、Cortex-R4F ベクタ割り込みの操作関数を示します。

表 5.33　ベクタ割り込み操作関数

関数名	概要	実装ファイル
VIC_EnableIRQ	Cortex-R4F ベクタ割り込みを有効にします。	vic.c
VIC_DisableIRQ	Cortex-R4F ベクタ割り込みを無効にします。	vic.c

VIC_SetDetectType	Cortex-R4F 割り込みのタイプの設定	vic.c
VIC_SetPriority	Cortex-R4F ベクタ割り込みの優先度を設定します。	vic.c
VIC_GetPriority	Cortex-R4F ベクタ割り込み優先度を取得します。	vic.c
VIC_Init	Cortex-R4F ベクタ割り込みコントローラ（VIC）の初期化	vic.c
VIC_Enable	Cortex-R4F ベクタ割り込みコントローラ（VIC）を有効にします。	vic.c
VIC_ClearPIC	割り込み信号をクリアします。	vic.c
VIC_EndInterrupt	割り込みの終了処理をします。	vic.c

6. 関数仕様

6.1 RZ/T1 評価ボード依存内部関数

6.1.1 Reset_Handler

Reset_Handler	RZ/T1 評価ボード依存内部関数
リセット・エントリ（ICCARM,RenesasGCC 用）	ハンドラ

ヘッダ	なし
宣　言	Reset_Handler .global Reset_Handler .func Reset_Handler
引　数	なし
戻り値	なし
呼出し元	割り込みベクタ・アドレスに配置、リセット時に自動実行
説　明	アセンブラ関数 以下の順で初期化処理を実行します ● CP15 の動作モード初期化（キャッシュ無効、仮想メモリ無効、分岐予測無効） ● 全例外モードのスタック設定（個別のスタックを割り当て） ● 必要な ROM セクションイメージを RAM セクションへコピー（コンパイラにより処理は異なる） ● SystemInit 関数の実行(プラットフォーム依存のペリフェラル初期化) ● __cmain 関数へジャンプ（ICCARM の場合）、cstartup 関数へジャンプ（RenesasGCC の場合）
補　足	ICCARM,RenesasGCC 用に関数が存在します（処理内容は若干異なります）。

6.1.2 user_init

user_init	RZ/T1 評価ボード依存内部関数
リセット・エントリ（ARMCC 用）	ハンドラ

ヘッダ	なし
宣　言	user_init
引　数	なし
戻り値	なし
呼出し元	RAM ブート時：cpu_init、NOR/SPI ブート時：リセット時に自動実行
説　明	アセンブラ関数 以下の順で初期化処理を実行します ● CP15 の動作モード初期化（キャッシュ無効、仮想メモリ無効、分岐予測無効） ● 全例外モードのスタック設定（個別のスタックを割り当て） ● 必要な ROM セクションイメージを RAM セクションへコピー ● SystemInit 関数の実行(プラットフォーム依存のペリフェラル初期化) ● __main 関数へジャンプ
補　足	ARMCC 用の関数です。

6.1.3 cpu_init

cpu_init

RZ/T1 評価ボード依存内部関数

リセット・エントリ（ARMCC 用）

ハンドラ

ヘッダ	なし
宣　言	user_init
引　数	なし
戻り値	なし
呼出し元	NOR/SPI ブート時：リセット時に自動実行、RAM ブート時には実行されません
説　明	アセンブラ関数 以下の順で初期化処理を実行します ● クロック、バスの初期化 ● 必要な ROM セクションイメージを RAM セクションへコピー ● エラーコントロールモジュール（ECM）を初期化します。 ● user_init 関数へジャンプ
補　足	ARMCC 用の関数です。 NOR/SPI ブート時のみ使用します。

6.1.4 Undef_Handler

Undef_Handler

RZ/T1 評価ボード依存内部関数

未定義命令例外ハンドラ

ハンドラ

ヘッダ	なし
宣　言	Undef_Handler
引　数	なし
戻り値	なし
呼出し元	未定義命令発生コード
説　明	未定義命令例外のハンドラ アセンブラハンドラ ● 未定義命令例外を発生した位置の動作モードを取得（ARM モード、Thumb モード） ● 未定義命令例外を発生した位置の opecode を取得 ● 例外発生位置（LR（リンクレジスタ）の値）を取得 ● 上記３つの値を引数として、CUndefHandler 関数をコールします。
補　足	各コンパイラ用に同じ関数が存在します。 CUndefHandler 関数については、6.1.7 章参照

6.1.5 PAbt_Handler

PAbt_Handler	RZ/T1 評価ボード依存内部関数
プリフェッチアボート例外ハンドラ	ハンドラ

ヘッダ	なし
宣　言	PAbt_Handler
引　数	なし
戻り値	なし
呼出し元	プリフェッチアボート例外発生コード
説　明	プリフェッチアボート例外のハンドラ アセンブラハンドラ ● 命令フォールト・ステータスレジスタを取得 ● 命令フォールト・アドレスレジスタを取得 ● 例外発生位置（LR（リンクレジスタ）の値）を取得 ● 上記３つの値を引数として、CPAbtHandler 関数をコールします。
補　足	各コンパイラ用に同じ関数が存在します。 CPAbtHandler 関数については 6.1.8 章参照

6.1.6 DAbt_Handler

DAbt_Handler	RZ/T1 評価ボード依存内部関数
データアボート例外ハンドラ	ハンドラ

ヘッダ	なし
宣　言	DAbt_Handler
引　数	なし
戻り値	なし
呼出し元	データアボート例外発生コード
説　明	データアボート例外のハンドラ アセンブラハンドラ ● データフォールト・ステータスレジスタを取得 ● データフォールト・アドレスレジスタを取得 ● 例外発生位置（LR（リンクレジスタ）の値）を取得 ● 上記３つの値を引数として、CDAbtHandler 関数をコールします。
補　足	各コンパイラ用に同じ関数が存在します。 CDAbtHandler 関数については 6.1.9 章参照

6.1.7 CUndefHandler

CUndefHandler	RZ/T1 評価ボード依存内部関数
未定義命令例外 C 言語ハンドラ	同期関数

ヘッダ	なし		
宣　言	uint32_t CUndefHandler(uint32_t opcode, uint32_t state, uint32_t LR);		
引　数	uint32_t opcode	I	未定義命令発生時の opcode
	uint32_t state	I	未定義例外発生時の state（ARM または Thumb）
	uint32_t LR	I	未定義例外発生位置（LR（リンクレジスタ）の値）
戻り値	uint32_t		
説　明	未定義命令例外が FPU 命令によって発生した場合、FPU をイネーブルにし、コール元へ return します。 それ以外の場合、無限ループします。		
補　足	各コンパイラ用に同じ関数が存在します。		

6.1.8 CPAbtHandler

CPAbtHandler

RZ/T1 評価ボード依存内部関数

プリフェッチアボート例外 C 言語ハンドラ

同期関数

ヘッダ	なし		
宣　言	void CPAbtHandler(uint32_t IFSR, uint32_t IFAR, uint32_t LR);		
引　数	uint32_t IFSR	I	命令フォールト・ステータスレジスタ
	uint32_t IFAR	I	命令フォールト・アドレスレジスタ
	uint32_t LR	I	未定義例外発生位置（LR（リンクレジスタ）の値）
戻り値	なし		
説　明	命令フォールト・ステータスレジスタの値より、復旧可能な場合、コール元へ return します。それ以外の場合、無限ループします。		
補　足	各コンパイラ用に同じ関数が存在します。		

6.1.9 CDAbtHandler

CDAbtHandler

RZ/T1 評価ボード依存内部関数

データアボート例外 C 言語ハンドラ

同期関数

ヘッダ	なし		
宣　言	void CDAbtHandler(uint32_t DFSR, uint32_t DFAR, uint32_t LR);		
引　数	uint32_t DFSR	I	データフォールト・ステータスレジスタ
	uint32_t DFAR	I	データフォールト・アドレスレジスタ
	uint32_t LR	I	未定義例外発生位置（LR（リンクレジスタ）の値）
戻り値	なし		
説　明	データフォールト・ステータスレジスタの値より、復旧可能な場合、コール元へ return します。それ以外の場合、無限ループします。		
補　足	各コンパイラ用に同じ関数が存在します。		

6.1.10 __SVC_1

__SVC_1

RZ/T1 評価ボード依存内部関数

キャッシュの初期化

同期関数

ヘッダ	なし
宣　言	void __SVC_1(void);
引　数	なし
戻り値	なし
呼出し元	submain() （各コンパイラ毎の system_Renesas_RZ_T1.c で定義）
説　明	キャッシュ、MPU を初期化します。
補　足	各コンパイラ用に同じ関数が存在します。

6.1.11 __cmain

__cmain	RZ/T1 評価ボード依存内部関数
ICCARM 用スタートアップ関数	同期関数

ヘッダ	RTX_CM_lib.h
宣　言	__noreturn __stackless void __cmain(void);
引　数	なし
戻り値	なし
説　明	OS 初期化の初期化を行い、OS を起動します。 以下の順で OS を起動します ● データセクションをシステム起動コードで初期化が必要な場合、初期化を行います。 ● submain 関数（6.1.14 参照）をコールし、使用割り込みの登録とキャッシュ初期化を行います。 ● osKernelInitialize 関数をコールし、CMSIS-RTOS RTX の初期化を行います。 ● main 関数を実施する os_thread_def_main スレッドを起動します。 ● osKernelStart 関数をコールして、RTOS カーネルを起動し、スレッドスイッチを開始します。
補　足	ICCARM 用の関数です

6.1.12 cstartup

cstartup	RZ/T1 評価ボード依存内部関数
RenesasGCC 用スタートアップ関数	同期関数

ヘッダ	RTX_CM_lib.h
宣　言	void cstartup(void);
引　数	なし
戻り値	なし
説　明	OS 初期化の初期化を行い、OS を起動します。 以下の順で OS を起動します ● submain 関数をコールし、使用割り込みの登録とキャッシュ初期化を行います。 ● osKernelInitialize 関数をコールし、CMSIS-RTOS RTX の初期化を行います。 ● main 関数を実施する os_thread_def_main スレッドを起動します。 ● osKernelStart 関数をコールして、RTOS カーネルを起動し、スレッドスイッチを開始します。
補　足	RenesasGCC 用の関数です

6.1.13 SystemInit

SystemInit	RZ/T1 評価ボード依存内部関数
RZ/T1 評価ボード依存ハードウェア初期化	同期関数

ヘッダ	system_Renesas_RZ_T1.h
宣　言	void SystemInit (void);
引　数	なし
戻り値	なし
説　明	プラットフォーム依存の以下の初期化設定を行います ● リセットフラグ判定処理 ● クロック倍率設定 ● エラーコントロールモジュール（ECM）を初期化 ● 割り込みコントローラ初期化
補　足	各コンパイラ用に同じ関数が存在します。 CMSIS-RTOS RTX サンプルプログラムのプラットフォーム依存の処理を記述する関数です

6.1.14 submain

submain

RZ/T1 評価ボード依存内部関数

使用割り込みの登録とキャッシュ初期化

同期関数

ヘッダ	RTX_CM_lib.h
宣　言	void submain(void);
引　数	なし
戻り値	なし
説　明	使用割り込みの登録とキャッシュ初期化を行います。 ● OS_Tick_Handler 割り込み登録及び初期化（CMT4） ● PendSV_Handler 割り込み登録及び初期化（CMT5） ● __SVC_1 関数（6.1.10 章参照）をコールし、キャッシュ初期化及び MPU 設定
補　足	各コンパイラ用に同じ関数が存在します。

6.1.15 __low_level_init

__low_level_init

RZ/T1 評価ボード依存内部関数

低レベル初期化関数

ハンドラ

ヘッダ	RTX_CM_lib.h
宣　言	int __low_level_init(void);
引　数	なし
戻り値	Int　0 :低レベル初期化を行わない。 1 :低レベル初期化を行う。
呼び出し元	void __cmain(void);
説　明	データセクションをシステム起動コードで初期化するかどうかを決定するための関数です。 関数が 0 を返す場合、データセクションは初期化されません。
補　足	ICCARM のみ__cmain 関数（6.1.11 章参照）よりコールされますが、機能実装されていません。

6.1.16 R_ATCM_WaitSet

R_ATCM_WaitSet

RZ/T1 評価ボード依存内部関数

ATCM アクセスウェイト設定処理

同期関数

ヘッダ	r_atcm_init.h
宣　言	void R_ATCM_WaitSet(uint32_t atcm_wait);
引　数	uint32_t atcm_wait　I　ATCM ウェイト設定ビット b1 b0 0　0　: 1-wait 最適化あり 0　1　: 1-wait 最適化なし 1　0　: 0-wait 1　1　: 設定禁止
戻り値	なし
説　明	ATCM のメモリアクセスウェイト数を設定します。 “最適化あり”の場合、ATCM からの命令フェッチアドレスが連続しているとき、先行アドレスを先読みすることでメモリアクセスを実質 wait-0 に高速化することが可能です。
補　足	CPU からのフェッチアクセスを防ぐため、本設定関数は必ず ATCM 以外のメモリ領域に配置されたプログラムから実行してください。

6.1.17 R_CPG_WriteEnable

R_CPG_WriteEnable

RZ/T1 評価ボード依存内部関数

クロック発生回路関連レジスタ書込許可

同期関数

ヘッダ	r_cpg.h
宣　言	void R_CPG_WriteEnable (void);
引　数	なし
戻り値	なし
説　明	クロック発生回路関連レジスタへの書き込みを許可にします。
補　足	-

6.1.18 R_CPG_WriteDisable

R_CPG_WriteDisable

RZ/T1 評価ボード依存内部関数

クロック発生回路関連レジスタ書込禁止

同期関数

ヘッダ	r_cpg.h
宣　言	void R_CPG_WriteDisable(void);
引　数	なし
戻り値	なし
説　明	クロック発生回路関連レジスタへの書き込みを禁止にします。
補　足	-

6.1.19 R_CPG_PLL_Wait

R_CPG_PLL_Wait

RZ/T1 評価ボード依存内部関数

PLL 安定待ち処理

同期関数

ヘッダ	r_cpg.h
宣　言	void R_CPG_PLL_Wait(void);
引　数	なし
戻り値	なし
説　明	PLL1 の発振安定待ち時間として CMT0 を使用し、100us 待ちを行います。
補　足	関数内での CMT0 の初期化や動作停止を行います。 CMT0 を使用する場合はご注意ください。

6.1.20 R_ECM_Init

R_ECM_Init

RZ/T1 評価ボード依存内部関数

エラーコントロールモジュールの初期化

同期関数

ヘッダ	r_ecm.h
宣　言	void R_ECM_Init(void);
引　数	なし
戻り値	なし
説　明	エラーコントロールモジュールを初期化します。
補　足	-

6.1.21 R_ECM_CompareError_Wait

R_ECM_CompareError_Wait

RZ/T1 評価ボード依存内部関数

コンペアマッチタイマ割込発生待ち処理

同期関数

ヘッダ	r_ecm.h
宣　言	void R_ECM_CompareError_Wait(void);
引　数	なし
戻り値	なし
説　明	CMT0 を使用し、15us の間コンペアマッチタイマ割り込みの発生待ちを行います。
補　足	関数内での CMT0 の初期化や動作停止を行います。 CMT0 を使用する場合はご注意ください。

6.1.22 R_ECM_Write_Reg8

R_ECM_Write_Reg8

RZ/T1 評価ボード依存内部関数

エラーコントロールモジュールレジスタへの 8bit 書込

同期関数

ヘッダ	r_ecm.h		
宣　言	uint8_t R_ECM_Write_Reg8(uint8_t reg_type, volatile unsigned char *reg, uint8_t value);		
引　数	uint8_t reg_type	I	ECM のマスタ/チェッカを指定します。
	volatile unsigned char *reg	I	ECM のレジスタアドレスを指定します（8 ビットアクセスのみ）。
	uint8_t value	I	書き込む値を指定します。
戻り値			
説　明	エラーコントロールモジュールレジスタへの 8bit 書き込みを行います。		
補　足	-		

6.1.23 R_ECM_Write_Reg32

R_ECM_Write_Reg32

RZ/T1 評価ボード依存内部関数

エラーコントロールモジュールレジスタへの 32bit 書込

同期関数

ヘッダ	r_ecm.h		
宣　言	uint8_t R_ECM_Write_Reg32(uint8_t reg_type, volatile unsigned long *reg, uint32_t value);		
引　数	uint8_t reg_type	I	ECM のマスタ/チェッカを指定します。
	volatile unsigned long *reg	I	ECM のレジスタアドレスを指定します（32 ビットアクセスのみ）。
	uint32_t value	I	書き込む値を指定します。
戻り値			
説　明	エラーコントロールモジュールレジスタへの 32bit 書き込みを行います。		
補　足	-		

6.1.24 R_MPC_WriteEnable

R_MPC_WriteEnable

RZ/T1 評価ボード依存内部関数

マルチピンファンクション関連レジスタ書込許可

同期関数

ヘッダ	r_mpc.h
宣　言	void R_MPC_WriteEnable(void);
引　数	なし
戻り値	なし
説　明	マルチピンファンクション関連レジスタへの書き込みを許可にします。
補　足	-

6.1.25 R_MPC_WriteDisable

R_MPC_WriteDisable

RZ/T1 評価ボード依存内部関数

マルチピンファンクション関連レジスタ書込禁止

同期関数

ヘッダ	r_mpc.h
宣　言	void R_MPC_WriteDisable(void);
引　数	なし
戻り値	なし
説　明	マルチピンファンクション関連レジスタへの書き込みを禁止にします。
補　足	-

6.1.26 R_RST_WriteEnable

R_RST_WriteEnable

RZ/T1 評価ボード依存内部関数

リセット関連レジスタへ書込許可

同期関数

ヘッダ	r_reset.h
宣　言	void R_RST_WriteEnable(void);
引　数	なし
戻り値	なし
説　明	リセット関連レジスタへの書き込みを許可にします。
補　足	-

6.1.27 R_RST_WriteDisable

R_RST_WriteDisable

RZ/T1 評価ボード依存内部関数

リセット関連レジスタへ書込禁止

同期関数

ヘッダ	r_reset.h
宣　言	void R_RST_WriteDisable(void);
引　数	なし
戻り値	なし
説　明	リセット関連レジスタへの書き込みを禁止にします。
補　足	-

6.1.28 reset_check

reset_check	RZ/T1 評価ボード依存内部関数
リセット判定	同期関数

ヘッダ	なし
宣　言	void reset_check(void);
引　数	なし
戻り値	なし
説　明	リセット要因を判定し、各シーケンスを実行します。
補　足	-

6.1.29 cpg_init

cpg_init	RZ/T1 評価ボード依存内部関数
クロック・パルス発信器（CPG）の初期化	同期関数

ヘッダ	なし
宣　言	void cpg_init(void)
引　数	なし
戻り値	なし
説　明	クロック・パルス発信器（CPG）の初期化を行います。 PLL1 を使用し、CPU クロックを 450MHz に設定します。 また低速オンチップオシレータを動作させます。
補　足	-

6.1.30 copy_to_atcm

copy_to_atcm	RZ/T1 評価ボード依存内部関数
ユーザーアプリケーションプログラムのコピー処理	同期関数

ヘッダ	なし
宣　言	void copy_to_atcm(void)
引　数	なし
戻り値	なし
説　明	4 バイト単位で外付けフラッシュ（NOR/シリアルフラッシュメモリ）から内蔵 RAM（ATCM）へユーザーアプリケーションプログラムのコピー処理を行います。また例外ベクタやユーザー・アプリケーション用初期値あり変数のコピーも行います。
補　足	各コンパイラ用に同じ関数が存在します。 RenesasGCC のみアセンブラで実装されています。

6.1.31 SER_Init

SER_Init

RZ/T1 評価ボード依存内部関数

デバッグ・シリアルの初期化

同期関数

ヘッダ	Serial.h
宣　言	void SER_Init(void)
引　数	なし
戻り値	なし
説　明	デバッグ・シリアルの端子設定を行い、初期化関数を実行します 設定は以下の通り。 対象：SCIF2 シリアルポート設定：調歩同期式モード、8 ビットキャラクタ、ストップビット長 1、フロー制御なし 転送速度：115200bps 出力端子：P3_0(TxD2) 入力端子：P3_2(RxD2)
補　足	デバッグ・シリアルとして SCIF2 を使用するためご注意ください。

6.1.32 SER_Enable

SER_Enable

RZ/T1 評価ボード依存内部関数

デバッグ・シリアルの有効化

同期関数

ヘッダ	Serial.h
宣　言	void SER_Enable(void)
引　数	なし
戻り値	なし
呼び出し元	SER_Init ()
説　明	デバッグ・シリアルの送信、受信機能を有効にします。
補　足	-

6.1.33 SER_Disable

SER_Disable

RZ/T1 評価ボード依存内部関数

デバッグ・シリアルの無効化

同期関数

ヘッダ	Serial.h
宣　言	void SER_Disable(void)
引　数	なし
戻り値	なし
説　明	デバッグ・シリアルの送信、受信機能を無効化します
補　足	サンプルでは未使用

6.1.34 SER_Set_baud_rate

SER_Set_baud_rate

RZ/T1 評価ボード依存内部関数

デバッグ・シリアル・ボーレート設定

同期関数

ヘッダ	Serial.h
宣　言	void SER_Set_baud_rate(uint32_t baud_rate)
引　数	uint32_t baud_rate　Ⅰ　未使用
戻り値	なし
呼び出し元	なし
説　明	デバッグ・シリアルのボーレート設定を行います。
補　足	機能実装していません（空関数です）。 ボーレート設定は SER_Init 関数（6.1.31 章参照）で行っています。

6.1.35 SER_PutChar

SER_PutChar

RZ/T1 評価ボード依存内部関数

デバッグ・シリアル 1 文字出力

非同期関数

ヘッダ	Serial.h
宣　言	void SER_PutChar(uint8_t buffer)
引　数	uint8_t buffer　Ⅰ　出力文字
戻り値	なし
呼び出し元	__read（）、__write()
説　明	デバッグ・シリアルへ 1 文字のデータを送信します 送信前に送信 FIFO が空ではない場合は、送信 FIFO が空になるまでポーリングし、空になってから送信を実行します
補　足	ポーリングによる待ちを行うため、使用には注意が必要です デバッグ用の入出力関数です

6.1.36 SER_GetChar

SER_GetChar

RZ/T1 評価ボード依存内部関数

デバッグ・シリアル 1 文字入力

同期関数

ヘッダ	Serial.h	
宣　言	uint32_t SER_GetChar (void)	
引　数	なし	
戻り値	0～255	成功：入力文字データ
	0	失敗：ハードウェアエラー発生
呼び出し元	__read()	
説　明	デバッグ・シリアルから 1 文字のデータを受信します 受信 FIFO にデータがない場合は、受信データが発生するまでリターンせず、ポーリングします	
補　足	ポーリングによる待ちを行うため、使用には注意が必要です デバッグ用の入出力関数です	

6.1.37 interrupt_SER

interrupt_SER

RZ/T1 評価ボード依存内部関数

デバッグ・シリアル割り込みハンドラ

同期関数

ヘッダ	Serial.h
宣　言	void interrupt_SER(void);
引　数	なし
戻り値	なし
呼び出し元	なし
説　明	デバッグ・リシアルの割込みハンドラでの処理を行います。
補　足	機能実装していません（空関数です）。

6.1.38 FPUEnable

FPUEnable

RZ/T1 評価ボード依存内部関数

浮動小数点ユニット（FPU）の有効化

同期関数

ヘッダ	なし
宣　言	void FPUEnable(void);
引　数	なし
戻り値	なし
説　明	浮動小数点ユニット（FPU）を初期化して有効にします。
補　足	各コンパイラ用に同じ関数が存在します。

6.1.39 vic_isr_001～vic_isr_300

vic_isr_001～vic_isr_300

RZ/T1 評価ボード依存内部関数

ベクタ割り込み処理関数

非同期関数

ヘッダ	なし

宣　言	void vic_isr_001(void);
	:
	void vic_isr_299(void);
	void vic_isr_300(void);

引　数	なし
戻り値	なし
説　明	ベクタ番号 1-300 の各割り込みのベクタ・アドレスです。 R0,R1 をスタックした後、R1 にベクタ番号を入力し、vic_isr 関数にジャンプします。
補　足	各コンパイラ用に同じ関数が存在します。

6.1.40 vic_isr

vic_isr	RZ/T1 評価ボード依存内部関数
ベクタ割り込み処理	非同期関数

ヘッダ	なし		
宣　言	void vic_isr(int);		
引　数	Int	I	ベクタ番号 1-300
戻り値	なし		
説　明	ベクタ割り込み処理の前処理を行います。 プロセッサモードをスーパーバイザモードに移行し、割り込みネストレベルを+1 します。 ベクタ番号の範囲を確認し、範囲内であれば act_irq 関数（6.1.41 章参照）にジャンプします。 ベクタ番号が範囲外の場合、ret_irq 関数（6.1.42 章参照）にジャンプします。		
補　足	各コンパイラ用に同じ関数が存在します。		

6.1.41 act_irq

act_irq	RZ/T1 評価ボード依存内部関数
割り込み実処理	非同期関数

ヘッダ	なし		
宣　言	void act_irq(int);		
引　数	Int	I	ベクタ番号 1-300
戻り値	なし		
説　明	割り込み信号のクリア関数 VIC_ClearPIC（6.7.8 章参照）を実行します。 割り込みテーブルに割り当てられた実処理関数を実行します。		
補　足	各コンパイラ用に同じ関数が存在します。		

6.1.42 ret_irq

ret_irq	RZ/T1 評価ボード依存内部関数
割り込みの後処理	非同期関数

ヘッダ	なし
宣　言	void ret_irq(void);
引　数	なし
戻り値	なし
説　明	割り込みアドレスレジスタ（HVA0）に 0 を設定し、割り込み処理を終了します。 割り込みのネストレベル-1 します。 割り込みの後処理を行います。
補　足	各コンパイラ用に同じ関数が存在します。

6.2 IAR C/C++コンパイラ依存内部関数

6.2.1 __read

__read	IAR C/C++依存内部関数
文字バッファのリード	同期関数

ヘッダ	なし		
宣　言	size_t __read(int handle, unsigned char * buffer, size_t size);		
引　数		int handle	I “_LLIO_STDIN”（standard in）のみ指定可能
		unsigned char * buffer	I/O リード・データ格納バッファのポインタ
		size_t size	I 受信要求サイズ（バイト）
戻り値	>= 0		成功：受信バイト数
	< 0		失敗：受信エラー
説　明	シリアルポートからデータを受信し、buffer で指定されたアドレスに格納します。		
補　足	-		

6.2.2 __write

__write	IAR C/C++依存内部関数
文字バッファのライト	同期関数

ヘッダ	なし		
宣　言	size_t __write(int handle, const unsigned char * buffer, size_t size);		
引　数		int handle	I “_LLIO_STDOUT”（standard out）または” _LLIO_STDERR”（standard err）のみ指定可能
		unsigned char * buffer	I ライトデータ格納バッファのポインタ
		size_t size	I 送信要求サイズ（バイト）
戻り値	>= 0		成功：送信バイト数
	< 0		失敗：送信エラー
説　明	シリアルポートからデータを送信します。		
補　足	-		

6.2.3 __lseek

__lseek	IAR コンパイラ依存内部関数
ファイル・シーク	同期関数

ヘッダ	なし		
宣　言	int __lseek(int handle, long offset, int whence);		
引　数		int handle	I 使用しません
		long offset	I 使用しません
		int whence	I 使用しません
戻り値	-1		固定
呼び出し元	DLIB ライブラリ(lseek 関数)		
説　明	seek 関数からコールされるプラットフォーム依存のシーク処理関数		
補　足	機能実装していません（何もせず、-1 をリターンします）。		

6.2.4 __close

__close	IAR コンパイラ依存内部関数
ファイル・クローズ	同期関数

ヘッダ	なし
宣　言	int __close(int handle);
引　数	int handle　　Ｉ　使用しません
戻り値	0
呼び出し元	DLIB ライブラリ(close 関数)
説　明	close 関数からコールされるプラットフォーム依存のクローズ処理関数
補　足	機能実装していません（何もせず、0 をリターンします）。

6.3 RenesasGCC コンパイラ依存内部関数

6.3.1 _read

_read	RenesasGCC 依存内部関数
文字バッファのリード	同期関数

ヘッダ	なし		
宣　言	int _read(int file, char * ptr, int len)		
引　数	int file	I	“STDIN”（standard in）のみ指定可能
	char * ptr	I/O	リード・データ格納バッファのポインタ
	Int len	I	受信要求サイズ（バイト）
戻り値	>= 0	成功：受信バイト数	
	< 0	失敗：受信エラー	
説　明	シリアルポートからデータを受信し、buffer で指定されたアドレスに格納します。 また、受信したデータをシリアルポートから出力します。		
補　足	-		

6.3.2 _write

_write	RenesasGCC 依存内部関数
文字バッファのライト	同期関数

ヘッダ	なし		
宣　言	int _write(int file, char * ptr, int len)		
引　数	int file	I	“STDOUT”（standard out）または” STDERR”（standard err）のみ指定可能
	char * ptr	I	ライトデータ格納バッファのポインタ
	Int len	I	送信要求サイズ（バイト）
戻り値	>= 0	成功：送信バイト数	
	< 0	失敗：送信エラー	
説　明	シリアルポートからデータを送信します。		
補　足	-		

6.3.3 _exit

_exit	RenesasGCC 依存内部関数
プログラムの終了	同期関数

ヘッダ	なし		
宣　言	void _exit(int status);		
引　数	int status	I	終了ステータス
戻り値	なし		
説　明	無限ループし、リターンしません。		
補　足	-		

6.3.4 SioRead

SioRead	RenesasGCC 依存内部関数
SCIF への文字出力	同期関数

ヘッダ	なし		
宣　言	int32_t SioRead(int32_t file_no, int_t * buffer, uint32_t reading_b);		
引　数	int32_t file_no	I	“STDIN”（standard in）のみ指定可能
	int_t * buffer	I/O	読み出し用文字バッファ
	uint32_t reading_b	I	文字バッファサイズ
戻り値	>= 0		成功：送信バイト数
	-1		失敗：ファイル番号エラー、読み出し文字エラー
説　明	シリアルポートからデータを受信し、buffer で指定されたアドレスに格納します。 また、受信したデータをシリアルポートから出力します。		
補　足	-		

6.3.5 SioWrite

SioWrite	RenesasGCC 依存内部関数
SCIF からの文字入力	同期関数

ヘッダ	なし		
宣　言	int32_t SioWrite(int32_t file_no, const int_t * buffer, uint32_t writing_b);		
引　数	int32_t file_no	I	ファイル番号
	const int_t * buffer	I	書き込み用文字バッファ
	uint32_t writing_b	I	文字バッファサイズ
戻り値	>= 0		成功：送信バイト数
	-1		失敗：ファイル番号エラー
説　明	シリアルポートからデータを送信します。		
補　足	-		

6.3.6 _top_of_heap

_top_of_heap	RenesasGCC 依存内部関数
ヒープ終端アドレス取得関数	同期関数

ヘッダ	なし	
宣　言	char* _top_of_heap(void);	
引　数	なし	
戻り値	>= 0	スタックの先頭アドレス
説　明	スタックの先頭アドレスを返却する。 optlib のオーバーライト関数。	
補　足	-	

6.3.7 __enable_irq

__enable_irq	RenesasGCC 依存内部関数
IRQ 割り込みの有効化	同期関数

ヘッダ	なし		
宣　言	void __enable_irq(void);		
引　数	なし		
戻り値	なし		
説　明	IRQ 割り込みを有効化します。		
補　足	-		

6.3.8 __disable_irq

__disable_irq	RenesasGCC 依存内部関数
IRQ 割り込みの無効化	同期関数

ヘッダ	なし	
宣　言	uint32_t __disable_irq(void);	
引　数	なし	
戻り値	0	割り込みを許可
	!0	割り込みを無効化（禁止）
説　明	IRQ 割り込みを無効化します。	
補　足	-	

6.3.9 __strex

__strex	RenesasGCC 依存内部関数
排他的レジスタストア命令	同期関数

ヘッダ	なし	
宣　言	int __strex(unsigned int val, volatile char *ptr);	
引　数	unsigned int val	I　レジスタに書き込む値
	volatile char *ptr	I　書き込み先レジスタのアドレス
戻り値	0	書き込みロックされておらず、成功となる。
	1	書き込みロックされており、失敗となる。
説　明	排他的レジスタストア命令を実行する。	
補　足	-	

6.3.10 __clrex

__clrex	RenesasGCC 依存内部関数
排他をクリアする命令	同期関数

ヘッダ	なし
宣　言	void __clrex(void);
引　数	なし
戻り値	なし
説　明	排他をクリアする命令を実行する。
補　足	-

6.3.11 __ldrex

__ldrex	RenesasGCC 依存内部関数
排他的レジスタロード命令	同期関数

ヘッダ	なし		
宣　言	unsigned int __ldrex(volatile char *ptr);		
引　数	volatile char *ptr	I/O	読み出し先レジスタのアドレス
戻り値	>= 0		読み出した値
説　明	排他的レジスタロード命令を実行する。		
補　足	-		

6.4 ARM C/C++コンパイラ依存内部関数

6.4.1 __rt_entry

__rt_entry	ARM C/C++依存内部関数
ARM C ライブラリ利用プログラムのエントリ	同期関数

ヘッダ	RTX_CM_lib.h
宣　言	void __rt_entry (void);
引　数	なし
戻り値	なし
呼び出し元	リセット時に、Reset_Handler()から__main()経由でジャンプ。 ARM C ライブラリの__main()関数は、スキャッタファイルに指定されたメモリ領域のロードを行った後に、本関数にジャンプします。
説　明	ARM C ライブラリ使用時に、メモリのロード後に最初に実行される関数です。 以下の初期化設定を行います ● ARM C ライブラリ用スタックとヒープの初期化(__user_setup_stackheap()の呼び出し) ● ARM C ライブラリの初期化(__rt_lib_init()の呼び出し) ● CMSIS-RTOS RTX の初期化とスレッドの生成(osKernelInitialize()、osThreadCreate の呼び出し) ● CMSIS-RTOS RTX の起動(osKernelStart()の呼び出し)
補　足	osKernelStart()の正常動作時は、CMSIS-RTOS RTX が動作を開始し本関数にはリターンしません。 osKernelStart()のエラー終了時は、本関数内で exit()を実行して自身を終了します。 CMSIS-RTOS RTX ヘッダの RTX_CM_lib.h にインライン実装された関数です。

6.4.2 _user_perthread_libspace

_user_perthread_libspace	ARM C/C++依存内部関数
ARM C ライブラリ・スレッドのローカルデータ領域の取得	同期関数

ヘッダ	RTX_CM_lib.h	
宣　言	void *__user_perthread_libspace (void);	
引　数	なし	
戻り値	BSP 内のバッファ・アドレス	スレッドからコールされた場合： CMSIS-RTOS RTX のスレッド別の 96 バイト領域へのポインタ
	__libspace_start のアドレス	非スレッドからコールされた場合： RTOS 非対応時のデフォルト 96 バイト領域へのポインタ
呼び出し元	ARM C ライブラリ	
説　明	ARM C ライブラリがスレッド別に管理するスタティック・データの領域を返す関数です。 自スレッド ID を取得して、スレッド別の 96 バイト領域を返す処理を実装します。 非スレッド・コンテキストでは共通のデフォルト領域__libspace_start を返します。	
補　足	スレッド別に管理するスタティック・データとは、具体的には errno や浮動小数点コプロセッサ状態などです。 CMSIS-RTOS RTX ヘッダの RTX_CM_lib.h にインライン実装された関数です。	

6.4.3 _mutex_initialize

_mutex_initialize	ARM C/C++依存内部関数
ARM C ライブラリ排他用ミューテックスの資源生成	同期関数

ヘッダ	RTX_CM_lib.h		
宣　言	int _mutex_initialize(OS_ID *mutex);		
引　数	OS_ID *mutex	I/O	mutex オブジェクトを指すポインタ格納領域
戻り値	1		成功：mutex の生成に成功
呼び出し元	ARM C ライブラリ		
説　明	ARM C ライブラリのスレッド・セーフ動作のために、CMSIS-RTOS RTX 依存のセマフォ資源を生成する関数です。		
補　足	RTX_CM_lib.h にインライン実装されています。		

6.4.4 _mutex_acquire

_mutex_acquire	ARM C/C++依存内部関数
ARM C ライブラリ排他用ミューテックスのロックを取得	同期関数

ヘッダ	RTX_CM_lib.h		
宣　言	void _mutex_acquire(OS_ID *mutex);		
引　数	OS_ID *mutex	I	ミューテックス資源へのポインタ _mutex_initialize()正常終了時の第 1 引数 mutex の値を指定
戻り値	なし		
呼び出し元	ARM C ライブラリ		
説　明	ARM C ライブラリ排他用ミューテックスのロックを取得する関数です。		
補　足	RTX_CM_lib.h にインライン実装されています。		

6.4.5 _mutex_release

_mutex_release	ARM C/C++依存内部関数
ARM C ライブラリ排他用ミューテックスのロックを解除	同期関数

ヘッダ	RTX_CM_lib.h		
宣　言	void _mutex_release(OS_ID *mutex);		
引　数	OS_ID *mutex	I	ミューテックス資源へのポインタ _mutex_initialize()正常終了時の第 1 引数 mutex の値を指定
戻り値	なし		
呼び出し元	ARM C ライブラリ		
説　明	ARM C ライブラリ排他用ミューテックスのロックを解除する関数です。		
補　足	RTX_CM_lib.h にインライン実装されています。		

6.4.6 fgetc

fgetc ARM C/C++依存内部関数

1文字読み込み 同期関数

ヘッダ	なし		
宣　言	int fgetc(FILE * file_p);		
引　数	FILE * file_p	I	未使用
戻り値	get_data		読み込み文字（1 文字）
	EOF		文字無し（文字が読み込めなかった）
説　明	シリアルポートから 1 文字を読み込んで返却します。		
補　足	-		

6.4.7 fputc

fputc ARM C/C++依存内部関数

1文字書き込み 同期関数

ヘッダ	なし		
宣　言	int fputc(int character, FILE * file_p);		
引　数	int character	I	書き込み文字（1 文字）
	FILE * file_p	I	未使用
戻り値	character		書き込み文字（1 文字）※引数をそのまま返却
説　明	1 文字データをシリアルポートに送信します。		
補　足	-		

6.4.8 ferror

ferror ARM C/C++依存内部関数

ファイルのエラー状態の検知を行う 同期関数

ヘッダ	なし		
宣　言	int ferror(FILE * file_p);		
引　数	FILE * file_p	I	未使用
戻り値	0		
説　明	何もせず、0 を返します。		
補　足	機能実装していません（何もせず、0 を返します。）。		

6.4.9 SioRead

SioRead ARM C/C++依存内部関数

文字バッファのリード 同期関数

ヘッダ	sio_char.h		
宣　言	size_t SioRead(int handle, unsigned char * buffer, size_t size);		
引　数	int handle	I	“STDIN”（standard in）のみ指定可能
	unsigned char * buffer	I/O	リード・データ格納バッファのポインタ
	size_t size	I	受信要求サイズ（バイト）
戻り値	>= 0		成功：受信バイト数
	STDERR		失敗：受信エラー
説　明	シリアルポートからデータを受信し、buffer で指定されたアドレスに格納します。		
補　足	-		

6.4.10 SioWrite

SioWrite	ARM C/C++依存内部関数
文字バッファのライト	同期関数

ヘッダ	sio_char.h		
宣　言	size_t SioWrite(int handle, const unsigned char * buffer, size_t size);		
引　数	int handle	I	“STDOUT”（standard out）または” STDERR”（standard err）のみ指定可能
	const unsigned char * buffer	I	ライトデータ格納バッファのポインタ
	size_t size	I	送信要求サイズ（バイト）
戻り値	>= 0	成功：送信バイト数	
	STDERR	失敗：送信エラー	
説　明	シリアルポートからデータを送信します。		
補　足	-		

6.4.11 _ttywrch

_ttywrch	ARM C/C++依存内部関数
ARM C ライブラリ　端末への 1 文字出力	同期関数

ヘッダ	rt_sys.h		
宣　言	void _ttywrch(int ch);		
引　数	int ch	I	未使用（出力文字）
戻り値	なし		
呼び出し元	ARM C ライブラリ		
説　明	デバッグ・シリアルへ引数 ch を 1 文字出力します。		
補　足	機能実装していません（空関数です）。		

6.4.12 _sys_exit

_sys_exit	ARM C/C++依存内部関数
プログラムの終了	同期関数

ヘッダ	なし		
宣　言	void _sys_exit(int returncode);		
引　数	int returncode	I	未使用
戻り値	なし		
説　明	無限ループし、リターンしません。		
補　足	-		

6.5 CMSIS-RTOS RTX 依存内部関数

6.5.1 os_idle_demon

os_idle_demon	CMSIS-RTOS RTX 依存内部関数
CMSIS-RTOS RTX の Idle スレッド	スレッド

ヘッダ	RTX_Config.h
宣　言	void os_idle_demon (void);
引　数	なし
戻り値	なし
呼び出し元	CMSIS-RTOS RTX の最低優先度のスレッドとして常駐している CMSIS-RTOS RTX スケジューラが実行すべきスレッドが存在しない時に Running 状態となる
説　明	Idle 状態で実行される CMSIS-RTOS RTX のスレッドです
補　足	Running 状態になると、Cortex-R4F コアを WFI モードにします システム・タイマ等の割り込み発生により、CMSIS-RTOS RTX スケジューラが動作を再開します 周辺マクロの省電力動作を実装する場合、この関数をカスタマイズします もし、カスタマイズで省電力モードを実装する場合、動作モードを変える前後に os_suspend()と os_resume()を実行します

6.5.2 os_tick_init

os_tick_init	CMSIS-RTOS RTX 依存内部関数
CMSIS-RTOS RTX 周辺機能タイマの初期化	同期関数

ヘッダ	RTX_Config.h
宣　言	int　os_tick_init (void);
引　数	なし
戻り値	正数　：成功：初期化した周辺タイマの IRQ 番号
呼び出し元	osKernelStart()経由で実行する CMSIS-RTOS RTX の初期化処理
説　明	CMSIS-RTOS RTX のシステム・タイマに、Cortex-R4F コア内蔵タイマではなく周辺マクロのタイマを使用する場合のタイマ初期化関数です 本関数は、RZ/T1 内蔵の CMT4 を初期化します
補　足	CMSIS-RTOS RTX のコンフィギュレーションで、OS_TIMER の値が 1 の場合に使用されます CMSIS-RTOS RTX サンプルプログラムのデフォルト設定は、「OS_TIMER==1(周辺タイマを使用)」であり、この関数が使用されます

6.5.3 os_tick_val

os_tick_val	CMSIS-RTOS RTX 依存内部関数
CMSIS-RTOS RTX 周辺機能タイマ	同期関数

ヘッダ	RTX_Config.h
宣　言	uint32_t　os_tick_val(void);
引　数	なし
戻り値	経過クロックカウント（１チック内）
呼び出し元	osKernelSysTick 関数コールで呼ばれます
説　明	CMSIS-RTOS RTX のシステム・タイマに、Cortex-R4F コア内蔵タイマではなく周辺マクロのタイマを使用する場合のにこの関数を実装します。現在の CMSIS-RTOS RTX のシステム・タイマ値を返します。
補　足	CMT4.CMCNT の値を返すよう実装しています。 CMSIS-RTOS RTX のコンフィギュレーションで、OS_TIMER の値が 1 の場合に使用されます CMSIS-RTOS RTX サンプルプログラムのデフォルト設定は、「OS_TIMER==1(周辺タイマを使用)」であり、この関数が使用されます

6.5.4 os_tick_ovf

os_tick_ovf

CMSIS-RTOS RTX 依存内部関数

CMSIS-RTOS RTX 周辺機能タイマ　　同期関数

ヘッダ	RTX_Config.h	
宣　言	uint32_t　os_tick_ovf(void);	
引　数	なし	
戻り値	0	オーバーフローなし
	1	オーバーフロー発生
呼び出し元	osKernelSysTick 関数コールで呼ばれます	
説　明	CMSIS-RTOS RTX のシステム・タイマに、Cortex-R4F コア内蔵タイマではなく周辺マクロのタイマを使用する場合のにこの関数を実装します。CMSIS-RTOS RTX のシステム・タイマがオーバーフローを起こしたかの結果を返します。	
補　足	現在は無条件で 0（オーバーフローあり）を返します。 CMSIS-RTOS RTX のコンフィギュレーションで、OS_TIMER の値が 1 の場合に使用されます CMSIS-RTOS RTX サンプルプログラムのデフォルト設定は、「OS_TIMER==1(周辺タイマを使用)」であり、この関数が使用されます	

6.5.5 os_tick_irqack

os_tick_irqack

CMSIS-RTOS RTX 依存内部関数

CMSIS-RTOS RTX 周辺機能タイマの割り込みステータス・クリア　　同期関数

ヘッダ	RTX_Config.h	
宣　言	void os_tick_irqack (void);	
引　数	なし	
戻り値	なし	
呼び出し元	OS_Tick_Handler	
説　明	CMSIS-RTOS RTX のシステム・タイマに、Cortex-R4F コア内蔵タイマではなく周辺マクロのタイマを使用する場合のタイマ割り込みステータス・クリア処理を実装します 本関数は、RZ/T1 内蔵の CMT4 を操作します	
補　足	CMSIS-RTOS RTX のコンフィギュレーションで、OS_TIMER の値が 1 の場合に使用されます CMSIS-RTOS RTX サンプルプログラムのデフォルト設定は、「OS_TIMER==1(周辺タイマを使用)」であり、この関数が使用されます	

6.5.6 os_error

os_error

CMSIS-RTOS RTX 依存内部関数

CMSIS-RTOS RTX のランタイム・エラー発生時のハングアップ関数　　同期関数

ヘッダ	RTX_Config.h		
宣　言	void os_error (uint32_t err_code)		
引　数	Uint32_t err_code	I	以下の CMSIS-RTOS RTX のランタイム・エラー・コード OS_ERR_STK_OVF、OS_ERR_FIFO_OVF、OS_ERR_MBX_OVF
戻り値	なし		
呼出し元	CMSIS-RTOS RTX スケジューラが、ランタイム・エラーの発生時に呼び出されます		
説　明	CMSIS-RTOS RTX の致命的なランタイム・エラー発生時のハングアップ関数です		
補　足	CMSIS-RTOS RTX スケジューラから呼び出され、無限ループでシステムをハングアップ状態にします 致命的エラーであるため、システムを再起動することなく本関数を終了する事は禁止です		

6.5.7 OS_Tick_Handler

OS_Tick_Handler

CMSIS-RTOS RTX 依存内部関数

CMSIS-RTOS RTX 周辺機能タイマの ISR ハンドラ

同期関数

ヘッダ	なし
宣　言	void OS_Tick_Handler(uint32_t　IRQn);
引　数	uint32_t　IRQn　　I　未使用
戻り値	なし
呼び出し元	CMSIS-RTOS RTX のタイマ割込みに指定した、割込み関数内からコールされます
説　明	CMSIS-RTOS RTX のシステム・タイマに、Cortex-R4F コア内蔵タイマではなく周辺マクロのタイマを使用する場合のタイマ割り込みハンドラです。本関数は、RZ/T1 内蔵の CMT4 を操作します
補　足	CMSIS-RTOS RTX のコンフィギュレーションで、OS_TIMER の値が 1 の場合に使用されます CMSIS-RTOS RTX サンプルプログラムのデフォルト設定は、「OS_TIMER==1(周辺タイマを使用)」だり、この関数が使用されます

6.5.8 os_pendsv_init

os_pendsv_init

CMSIS-RTOS RTX 依存内部関数

CMSIS-RTOS RTX PendSV 割り込みの初期化

同期関数

ヘッダ	RTX_Config.h
宣　言	void os_pendsv_init (void);
引　数	なし
戻り値	なし
呼び出し元	osKernelInitialize
説　明	CMSIS-RTOS RTX に使用する PendSV 割り込みの初期化関数です 本関数は、RZ/T1 内蔵の CMT5 を初期化します
補　足	-

6.5.9 os_pendsv_irqack

os_pendsv_irqack

CMSIS-RTOS RTX 依存内部関数

CMSIS-RTOS RTX PendSV 割り込みステータス・クリア

同期関数

ヘッダ	RTX_Config.h
宣　言	void os_pendsv_irqack (void);
引　数	なし
戻り値	なし
呼び出し元	PendSV_Handler
説　明	CMSIS-RTOS RTX に使用する PendSV 割り込みのステータス・クリア処理を実装します 本関数は、RZ/T1 内蔵の CMT5 を操作します
補　足	-

6.5.10 os_pendsv

os_pendsv	CMSIS-RTOS RTX 依存内部関数
CMSIS-RTOS RTX PendSV 割込みハンドラの処理	同期関数

ヘッダ	なし
宣　言	void os_pendsv (void);
引　数	なし
戻り値	なし
呼び出し元	SVC_Handler あるいは osMailFree、osSignalSet、osMessagePut、osMessageGet、osSemaphoreRelease
説　明	CMSIS-RTOS RTX に使用する PendSV 割り込み処理です。 本関数は、RZ/T1 内蔵の CMT5 を操作します。
補　足	PendSV 割り込みは CMT5 を使用します。起動後に割り込み発生待ちのため 割り込み要求フラグ IRQ300 をポーリングします。

6.6 ユーティリティ関数

6.6.1 RZ_T1_RSK_InitClock

RZ_T1_RSK_InitClock	RZ/T1 評価ボード依存内部関数
内部クロック倍率設定	同期関数

ヘッダ	RZ_T1_RSK_Init.h		
宣　言	void RZ_T1_RSK_InitClock(void);		
引　数	なし		
戻り値	なし		
呼び出し元	SystemInit ()関数		
説　明	内部クロック倍率設定を行う		
	CPU クロック 450MHz		
補　足			

6.6.2 InterruptHandlerRegister

InterruptHandlerRegister	ユーティリティ関数
IRQ ハンドラ登録関数	同期関数

ヘッダ	system_Renesas_RZ_T1.h		
宣　言	uint32_t InterruptHandlerRegister (IRQn_Type irq, IRQHandler handler);		
引　数	IRQn_Type irq	I	IRQ 番号
	IRQHandler handler	I	ハンドラ関数へのポインタ
戻り値	0		IRQ ハンドラ登録成功
	1		引数 irq が設定可能な割り込み番号の範囲外
説　明	IRQ ハンドラを登録します		
	ハンドラ登録済みの IRQ 番号を指定した場合は新しいハンドラ関数で上書きします		
補　足	-		
使用例	InterruptHandlerRegister(OSTMI0TINT_IRQn, OS_Tick_Handler);		

6.6.3 InterruptHandlerUnregister

InterruptHandlerUnregister	ユーティリティ関数
IRQ ハンドラ登録解除関数	同期関数

ヘッダ	system_Renesas_RZ_T1.h		
宣　言	uint32_t InterruptHandlerUnregister (IRQn_Type irq)		
引　数	IRQn_Type irq	I	IRQ 番号
戻り値	0		IRQ ハンドラ登録解除成功
	1		引数 irq が設定可能な割り込み番号の範囲外
説　明	IRQ ハンドラの登録を解除します		
	ハンドラ未登録の IRQ 番号を指定した場合も正常終了します		
補　足			
使用例	InterruptHandlerUnregister(OSTMI0TINT_IRQn);		

6.7 ベクタ割り込み操作関数

6.7.1 VIC_EnableIRQ

VIC_EnableIRQ	ベクタ割り込み操作関数
ベクタ割り込みの有効	同期関数

ヘッダ	vic.h		
宣　言	void　VIC_EnableIRQ(IRQn_Type IRQn);		
引　数	IRQn_Type IRQn	I	ベクタ番号 1～300
戻り値	なし		
説　明	ベクタ番号の割り込みを有効にします。		
補　足	-		

6.7.2 VIC_DisableIRQ

VIC_DisableIRQ	ベクタ割り込み操作関数
ベクタ割り込みの無効	同期関数

ヘッダ	vic.h		
宣　言	void　VIC_DisableIRQ(IRQn_Type IRQn);		
引　数	IRQn_Type IRQn	I	ベクタ番号 1～300
戻り値	なし		
説　明	ベクタ番号の割り込みを無効にします。		
補　足	-		

6.7.3 VIC_SetDetectType

VIC_SetDetectType	ベクタ割り込み操作関数
割り込みのタイプの設定	同期関数

ヘッダ	vic.h		
宣　言	void VIC_SetDetectType(IRQn_Type IRQn, uint32_t detecttype);		
引　数	IRQn_Type IRQn	I	ベクタ番号 1～300
	uint32_t detecttype	I	0：レベル割込みに設定、1：エッジ割り込みに設定
戻り値	なし		
説　明	指定したベクタ番号の割り込みのタイプ（レベル割込み／エッジ割込み）を設定します。		
補　足	-		

6.7.4 VIC_SetPriority

VIC_SetPriority	ベクタ割り込み操作関数
ベクタ割り込みの優先度の設定	同期関数

ヘッダ	vic.h		
宣　言	void　VIC_SetPriority(IRQn_Type IRQn, uint32_t priority);		
引　数	IRQn_Type IRQn	I	ベクタ番号 1～300
引　数	uint32_t priority	I	割り込み優先度 0～31 ベクタ番号 1　～255 の時 : 0～15 256～300 の時 :16～31
戻り値	なし		
説　明	指定のベクタ番号の割り込みの優先度を設定します。		
補　足	-		

6.7.5 VIC_GetPriority

VIC_GetPriority

ベクタ割り込み操作関数

ベクタ割り込みの優先度の取得

同期関数

ヘッダ	vic.h		
宣　言	uint32_t VIC_GetPriority(IRQn_Type IRQn);		
引　数	IRQn_Type IRQn	I	ベクタ番号 1～300
戻り値	割込み優先度		成功時
	0xFFFFFFFF		失敗時
説　明	指定のベクタ番号の割り込みの優先度を取得します。		
補　足	-		

6.7.6 VIC_Init

VIC_Init

ベクタ割り込み操作関数

ベクタ割り込みの初期化

同期関数

ヘッダ	なし		
宣　言	void VIC_Init(void)		
引　数	なし		
戻り値	なし		
呼び出し元	VIC_Enable()		
説　明	ベクタ割り込みの初期化を行います。		
補　足	-		

6.7.7 VIC_Enable

VIC_Enable

ベクタ割り込み操作関数

ベクタ割り込みの有効化

同期関数

ヘッダ	vic.h		
宣　言	void VIC_Enable(void)		
引　数	なし		
戻り値	なし		
説　明	ベクタ割り込みを有効にします。		
補　足	-		

6.7.8 VIC_ClearPIC

VIC_ClearPIC

ベクタ割り込み操作関数

ベクタ割り込みの検出割り込みのクリア

同期関数

ヘッダ	vic.h		
宣　言	void VIC_ClearPIC(IRQn_Type IRQn);		
引　数	IRQn_Type IRQn	I	ベクタ番号 1～300
戻り値	なし		
説　明	ベクタ番号の割り込みをクリアします。		
補　足	-		

6.7.9 VIC_EndInterrupt

VIC_EndInterrupt

ベクタ割り込み操作関数

割り込み処理の終了

同期関数

ヘッダ	vic.h
宣　言	void VIC_EndInterrupt(void);
引　数	なし
戻り値	なし
説　明	必要な割込み終了処理を行います。
補　足	割り込みアドレスレジスタ（HVA0）に任意の値をライトします。 割り込みコントローラは割り込み処理が終了することを認識し、割り込みの優先レベルをクリアします。

7. サンプルコード

サンプルコードは、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。

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com

お問合せ先
http://japan.renesas.com/contact/

改訂記録	RZ/T1 グループ CMSIS-RTOS RTX for Cortex-R4 RTX サンプルプログラム アプリケーションノート (EWARM / ICCARM, e2studio/RenesasGCC, DS-5/ARMCC)

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.12.22	—	新規発行

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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

1. 未使用端子の処理

【注意】未使用端子は、本文の「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。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。

2. 電源投入時の処置

【注意】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. リザーブアドレスのアクセス禁止

【注意】リザーブアドレスのアクセスを禁止します。

アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。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。

4. クロックについて

【注意】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。

5. 製品間の相違について

【注意】型名の異なる製品に変更する場合は、事前に問題ないことをご確認下さい。

同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。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。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、
   各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通用信号機器、
   防災・防犯装置、各種安全装置等
   当社製品は、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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、使用することはできません。たとえ、意図しない用途に当社製品を使用したことによりお客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 本資料に記載されている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。また、当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。

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その総株主の議決権の過半数を直接または間接に保有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。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/contact/