RL78 ファミリ
ボードサポートパッケージモジュール
Software Integration System

要旨

Software Integration System（以下 SIS と称す）モジュールを使用するプロジェクトの基盤となるのがルネサスボードサポートパッケージ SIS モジュール（r_bsp）です。r_bsp は設定が簡単で、リセットから main()関数までに MCU と使用するボードが必要とする全てのコードを提供します。本ドキュメントでは、r_bsp の規約を説明し、その使用方法、設定方法、ご使用のボードに対応した BSP の作成方法を紹介します。

動作確認デバイス

- RL78/F23、RL78/F24 グループ
- RL78/G15 グループ
- RL78/G16 グループ
- RL78/G22 グループ
- RL78/G23 グループ

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

対象コンパイラ

- Renesas Electronics C/C++ Compiler Package for RL78 Family
- IAR C/C++ Compiler for Renesas RL78
- LLVM C/C++ Compiler for Renesas RL78

各コンパイラの動作確認内容については 7.1 動作確認環境を参照してください。

一部機能には制限があります。4.4 制限事項を参照してください。

目次

1. 概要

MCU を正しく設定するためには、ユーザアプリケーションを実行する前に、一連の作業を行う必要があります。必要な作業や量は使用する MCU によって異なります。一般的な例としては、スタックの設定、メモリの初期化、CPU/周辺ハードウェア・クロックの設定、ポートの端子の設定などがあります。これらの設定は本書で示す手順に沿って行う必要があります。r_bsp は、これらの設定を簡単に行えるように提供されるものです。

r_bsp は、ご使用の MCU がリセットからユーザアプリケーションの main()関数を開始するまでに必要な要素を提供します。また、r_bsp には、多くのアプリケーションで必要となる共通の機能も備えられています。それらの中には、クロックの発振/停止を設定するための関数や CPU/周辺ハードウェア・クロックの周波数を取得する関数などがあります。

すべてのアプリケーションでリセット後に必要な手順は同じですが、各設定の内容も同じというわけではありません。例えば、アプリケーションごとに、スタックサイズや使用するクロックが異なります。r_bsp の設定に関するオプションはコンフィグヘッダファイルに納められているので、簡単に設定オプションを変更することができます。

1.1 用語

用語	説明
プラットフォーム	ユーザの開発ボード。「ボード」を使用する場合もあり。
BSP	ボードサポートパッケージの略称。

1.2　ファイル構成

r_bsp のファイル構成を図 1.1 に示します。r_bsp フォルダの下に、3 つのフォルダと 2 つのファイルがあります。

doc フォルダには r_bsp のドキュメントが含まれます。

board フォルダには、generic フォルダがあります。

generic フォルダには、MCU ごとにフォルダがあります。

generic フォルダの構成を図 1.2 に示します。

mcu フォルダには、MCU ごとに 1 フォルダが含まれます。mcu フォルダには他に all フォルダがあり、このフォルダには r_bsp で全 MCU に共通のソースが含まれます。

platform.h は、ユーザが開発プラットフォームを選択するためのファイルで、ユーザプロジェクトに必要なすべてのヘッダファイルを board および mcu フォルダから選択します。これについては、後のセクションで詳しく説明します。

readme.txt には r_bsp に関する情報が要約されています。

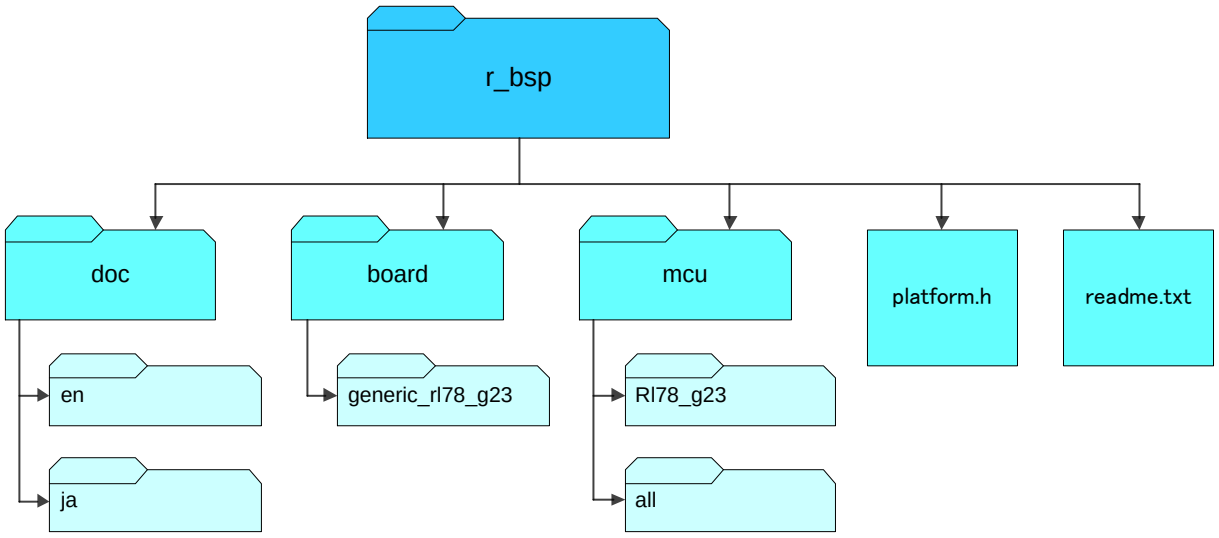


図 1.1　r_bsp ファイル構成

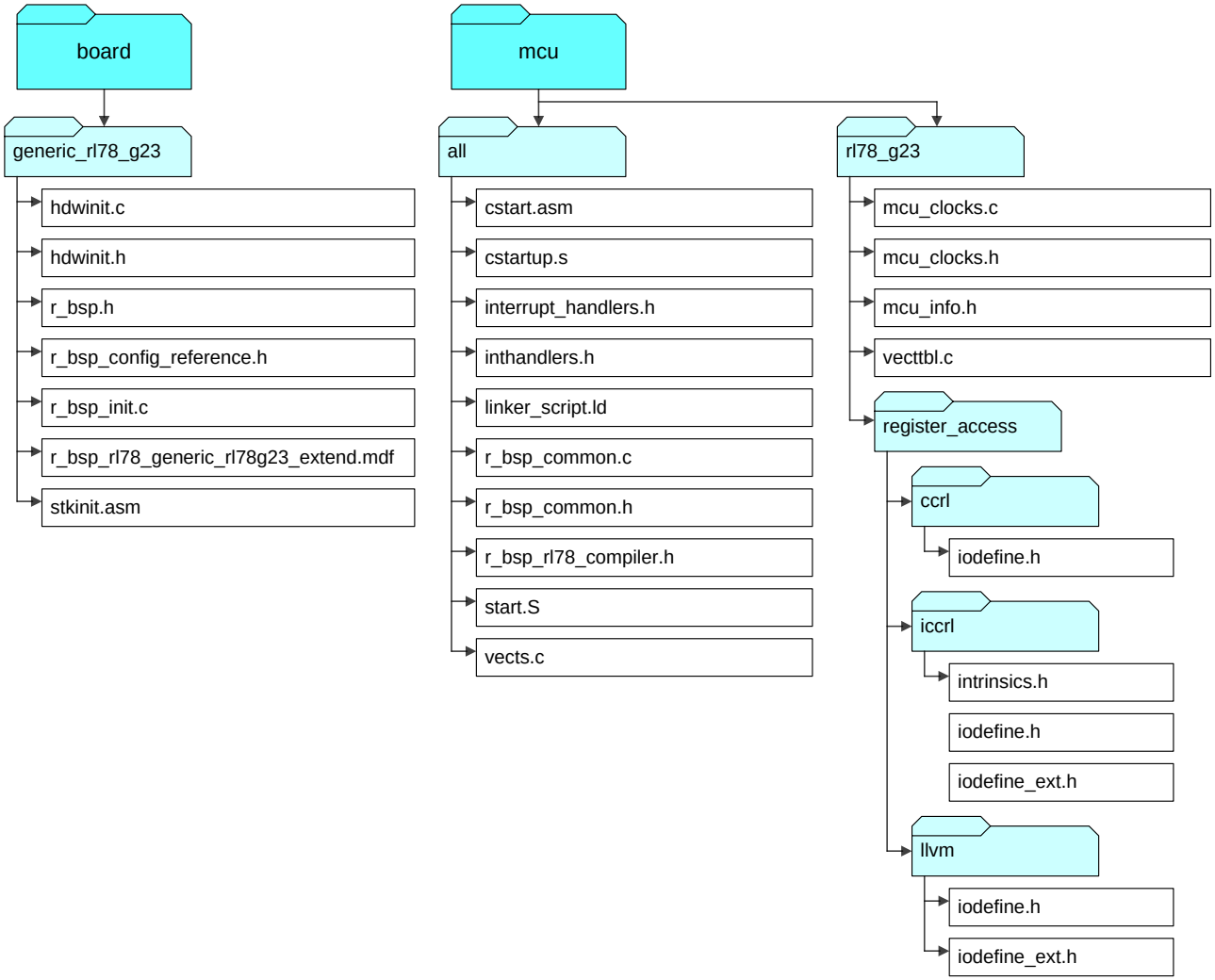


図 1.2　generic フォルダの構成

2. 機能

ここでは、r_bsp 搭載の機能について詳しく説明していきます。

2.1 MCU 情報

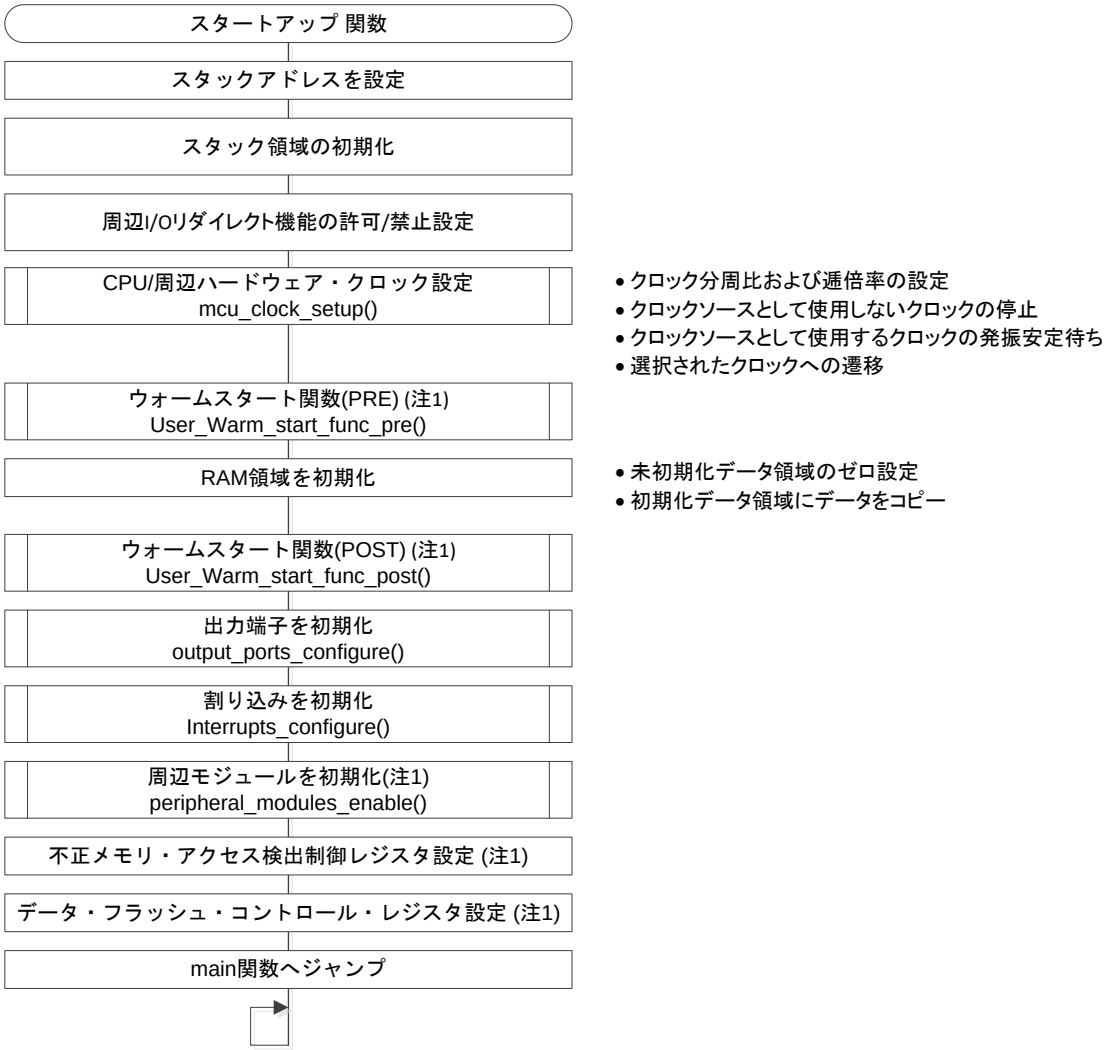
r_bsp の強みは、システム全体の設定をプロジェクトの 1 カ所で 1 度だけ定義すれば、その設定を共通で使用できるという点です。この情報は r_bsp で定義され、SIS モジュール、およびユーザコードで使用できます。SIS モジュールはこの情報を使って、自動的にコードをユーザのシステムに応じて設定します。r_bsp でこの情報が提供されなければ、ユーザが SIS モジュールごとに個別にシステム情報を設定する必要があります。

r_bsp の設定については、3 章で説明します。r_bsp はこの設定情報を使って、mcu_info.h のマクロ定義を設定します。MCU の mcu_info.h に存在するマクロの例を以下に示します。

定義	説明
BSP_MCU_FAMILY_<MCU_FAMILY>	この MCU が属している MCU ファミリ。たとえば、MCU が RL78/G23 であれば BSP_MCU_FAMILY_RL78 が定義されます。
BSP_MCU_SERIES_<MCU_SERIES>	この MCU が属している MCU シリーズ。たとえば、MCU が RL78/G23 であれば、BSP_MCU_SERIES_RL78G2X が定義されます。
BSP_MCU_GROUP_<MCU_GROUP>	この MCU が属している MCU グループ。たとえば、MCU が RL78/G23 であれば、BSP_MCU_GROUP_RL78G23 が定義されます。
BSP_<CLOCK>_HZ	MCU 上の各クロックについてこれらのマクロの 1 つが対応します。各マクロは、そのクロックの周波数を（ヘルツ単位で）定義します。たとえば、BSP_LOCO_HZ は、LOCO の周波数（Hz）を定義します。BSP_SUB_CLOCK_HZ は、サブシステム・クロック（Hz）を定義します。

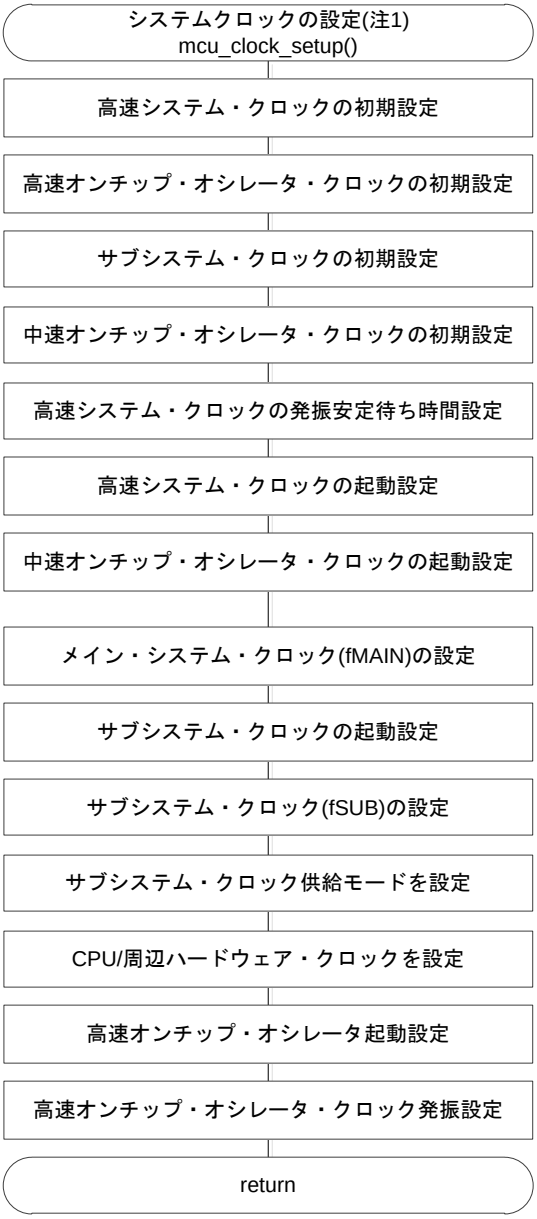
2.2　初期設定

Renesas コンパイラを使用する場合は_start 関数を、LLVM を使用する場合は PowerON_Reset 関数を MCU のリセットベクタとして設定します。IAR コンパイラを使用する場合は__iar_program_start 関数を MCU のリセットベクタとして設定します。_start と PowerON_Reset_PC と__iar_program_start 関数（スタートアップ関数）は、MCU がユーザアプリケーションを使用できる状態にするために、様々な初期化処理を行います。以下に、スタートアップ関数と CPU/周辺ハードウェア・クロック設定の動作をフローチャートで示します。



注1: r_bsp_config.hの設定によって、動作は異なります。

図 2.1　スタートアップ関数のフローチャート



注1: r_bsp_config.hの設定によって、動作は異なります。

図 2.2　CPU/周辺ハードウェア・クロック設定のフローチャート

2.3　グローバル割り込み

リセット解除後、割り込みは禁止になっています。必要に応じて割り込みを許可してください。

割り込みを許可する場合は”BSP_ENABLE_INTERRUPT”関数を、禁止する場合は”BSP_DISABLE_INTERRUPT”関数を使用してください。

詳細は 5.1 概要を参照してください。

RL78 のデバイスには固定ベクタテーブルがあります。固定ベクタテーブルはメモリマップ先頭の固定のロケーションに配置されます。

固定ベクタテーブルは iodefine.h に定義されます。

2.4　クロックの設定

CPU/周辺ハードウェア・クロックは r_bsp の初期化処理で設定されます。クロックは、r_bsp_config.h ファイルのユーザ設定に従って設定されます(3.2.8 参照)。クロックの設定は、C ランタイム環境の初期化処理の前に行われます。クロック選択時、r_bsp のコードによって、選択されたクロックが安定するのに要する遅延時間が取られます。

2.5　スタック領域

スタックはリセット後にスタートアップ関数内で設定、および初期化されます。

IAR コンパイラを使用する場合は GUI でスタックサイズを設定してください。

2.6　ID コード

RL78 MCU には ROM にある ID コードを使って、デバッガを介しての MCU メモリの読み出し、シリアルブートモードでの MCU メモリの読み出し、あるいはデバイスからのファームウェアの取り出しが行われないように保護します。ID コードは、オンチップ・デバッグ・セキュリティ ID 設定メモリに配置されます。セキュリティ ID の値は Renesas コンパイラ環境ではコンパイルオプションで設定します。IAR 環境または LLVM 環境では r_bsp_config.h で設定します。ID コードのオプションに関する詳細は、ユーザーズマニュアル ハードウェア編の「オプション・バイト」章や「オンチップ・デバッグ機能」章を参照ください。

2.7　オプション・バイト

RL78 MCU はフラッシュメモリにオプション・バイトが配置されています。電源投入時またはリセットからの起動時に、自動的にオプション・バイトを参照して、指定された機能の設定を行います。オプション・バイトにはウォッチドッグ・タイマの設定や電圧検出回路の設定などがあります。オプション・バイトに設定する値は、Renesas コンパイラ環境または LLVM 環境ではコンパイルオプションで設定します。IAR 環境では r_bsp_config.h で設定します(3.2.9 参照)。

2.8　RAM/SFR ガード機能

RL78 MCU には不正メモリ・アクセス検出制御レジスタがあり、指定した RAM 空間のデータおよびポート機能、割り込み機能、クロック制御機能、電圧検出回路、RAM パリティ・エラー検出機能の制御レジスタのデータを保護します。設定に使用する値は r_bsp_config.h で設定できます。

2.9　CPU 機能

割り込みの許可/禁止など、CPU 機能に関する設定を行う API 関数が用意されています。詳細は 5 章をご覧ください。

2.10　スタートアップ無効化

スタートアップを無効化する場合、スタートアップアセンブラコードを手動で削除してください。

環境とスタートアップアセンブラの関係は以下の通りです。

・Renesas コンパイラ　　：cstart.asm

・LLVM コンパイラ　　：start.S

・IAR コンパイラ　　：cstartup.s

また、ユーザスタートアップを追加してください。

2.10.1　スタートアップ無効化の設定方法

BSP のスタートアップ処理を無効化する場合は、以下の設定をしてください。

(1)コンフィグレーションファイルの設定

ユーザが作成したスタートアップ処理の内容を r_bsp_config.h に設定してください。BSP の API 関数や周辺の SIS モジュールは r_bsp_config.h の内容を参照する場合があります。ユーザが作成したスタートアップ処理の内容と r_bsp_config.h の内容に差異がある場合、SIS モジュールは正常に動作しない場合があります。

各周辺 SIS モジュールから参照される BSP の情報は、r_bsp_config.h から生成されるため、ユーザが作成したスタートアップ処理の内容と r_bsp_config.h に設定された内容を同じにする必要があります。

図 2.3 にコンフィグレーションファイルの設定を示します。

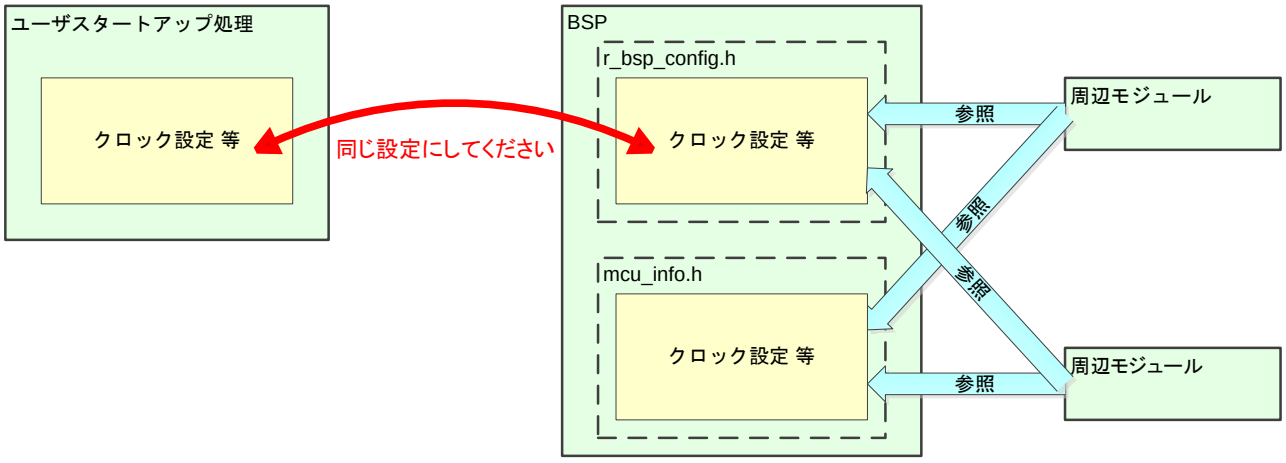


図 2.3　コンフィグレーションファイルの設定

3. コンフィギュレーション

r_bsp では、2 つのヘッダファイルを使って設定を行います。1 つはプラットフォームの選択を、他方は選択したプラットフォームの設定を行います。

3.1 プラットフォームを選択する

r_bsp は様々な MCU に対応するボードサポートパッケージを提供します。r_bsp フォルダの直下にある platform.h を変更して、使用するプラットフォームを選択します。

3.2 プラットフォームの設定

プラットフォームを選択したら、次にそのプラットフォームの設定を行わなければなりません。プラットフォームの設定は r_bsp_config.h を使って行います。プラットフォームごとに、r_bsp_config_reference.h というコンフィギュレーションファイルがあり、各プラットフォームの board フォルダに格納されています。

r_bsp_config.h の内容はそれぞれの MCU によって異なりますが、同じオプションも多数あります。以降のセクションでは、それらの設定オプションについて詳しく説明します。各マクロは共通で‘BSP_CFG_’から始まっており、検索や識別が簡単に行えます。

Smart Configurator を使用する場合は、ソフトウェアコンポーネント設定画面でコンフィグレーションオプションを設定できます。設定値はモジュールを追加する際に、自動的に r_bsp_config.h に反映されます。

3.2.1　MCU 製品型名情報

MCU の製品型名情報によって、MCU の様々な情報とともに r_bsp を提供できます。コンフィギュレーションファイルの先頭には、MCU の製品型名に関する情報が定義されます。これらのマクロ名はすべて‘BSP_CFG_MCU_PART’から始まります。MCU によって製品型名に関する情報量は異なりますが、以下に標準的な定義を示します。

表 3.1　製品型名の定義

定義	値	説明
BSP_CFG_MCU_PART_ROM_SIZE	r_bsp_config.h にて、#define の上にあるコメントを参照。	ROM のサイズを定義します。
BSP_CFG_MCU_PART_PIN_NUM		ピン数を定義します。
BSP_CFG_MCU_PART_HAS_DATA_FLASH		フラッシュメモリ搭載の有無を定義します。
BSP_CFG_MCU_PART_ROM_TYPE		デバイスの種類を定義します。

3.2.2　周辺 I/O リダイレクション・レジスタ

RL78 MCU には、兼用機能を割り当てるポートを切り替える機能があります。リセット後、r_bsp は、r_bsp_config.h の端子割り当て設定マクロを使って、MCU の端子割り当てを設定します。

表 3.2　周辺 I/O リダイレクション・レジスタの定義

定義	対応デバイス	値	説明
BSP_CFG_PIORx (x=0～5)	RL78/G22, RL78/G23	r_bsp_config.h にて、#define の上にあるコメントを参照。	兼用機能を割り当てるポートを定義します。 x の値は対象デバイスごとに異なります。 各定義の詳細は r_bsp_config.h をご参照ください。
BSP_CFG_PIORyy (yy=00～99)	RL78/F23, RL78/F24, RL78/G15 RL78/G16	r_bsp_config.h にて、#define の上にあるコメントを参照。	兼用機能を割り当てるポートを定義します。 yy の値は対象デバイスごとに異なります。 各定義の詳細は r_bsp_config.h をご参照ください。

3.2.3　RAM/SFR ガード機能

RL78 MCU には指定した RAM 空間のデータおよびポート機能、割り込み機能、クロック制御機能、電圧検出回路、RAM パリティ・エラー検出機能の制御レジスタのデータを保護する機能があります。リセット後、r_bsp は、r_bsp_config.h のガード機能設定マクロを使って、MCU のガード領域を設定します。

表 3.3　RAM/SFR ガード機能の定義

定義	値	説明
BSP_CFG_INVALID_MEMORY_ACCESS_DETECTION_ENABLE	r_bsp_config.h にて、#define の上にあるコメントを参照。	不正メモリ・アクセスの検出をするかしないかを定義します。
BSP_CFG_RAM_GUARD_FUNC		RAM ガード空間のサイズを定義します。
BSP_CFG_PORT_FUNCTION_GUARD		ポート機能の制御レジスタをガードするかしないかを定義します。
BSP_CFG_INT_FUNCTION_GUARD		割り込み機能のレジスタをガードするかしないかを定義します。
BSP_CFG_CHIP_STATE_CTRL_GUARD		クロック制御機能、電圧検出回路、RAM パリティ・エラー検出機能の制御レジスタをガードするかしないかを定義します。

3.2.4　RAM 開始アドレス

RL78 MCU には RAM の開始アドレスを変更できる機能があります。リセット後、r_bsp は、r_bsp_config.inc の RAMSAR アドレス定義と RAM 開始アドレス定義を使って、RAM の開始アドレスを設定します。

表 3.4　RAM/SFR ガード機能の定義

定義	値	説明
BSP_CFG_ASM_RAMSAR_ADDRESS	r_bsp_config.inc にて、設定	RAMSAR レジスタのアドレスを定義します。 CC-RL 環境では設定不要です。
BSP_CFG_ASM_RAM_GUARD_START_ADDRESS		RAM 開始アドレスを定義します。 RAMSAR レジスタを搭載していないデバイスでは、定義しないでください。

3.2.5　　データ・フラッシュのアクセス制限

RL78 MCU にはデータ・フラッシュへのアクセスを許可/禁止する機能があります。リセット後、r_bsp は、r_bsp_config.h のデータ・フラッシュのアクセス制限機能設定マクロを使って、データ・フラッシュへのアクセスを設定します。

表 3.5　データ・フラッシュのアクセス制限の定義

定義	値	説明
BSP_CFG_DATA_FLASH_ACCESS_ENABLE	r_bsp_config.h にて、#define の上にあるコメントを参照。	データ・フラッシュへのアクセスを許可するか禁止するかを定義します。

3.2.6　　RTOS(r_bsp_config.h)

使用するアプリケーションで RTOS を使用するかどうかを定義します。

リセット後、r_bsp は、r_bsp_config.h の RTOS 機能設定マクロを使って、RTOS を設定します。

表 3.6　RTOS(r_bsp_config.h)の定義

定義	値	説明
BSP_CFG_RTOS_USED	0=RTOS を使用しない 1=予約 2=予約 3=予約 4=Renesas ITRON を使用する	使用するアプリケーションで RTOS を使用するかどうかを定義します。 RTOS(Renesas ITRON)を使用する場合のみこの定義は有効になります。 r_bsp_config.inc 内の BSP_CFG_ASM_RTOS_USED と同じ値を設定してください。

3.2.7　　RTOS(r_bsp_config.inc)

使用するアプリケーションで RTOS を使用するかどうかを定義します。

リセット後、r_bsp は r_bsp_config.inc の RTOS 機能設定マクロを使って、RTOS を設定します。

表 3.7　RTOS(r_bsp_config.inc)の定義

定義	値	説明
BSP_CFG_ASM_RTOS_USED	0=RTOS を使用しない 1=予約 2=予約 3=予約 4=Renesas ITRON を使用する	使用するアプリケーションで RTOS を使用するかどうかを定義します。 RTOS(Renesas ITRON)を使用する場合のみこの定義は有効になります。 r_bsp_config.h 内の BSP_CFG_RTOS_USED と同じ値を設定してください。

3.2.8　　クロックの設定

RL78 MCU が使用できるクロックは MCU によって異なりますが、基本的な概念はすべてに共通です。リセット後、r_bsp は、r_bsp_config.h のクロック設定マクロを使って、MCU クロックを初期化します。

表 3.8　クロック設定の定義 (1/4)

定義	値	説明
BSP_CFG_HISYSCLK_SOURCE	0=ポート 1=水晶/セラミック発振子接続 2=外部クロック入力	高速システム・クロックの発振源を定義します。
BSP_CFG_HISYSCLK_OPERATION	(X1 発振モード時) 0=X1 発振回路動作 1=X1 発振回路停止 (外部クロック入力モード時) 0=EXCLK 端子からの 外部クロック有効 1=EXCLK 端子からの 外部クロック無効 (ポート・モード時) 0=入出力ポート 1=入出力ポート	高速システム・クロックの動作制御を定義します。
BSP_CFG_SUBCLK_SOURCE	0=入力ポート 1=水晶発振子接続 2=外部クロック入力	サブシステム・クロックの発振源を定義します。
BSP_CFG_SUBCLK_OPERATION	(XT1 発振モード時) 0=XT1 発振回路動作 1=XT1 発振回路停止 (外部クロック入力モード時) 0=EXCLKS 端子からの 外部クロック有効 1=EXCLKS 端子からの 外部クロック無効 (ポート・モード時) 0=入力ポート 1=入力ポート	サブシステム・クロックの動作制御を定義します。
BSP_CFG_MOCO_SOURCE BSP_CFG_MOCO_OPERATION	0=中速オンチップ・オシレータ停止 1=中速オンチップ・オシレータ動作	中速オンチップ･オシレータ･クロックの制御を定義します。
BSP_CFG_OCOCLK_SOURCE	0=高速オンチップ・オシレータ・クロック 1=中速オンチップ・オシレータ・クロック	メイン・オンチップ・オシレータ・クロック(f_{OCO})に使用するクロックソースを定義します。
BSP_CFG_MAINCLK_SOURCE	0=メイン・オンチップ・オシレータ・クロック(f_{OCO}) 1=高速システム・クロック(f_{MX})	メイン・システム・クロック(f_{MAIN})に使用するクロックソースを定義します。
BSP_CFG_SUBSYSCLK_SOURCE	0=サブ・クロック 1=低速オンチップ・オシレータ・クロック	サブシステム・クロックに使用するクロックソースを定義します。
BSP_CFG_FCLK_SOURCE	0=メイン・システム・クロック(f_{MAIN}) 1=サブシステム・クロック(f_{SUB})	CPU/周辺ハードウェア・クロック(f_{CLK})に使用するクロックソースを定義します。

表 3.8　クロック設定の定義 (2/4)

定義	値	説明
BSP_CFG_XT1_OSCMODE	0=低消費発振 1(デフォルト) 1=通常発振 2=低消費発振 2 3=低消費発振 3	XT1 発振回路の発振モードを定義します。
BSP_CFG_FMX_HZ	高速システム・クロック周波数(単位: Hz)	高速システム・クロックの周波数を定義します。
BSP_CFG_X1_WAIT_TIME_SEL	0=$2^8/f_X$ 1=$2^9/f_X$ 2=$2^{10}/f_X$ 3=$2^{11}/f_X$ 4=$2^{13}/f_X$ 5=$2^{15}/f_X$ 6=$2^{17}/f_X$ 7=$2^{18}/f_X$	X1 クロックの発振安定時間を定義します。
BSP_CFG_ALLOW_FSUB_IN_STOPHALT	0=周辺機能へのサブシステム・クロック供給許可 1=リアルタイム・クロック以外の周辺機能へのサブシステム・クロック供給停止	STOP モード時およびサブシステム・クロックで CPU 動作中の HALT モード時のサブシステム・クロックの供給を定義します。
BSP_CFG_ALLOW_FSL_IN_STOPHALT	0=周辺機能へのサブシステム・クロックまたは低速オンチップ・オシレータ供給許可 1=周辺機能へのサブシステム・クロックまたは低速オンチップ・オシレータ供給停止	STOP モード時およびサブシステム・クロックまたは低速オンチップ・オシレータで CPI 動作中の HALT モード時のクロック供給を定義します。
BSP_CFG_FIL_OPERATION	0=低速オンチップ・オシレータ停止 1=低速オンチップ・オシレータ発振	低速オンチップ・オシレータの動作制御を定義します。
BSP_CFG_RTC_OUT_CLK_SOURCE	0=サブシステム・クロック 1=低速オンチップ・オシレータ・クロック	リアルタイム・クロック、32 ビット・インターバル・タイマ、シリアル・インタフェース UART0,1、リモコン信号受信機能、クロック出力/ブザー出力制御回路の動作クロックを定義します。
BSP_CFG_HOCO_DIVIDE	r_bsp_config.h にて、#define の上にあるコメントを参照	高速オンチップ・オシレータの周波数を定義します。
BSP_CFG_WAKEUP_MODE	0=通常起動 1=高速起動	STOP モード解除および SNOOZE モード遷移時の高速オンチップ・オシレータ起動設定を定義します。
BSP_CFG_MOSC_DIVIDE	0=f_{MX} 1=$f_{MX}/2$ 2=$f_{MX}/4$ 3=$f_{MX}/8$ 4=$f_{MX}/16$	高速システム・クロックの分周比を定義します。
BSP_CFG_MOCO_DIVIDE	0=4MHz 1=2MHz 2=1MHz	中速オンチップ・オシレータの周波数を定義します。
BSP_CFG_FMP_DIVIDE	r_bsp_config.h にて、#define の上にあるコメントを参照	fMP クロックの分周比を定義します。

表 3.8　クロック設定の定義 (3/4)

BSP_CFG_PLL_DIVIDE	0=分周なし 1=2 分周 2=4 分周	PLL 周波数の分周設定を定義します。 PLL を使用しない場合、この定義は使用されないので無視して構いません。
BSP_CFG_PLL_OPERATION	0=PLL 発振停止 1=PLL 発振開始	PLL 発振設定を定義します。
BSP_CFG_FMAIN_DIVIDE	0=分周なし 1=2 分周(fMAIN=16MHz 入力の場合のみ) 2=4 分周(fMAIN=20MHz 入力の場合のみ)	PLL 入力クロックの分周設定を定義します。PLL を使用しない場合、この定義は使用されないので無視して構いません。
BSP_CFG_PLL_MULTI	0=12 逓倍 1=16 逓倍 2=10 逓倍 3=20 逓倍	PLL 逓倍率を定義します。PLL を使用しない場合、この定義は使用されないので無視して構いません。
BSP_CFG_PLL_MODE	0=クロックスルーモード (fMAIN) 1=PLL クロック選択モード (fPLL)	クロックモードの設定を定義します。
BSP_CFG_FPLL_HZ	PLL クロック周波数 (単位: Hz)	PLL クロックの周波数を定義します。
BSP_CFG_LOCKUP_WAIT_COUNT_SEL	0=128/fMAIN 1=256/fMAIN 2=512/fMAIN 3=1024/fMAIN	PLL ロックアップ待ちカウント設定を定義します。PLL を使用しない場合、この定義は使用されないので無視して構いません。
BSP_CFG_CAN_CLOCK_OPERATION	0=CAN X1 クロック(fX)供給停止 1=CAN X1 クロック(fX)供給許可	CAN X1 クロック(fX)の供給制御を定義します。
BSP_CFG_LIN1_CLOCK_SOURCE	0=fCLK 1=fMX	LIN1 通信に使用するクロック選択を定義します。
BSP_CFG_LIN1_CLOCK_OPERATION	0=LIN1 通信クロック供給停止 1=LIN1 通信クロック供給許可	LIN1 通信クロックの供給制御を定義します。
BSP_CFG_LIN0_CLOCK_SOURCE	0=fCLK 1=fMX	LIN0 通信に使用するクロック選択を定義します。
BSP_CFG_LIN0_CLOCK_OPERATION	0=LIN0 通信クロック供給停止 1=LIN0 通信クロック供給許可	LIN0 通信クロックの供給制御を定義します。
BSP_CFG_TRD_CLOCK_SOURCE	0=fCLK または fMP 1=fSL	TRDe に使用するクロック選択を定義します。
BSP_CFG_ADC_ENABLE	0=ADC クロック供給停止 1=ADC クロック供給許可	12 ビット A/D コンバータへのクロック供給制御を定義します。
BSP_CFG_ADCLK_DIVIDE	0=分周なし 1=2 分周 2=4 分周 3=8 分周	A/D コンバータに使用するクロックの分周比を定義します。
BSP_CFG_SUBWAITTIME	ループ回数(単位: 回)	サブシステム・クロック発振安定待ち時間を定義します。 メイン・システム・クロックを使用したループ回数を定義します。(注)

表 3.8　クロック設定の定義 (4/4)

BSP_CFG_FIHWAITTIME	ループ回数(単位: 回)	高速オンチップ・オシレータ・クロックの発振安定待ち時間を定義します。 メイン・システム・クロックを使用したループ回数を定義します。(注)
BSP_CFG_FIMWAITTIME	ループ回数(単位: 回)	中速オンチップ・オシレータ・クロックの発振安定待ち時間を定義します。 メイン・システム・クロックを使用したループ回数を定義します。(注)
BSP_CFG_FILWAITTIME	ループ回数(単位: 回)	低速オンチップ・オシレータ・クロックの発振安定待ち時間を定義します。 メイン・システム・クロックを使用したループ回数を定義します。(注)
BSP_CFG_PLLWAITTIME	ループ回数(単位: 回)	PLL 逓倍率設定の安定待ち時間を定義します。 メイン・システム・クロックを使用したループ回数を定義します。(注)
BSP_CFG_FIH_START_ON_STARTUP	0=高速オンチップ・オシレータ・クロックを停止する 1=高速オンチップ・オシレータ・クロックを起動する	初期化時の高速オンチップ・オシレータ・クロックの動作を定義する。

(注) ループ回数とは、nop 命令を 1 回実行する for 文によるループです。

実際のソースコードは以下の通りです。

```
/* WAIT_LOOP */
for (w_count = 0U; w_count <= BSP_CFG_SUBWAITTIME; w_count++)
{
    BSP_NOP();
}
```

ただし、実際のサイクル数は最適化オプションなどにより異なるため、ご使用の環境に合わせて設定してください。

3.2.9　オプション・バイト

オプション・バイトを設定して、リセット時の動作を選択できます。例えばウォッチドッグ・タイマや電圧検出回路の設定が行えます。

IAR 環境を使用する場合は r_bsp_config.h にオプション・バイトの設定値を定義してください。それ以外の環境ではプロジェクトのプロパティから設定してください。

表 3.9　オプション・バイトの定義

定義	値	説明
BSP_CFG_OPTBYTE0_VALUE BSP_CFG_OPTBYTE1_VALUE BSP_CFG_OPTBYTE2_VALUE BSP_CFG_OPTBYTE3_VALUE BSP_CFG_OPTBYTE4_VALUE	オプション・バイトの値	オプション・バイトに設定する値を定義します。 本マクロ定義は IAR 環境でのみ使用されます。 Renesas コンパイラまたは LLVM 環境ではコンパイルオプションで設定します。

3.2.10　オンチップ・デバッグ・セキュリティ ID

オンチップ・デバッグ・セキュリティ ID を設定して、第三者からメモリの内容を読み取られないようにすることができます。

IAR 環境を使用する場合は r_bsp_config.h にオンチップ・デバッグ・セキュリティ ID の設定値を定義してください。それ以外の環境ではプロジェクトのプロパティから設定してください。

表 3.10　オンチップ・デバッグ・セキュリティ ID の定義

定義	値	説明
BSP_CFG_SECUID0_VALUE BSP_CFG_SECUID1_VALUE BSP_CFG_SECUID2_VALUE BSP_CFG_SECUID3_VALUE BSP_CFG_SECUID4_VALUE BSP_CFG_SECUID5_VALUE BSP_CFG_SECUID6_VALUE BSP_CFG_SECUID7_VALUE BSP_CFG_SECUID8_VALUE BSP_CFG_SECUID9_VALUE BSP_CFG_SECUIDA_VALUE BSP_CFG_SECUIDB_VALUE BSP_CFG_SECUIDC_VALUE BSP_CFG_SECUIDD_VALUE BSP_CFG_SECUIDE_VALUE BSP_CFG_SECUIDF_VALUE	オンチップ・デバッグ・セキュリティ ID の値	オンチップ・デバッグ・セキュリティ ID に設定する値を定義します。 本マクロ定義は IAR 環境でのみ使用されます。 Renesas コンパイラまたは LLVM 環境ではコンパイルオプションで設定します。

3.2.11　スタートアップ無効化

表 3.11　スタートアップ無効化の定義

定義	値	説明
BSP_CFG_STARTUP_DISABLE	0 = BSP スタートアップ有効 1 = BSP スタートアップ無効	クロックの初期設定処理の有効/無効を定義します。無効を選択し場合、クロックの初期設定処理が無効になります。 スタートアップ全体を無効化する場合は、スタートアップアセンブラを手動で削除し、ユーザスタートアップを追加してください。

3.2.12　Smart Configurator

表 3.12　Smart Configurator の定義

定義	値	説明
BSP_CFG_CONFIGURATOR_SELECT	0=Smart Configurator を使用しない 1=Smart Configurator を使用する	Smart Configurator を現在のプロジェクトで使用するかどうかを定義します。 BSP_CFG_CONFIGURATOR_SELECT = 1 の場合は、Smart Configurator 初期化関数が呼び出されます。
BSP_CFG_CONFIGURATOR_VERSION	r_bsp_config.h にて、#define の上にあるコメントを参照。	使用している Smart Configurator のバージョンを定義します。

3.2.13　API 関数無効

表 3.13　API 関数無効の定義

定義	値	説明
BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE	0=API 関数有効 1=API 関数無効	API 関数(R_BSP_StartClock, R_BSP_StopClock)を使用するかどうかを定義します。 BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE=1 の場合は、API 関数を使用できませんが、メモリサイズを削減できます。
BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE		API 関数(R_BSP_GetFclkFreqHz)を使用するかどうかを定義します。 BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE=1 の場合は、API 関数を使用できませんが、メモリサイズを削減できます。
BSP_CFG_SET_CLOCK_SOURCE_API_FUNCTIONS_DISABLE		API 関数(R_BSP_SetClockSource)を使用するかどうかを定義します。 BSP_CFG_SET_CLOCK_SOURCE_API_FUNCTIONS_DISABLE=1 の場合は、API 関数を使用できませんが、メモリサイズを削減できます。
BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS_DISABLE		API 関数(R_BSP_ChangeClockSetting)を使用するかどうかを定義します。 BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS_DISABLE=1 の場合は、API 関数を使用できませんが、メモリサイズを削減できます。
BSP_CFG_SOFTWARE_DELAY_API_FUNCTIONS_DISABLE		API 関数(R_BSP_SoftwareDelay)を使用するかどうかを定義します。 BSP_CFG_SOFTWARE_DELAY_API_FUNCTIONS_DISABLE=1 の場合は、API 関数を使用できませんが、メモリサイズを削減できます。

3.2.14　パラメータチェック

表 3.14　パラメータチェックの定義

定義	値	説明
BSP_CFG_PARAM_CHECKING_ENABLE	0=パラメータ チェック無効 1=パラメータ チェック有効	パラメータチェックを有効にするかどうかを定義します。 fCLK ソース切り替え時に不正な設定に対してエラーを返します。

3.2.15　ウォームスタート時のコールバック機能

表 3.15　ウォームスタート時のコールバック機能の定義

定義	値	説明
BSP_CFG_USER_WARM_START_CALLBACK_PRE_INITC_ENABLED	0=C ランタイム環境を初期化する前にユーザ関数を呼び出さない 1=C ランタイム環境を初期化する前にユーザ関数を呼び出す	C ランタイム環境を初期化する前にユーザ関数を呼び出すかどうかを定義します。
BSP_CFG_USER_WARM_START_PRE_C_FUNCTION	C ランタイム環境が初期化される前に呼び出される関数	C ランタイム環境を初期化する前にユーザ関数を呼び出す際に呼び出される関数を定義します。
BSP_CFG_USER_WARM_START_CALLBACK_POST_INITC_ENABLED	0=C ランタイム環境を初期化した後にユーザ関数を呼び出さない 1=C ランタイム環境を初期化した後にユーザ関数を呼び出す	C ランタイム環境を初期化した後にユーザ関数を呼び出すかどうかを定義します。
BSP_CFG_USER_WARM_START_POST_C_FUNCTION	C ランタイム環境が初期化された後に呼び出される関数	C ランタイム環境を初期化した後にユーザ関数を呼び出す際に呼び出される関数を定義します。

3.2.16　ウォッチドッグ・タイマのリフレッシュ機能

表 3.16　ウォッチドッグ・タイマのリフレッシュ機能の定義

定義	値	説明
BSP_CFG_WDT_REFRESH_ENABLE	0=ウォッチドッグ・タイマを使用しない 1=ウォッチドッグ・タイマをウィンドウ・オープン期間100%で使用する 2=ウォッチドッグ・タイマをウィンドウ・オープン期間50%で使用する 3=ウォッチドッグ・タイマをウィンドウ・オープン期間75%で使用する	ウォッチドッグ・タイマの使用方法を定義します。 ウォッチドッグ・タイマのコンポーネント設定と同じ設定にしてください。
BSP_CFG_USER_WDT_REFRESH_INIT_FUNCTION	ウォッチドッグ・タイマのインターバル割り込み設定を行う関数	クロック設定を行う前にユーザ関数を呼び出す際に呼び出される関数を定義します。
BSP_CFG_USER_WDT_REFRESH_SETTING_FUNCTION	ウォッチドッグ・タイマのリフレッシュ許可フラグを設定する関数	クロック発振安定待ち中にウォッチドッグ・タイマのリフレッシュを許可するフラグを設定する関数を定義します。

4. API 情報

本ドライバ API はルネサスの API ネーミング規則に準じています。

4.1 ハードウェアの必要条件

適用外

4.2 ハードウェアリソースの必要条件

適用外

4.3 ソフトウェアの必要条件

なし

4.4 制限事項

4.4.1 IAR コンパイラに関する制限事項

IAR コンパイラではオプション・バイトの設定を r_bsp_config.h を使用して行ってください。

4.4.2 ウォッチドッグ・タイマのリフレッシュに関する制限事項

ウォッチドッグ・タイマのウィンドウ・オープン期間を 50%または 75%に設定した場合、リフレッシュタイミングはインターバル割り込みを前提としています。
インターバル割り込み以外のタイミングでリフレッシュを行わないでください。

4.5 サポートされているツールチェーン

本 SIS モジュールは「7.1 動作確認環境」に示すツールチェーンで動作確認を行っています。

4.6 使用する割り込みベクタ

本 SIS モジュールで使用する割り込みベクタはありません。

4.7 ヘッダファイル

platform.h を組み込むことによって、すべての API 呼び出しがインクルードされます。platform.h は本ドライバのプロジェクトコードと一緒に提供されます。

4.8 整数型

本プロジェクトは、コードをわかりやすく、移植可能なものとするために、ANSI C99 “Exact width integer types” を使用しています。これらの型は stdint.h に定義されます。

4.9　API Typedef

4.9.1　クロックリソース

この typedef は、R_BSP_StartClock()、R_BSP_StopClock()、R_BSP_SetClockSource()、R_BSP_ChangeClockSetting()関数で使用できるコマンドを定義します。

デバイスごとに使用可能なリソースが異なります。ユーザーズマニュアルまたは r_bsp_common.h を参照してください。

```
/* clock mode */
typedef enum
{
    HIOCLK,       // High-speed on-chip oscillator
    SYSCLK,       // High-speed system clock
    SXCLK,        // Subsystem clock
    MIOCLK,       // Middle-speed on-chip oscillator
    LOCLK,        // Low-speed on-chip oscillator
    PLLCLK,       // PLL clock
    ADCLK         // A/D conversion clcok
} e_clock_mode_t;
```

4.9.2　ソフトウェアディレイ単位

この typedef は、R_BSP_SoftwareDelay()関数で使用できる単位を定義します。

```
/* Available delay units. */
typedef enum
{
    BSP_DELAY_SECS = 0,         /* Requested delay amount is in seconds. */
    BSP_DELAY_MILLISECS,        /* Requested delay amount is in milliseconds. */
    BSP_DELAY_MICROSECS         /* Requested delay amount is in microseconds. */
} e_bsp_delay_units_t;
```

4.10　戻り値

4.10.1　エラーコード

この typedef は、R_BSP_StartClock()、R_BSP_StopClock()、R_BSP_SetClockSource()、R_BSP_ChangeClockSetting()関数から返すエラーコードを定義します。

```
/* Error identification */
typedef enum
{
    /* メンバは以下の表を参照ください。 */
} e_bsp_err_t;
```

メンバ	説明
BSP_OK	成功
BSP_ARG_ERROR	無効な引数が入力されました。
BSP_ERROR1	指定されたクロックが発振または停止していません。 関数によってエラー発生条件が異なります。
BSP_ERROR2	クロックのリソース切り替え中に、発振していないクロックリソースに切り替えられる可能性があります。
BSP_ERROR3	設定できない状態移行が指定されました。 状態移行の条件については、各デバイスのユーザーズマニュアルに記載されている CPU クロック状態移行図を参照してください。

4.11　コードサイズ

下表の値は下記条件で確認しています。本モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。RL78/F2x シリーズ、RL78/G1x シリーズ、RL78/G2x シリーズから代表して 1 デバイスずつ掲載しています。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「3 コンフィギュレーション」のコンフィギュレーションオプションによって決まります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_bsp v1.30

コンパイラバージョン: Renesas Electronics C Compiler Package for RL78 Family V1.11.00

LLVM C/C++ Compiler for Renesas RL78 10.0.0.202203

IAR C/C++ Compiler for Renesas RL78 version 4.21.3

ROM、RAM およびスタックのコードサイズ(RL78/F24)

コンパイラ	API 関数 (注 1)	クロック設定 (注 2)	ROM	RAM	STACK
Renesas compiler	無効	デフォルト	319	0	12
		全て有効	462	0	12
	有効	デフォルト	2093	0	62
		全て有効	2335	0	62
LLVM compiler (注 3)	無効	デフォルト	446	0	6
		全て有効	916	0	72
	有効	デフォルト	5186	0	286
		全て有効	5917	0	286
IAR compiler	無効	デフォルト	254	0	32
		全て有効	435	0	32
	有効	デフォルト	2344	0	66
		全て有効	2631	0	66

ROM、RAM およびスタックのコードサイズ(RL78/G15)					
コンパイラ	API 関数 （注 1）	クロック設定 （注 2）	ROM	RAM	STACK
Renesas compiler	無効	デフォルト	260	0	12
		全て有効	260	0	12
	有効	デフォルト	1020	0	62
		全て有効	1096	0	62
LLVM compiler (注 3)	無効	デフォルト	336	0	2
		全て有効	457	0	16
	有効	デフォルト	2625	0	286
		全て有効	2856	0	286
IAR compiler	無効	デフォルト	160	0	32
		全て有効	186	0	32
	有効	デフォルト	930	0	32
		全て有効	977	0	32

ROM、RAM およびスタックのコードサイズ(RL78/G23)					
コンパイラ	API 関数 （注 1）	クロック設定 （注 2）	ROM	RAM	STACK
Renesas compiler	無効	デフォルト	291	0	16
		全て有効	363	0	16
	有効	デフォルト	1568	0	62
		全て有効	1670	0	62
LLVM compiler (注 3)	無効	デフォルト	495	0	30
		全て有効	774	0	74
	有効	デフォルト	4142	0	286
		全て有効	4531	0	286
IAR compiler	無効	デフォルト	233	0	32
		全て有効	319	0	32
	有効	デフォルト	1684	0	66
		全て有効	1793	0	66

(注 1) r_bsp_config.h 内のマクロ定義 BSP_CFG_XXXX_API_FUNCTIONS_DISABLE を使用して有効/無効を切り替えます。
上記測定結果はすべてのマクロ定義を有効または無効に設定した場合の値です。
(注 2) デフォルトは Smart Configurator の初期設定です。
高速オンチップ・オシレータ・クロックのみ有効です。
(注 3) LLVM compiler を使用してスタックサイズを測定する場合、Compiler options に
”-fstack-size-section”を追加しています。

4.12　for 文、while 文、do while 文について

本モジュールでは、レジスタの反映待ち処理等で for 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

以下に記述例を示します。

```
for 文の例：
HIOSTOP = 0;
/* WAIT_LOOP */
for (w_count = 0U; w_count <= BSP_CFG_FIHWAITTIME; w_count++)
{
    BSP_NOP();
}

do while 文の例：
MSTOP = 0;
/* WAIT_LOOP */
do{
    tmp_stab_wait = OSTC;
    tmp_stab_wait &= STAB_WAIT;
}while(tmp_stab_wait != STAB_WAIT);
```

5. API 関数

5.1 概要

本モジュールでは、以下の関数を使用します。

関数	説明
R_BSP_StartClock	指定したクロックを発振する。
R_BSP_StopClock	指定したクロックの発振を停止する。
R_BSP_GetFclkFreqHz	CPU/周辺ハードウェア・クロックの周波数を返す。
R_BSP_SetClockSource	CPU/周辺ハードウェア・クロックのクロックソースを指定したクロックに変更する。
R_BSP_ChangeClockSetting	指定したクロックの設定を変更する。
R_BSP_SoftwareDelay	指定した時間だけ遅延させる。
BSP_DISABLE_INTERRUPT	すべてのマスカブル割り込みの受け付けを禁止します。本関数はマクロ関数です。
BSP_ENABLE_INTERRUPT	すべてのマスカブル割り込みの受け付けを許可します。本関数はマクロ関数です。
BSP_NOP	NOP 命令を実行します。本関数はマクロ関数です。

5.2　R_BSP_StartClock()

この関数は、指定したクロックを発振します。

Format

```
e_bsp_err_t R_BSP_StartClock(e_clock_mode_t mode);
```

Parameters

mode

発振するクロックを指定(4.9.1 参照)

Return Values

BSP_OK　/* 指定したクロックが正しく発振された */

BSP_ARG_ERROR　/* 無効な引数が入力された */

Properties

r_bsp_common.h にプロトタイプ宣言されています。

Description

本関数は、指定したクロックを発振させます。

本関数を使用して高速システム・クロックまたはサブシステム・クロックを発振させる場合、クロック動作モード制御レジスタ(CMC)が正しく設定されている必要があります。

例えば、本関数の引数に高速システム・クロックを指定した場合でも、EXCLK/OSCSEL がポートに設定されていた場合、高速システム・クロックを発振させることができません。

CMC レジスタはリセット解除後、1 回のみ書き込み可能であるため、高速システム・クロック、サブシステム・クロックを使用する場合は初期設定時に有効に設定してください。

Example

```
e_bsp_err_t err;

/* Start High-speed on-chip oscillator */
err = R_BSP_StartClock(HIOCLK);

if (err != BSP_OK)
{
    /* NG processing */
}
```

Special Note:

この関数は、マクロ定義(BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE)を 0 に設定した場合のみ使用できます。

5.3　R_BSP_StopClock()

この関数は、指定したクロックの発振を停止します。ただし、CPU/周辺ハードウェア・クロックとして動作しているクロックを停止した場合の動作は保証しません。

Format

```
e_bsp_err_t R_BSP_StopClock(e_clock_mode_t mode);
```

Parameters

mode

発振を停止するクロックを指定(4.9.1 参照)

Return Values

BSP_OK	/* 指定したクロックに対する停止処理が行われた */
BSP_ARG_ERROR	/* 無効な引数が入力された */

Properties

r_bsp_common.h にプロトタイプ宣言されています。

Description

本関数は、指定したクロックの発振を停止させます。

本関数は指定するクロックに対するエラーチェックを行わないため、CPU/周辺ハードウェア・クロックとして使用しているクロックを停止した場合の動作を保証しません。

Example

```
e_bsp_err_t err;

/* Stop High-speed on-chip oscillator */
err = R_BSP_StopClock(HIOCLK);

if (err != BSP_OK)
{
    /* NG processing */
}
```

Special Note:

この関数は、マクロ定義(BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE)を 0 に設定した場合のみ使用できます。

5.4　R_BSP_SetClockSource()

この関数は、CPU/周辺ハードウェア・クロックに供給するクロックリソースを変更します。

高速システム・クロックまたはサブシステム・クロックに変更する場合、初期設定時に同クロックを有効にしておく必要があります。

同クロックの制御を行うクロック動作モード制御レジスタ(CMC)はリセット解除後 1 回のみ書き込み可能です。

そのため初期設定時に無効に設定した場合、動作中に有効にすることができません。

Format

```
e_bsp_err_t R_BSP_SetClockSource(e_clock_mode_t mode);
```

Parameters

mode

CPU/周辺ハードウェア・クロックに供給するクロックリソースを指定(4.9.1 参照)

Return Values

BSP_OK	/* CPU/周辺ハードウェア・クロックが指定したクロックに切り替えられた */
BSP_ERROR1	/* 指定されたクロックが発振していない */
BSP_ERROR2	/* CPU/周辺ハードウェア・クロックのリソース切り替え中に、発振していないクロックリソースに切り替えられる可能性がある状態移行が指定された */
BSP_ERROR3	/* 設定できない状態移行が指定された */
BSP_ARG_ERROR	/* 無効な引数が入力された */

Properties

r_bsp_common.h にプロトタイプ宣言されています。

Description

本関数は、CPU/周辺ハードウェア・クロックに供給するクロックリソースを変更します。

Example

```
e_bsp_err_t err;

/* Start clock operation(HIOCLK) */
err = R_BSP_StartClock(HIOCLK);

if(err != BSP_OK)
{
    /* NG processing */
}
/* Change clock source */
err = R_BSP_SetClockSource(HIOCLK);

if (err != BSP_OK)
{
    /* NG processing */
}
```

Special Note:

この関数は、マクロ定義(BSP_CFG_SET_CLOCK_SOURCE_ API_FUNCTIONS_DISABLE)を 0 に設定した場合のみ使用できます。

クロックの切り替えについては、ユーザーズマニュアルの注意事項等をご確認の上ご使用ください。

5.5 R_BSP_GetFclkFreqHz()

この関数は、CPU/周辺ハードウェア・クロックの周波数を返します。

Format

```
uint32_t R_BSP_GetFclkFreqHz(void);
```

Parameters

なし

Return Values

CPU/周辺ハードウェア・クロックの周波数

Properties

r_bsp_common.h にプロトタイプ宣言されています。

Description

この関数は、CPU/周辺ハードウェア・クロックの周波数を返します。例えば、CPU/周辺ハードウェア・クロックが 20MHz になるように r_bsp_config.h を設定します。このとき、r_bsp によるクロック設定終了後、ユーザが CPU/周辺ハードウェア・クロックを 5MHz に変更した場合、関数の戻り値は’5000000’となります。

Example

```
uint32_t fclk_freq;

fclk_freq = R_BSP_GetFclkFreqHz();
```

Special Note:

この関数は、マクロ定義(BSP_CFG_GET_FREQ_ API_FUNCTIONS_DISABLE)を 0 に設定した場合のみ使用できます。

5.6　R_BSP_ChangeClockSetting()

この関数は、クロックの設定を変更します。

設定値は配列ポインタを引数に設定することで指定します。

デバイスおよびクロックリソースにより配列に格納する設定値が異なるため、以下のパラメータを参考に設定してください。

Format

```
e_bsp_err_t R_BSP_ChangeClockSetting(e_clock_mode_t mode, uint8_t *set_values);
```

Parameters

mode

設定を変更するクロックリソースを指定(4.9.1 参照)

set_values

変更する設定値を指定(以下参照)

各設定値の詳細については、ファイル r_bsp_config.h 内のコメントを参照。

(RL78/F23, RL78/F24)

mode に HIOCLK を指定した場合

set_values[0] : BSP_CFG_HOCO_DIVIDE のコメント参照

set_values[1] : BSP_CFG_FMP_DIVIDE のコメント参照

mode に SYSCLK を指定した場合

set_values[0] : BSP_CFG_FMP_DIVIDE のコメント参照

mode に PLLCLK を指定した場合

set_values[0] : 0 fPLL の周波数が 40MHz 以下の場合

1 fPLL の周波数が 40MHz より速い場合

set_values[1] : BSP_CFG_LOCKUP_WAIT_COUNT_SEL のコメント参照

set_values[2] : BSP_CFG_FMAIN_DIVIDE のコメント参照

set_values[3] : BSP_CFG_PLL_DIVIDE のコメント参照

set_values[4] : BSP_CFG_PLL_MULTI のコメント参照

set_values[5] : BSP_CFG_PLLWAITTIME のコメント参照

set_values[6] : BSP_CFG_FMP_DIVIDE のコメント参照

mode に ADCLK を指定した場合

set_values[0] : BSP_CFG_ADCLK_DIVIDE のコメント参照

(RL78/G15)

mode に HIOCLK を指定した場合

set_values[0] : BSP_CFG_HOCO_DIVIDE のコメント参照

(RL78/G16)

mode に HIOCLK を指定した場合

set_values[0] : BSP_CFG_HOCO_DIVIDE のコメント参照

(RL78/G22)

mode に HIOCLK を指定した場合

set_values[0] : BSP_CFG_HOCO_DIVIDE のコメント参照

mode に MIOCLK を指定した場合

set_values[0] : BSP_CFG_MOCO_DIVIDE のコメント参照

mode に SYSCLK を指定した場合

set_values[0] : BSP_CFG_MOSC_DIVIDE のコメント参照

(RL78/G23)

mode に HIOCLK を指定した場合

set_values[0] : BSP_CFG_HOCO_DIVIDE のコメント参照

mode に MIOCLK を指定した場合

set_values[0] : BSP_CFG_MOCO_DIVIDE のコメント参照

mode に SYSCLK を指定した場合

set_values[0] : BSP_CFG_MOSC_DIVIDE のコメント参照

Return Values

エラーが発生する原因はデバイスごとに異なります。(以下参照)

BSP_OK　　/* 指定したクロックの設定が変更された */

BSP_ERROR1

(RL78/F23, RL78/F24)

mode に HIOCLK を指定した場合

・fCLK に高速オンチップ・オシレータが供給されている状態で、高速オンチップ・オシレータが停止している状態

・fCLK に高速オンチップ・オシレータが供給されていない状態で、高速オンチップ・オシレータが発振している場合

・高速オンチップ・オシレータを入力とした PLL クロックが fCLK に供給されている場合

mode に ADCLK を指定した場合

・AD コンバータにクロックが供給されていない場合

・AD 変換が実行されている場合

・ガードビット(GCSC)が有効に設定されている場合

mode に上記以外のクロックを指定した場合

・指定したクロックが発振している場合

(RL78/G15)

mode に HIOCLK を指定した場合

・fCLK に高速オンチップ・オシレータが供給されている状態で、高速オンチップ・オシレータが停止している場合

・fCLK に高速オンチップ・オシレータが供給されていない場合

(RL78/G16)

mode に HIOCLK を指定した場合

・fCLK に高速オンチップ・オシレータが供給されている状態で、高速オンチップ・オシレータが停止している場合

・fCLK に高速オンチップ・オシレータが供給されていない場合

(RL78/G22)

mode に HIOCLK を指定した場合

・fCLK に高速オンチップ・オシレータが供給されている状態で、高速オンチップ・オシレータが停止している場合

・fCLK に高速オンチップ・オシレータが供給されていない場合

mode に上記以外のクロックを指定した場合

・指定したクロックが発振している場合

(RL78/G23)

mode に HIOCLK を指定した場合

・fCLK に高速オンチップ・オシレータが供給されている状態で、高速オンチップ・オシレータが停止している場合

・fCLK に高速オンチップ・オシレータが供給されていない場合

mode に上記以外のクロックを指定した場合

・指定したクロックが発振している場合

BSP_ARG_ERROR　　/* 無効な引数が入力された */

Properties

r_bsp_common.h にプロトタイプ宣言されています。

Description

本関数は、クロックの設定を変更します。

Example

```
e_bsp_err_t err;
uint8_t        set_values[2];

set_values[0] = 2U;
set_values[1] = 3U;

/* Stop clock(HIOCLK) */
err = R_BSP_StopClock(HIOCLK);

/* Change clock setting(HIOCLK) */
err = R_BSP_ChangeClockSetting(HIOCLK, set_values);

if(err != BSP_OK)
{
      /* NG processing */
}

/* Start clock(HIOCLK) */
err = R_BSP_StartClock(HIOCLK);
```

Special Note:

この関数は、マクロ定義(BSP_CFG_CHANGE_CLOCK_SETTING_ API_FUNCTIONS_DISABLE)を 0 に設定した場合のみ使用できます。

クロックの設定変更については、ユーザーズマニュアルの注意事項等をご確認の上ご使用ください。

5.7　R_BSP_SoftwareDelay()

この関数は、指定した単位の時間だけ遅延させてから戻ります。

Format

```
e_bsp_err_t R_BSP_SoftwareDelay(uint32_t delay, e_bsp_delay_units_t units);
```

Parameters

delay

遅延させる単位の数値

units

指定した単位のベース(4.9.2 参照)

Return Values

BSP_OK　　/* 遅延が実行された場合 */

BSP_ERROR1　　/* delay/units の組み合わせがオーバフロー/アンダフローになる場合 */

Properties

r_bsp_common.h にプロトタイプ宣言されています。

Description

本関数は、特定の待ち時間を実装するためにすべての MCU ターゲットに対して呼ぶことのできる関数です。

実際の遅延時間はオーバヘッドを考慮したものになります。オーバヘッドはコンパイラ、動作周波数、ROM キャッシュなどの影響で変化します。動作周波数が低い、または指定した単位時間がμ秒レベルの場合は誤差が大きくなるので、ご注意ください。

Example

```
e_bsp_err_t       ret;

/* Delay 5 seconds before returning */
ret = R_BSP_SoftwareDelay(5, BSP_DELAY_SECS);

if (BSP_OK != ret)
{
      /* NG processing */
}

/* Delay 5 milliseconds before returning */
ret = R_BSP_SoftwareDelay(5, BSP_DELAY_MILLISECS);

if (BSP_OK != ret)
{
      /* NG processing */
}

/* Delay 50 microseconds before returning */
ret = R_BSP_SoftwareDelay(50, BSP_DELAY_MICROSECS);

if (BSP_OK != ret)
{
      /* NG processing */
}
```

Special Note:

この関数は、マクロ定義(BSP_CFG_SOFTWARE_DELAY_ API_FUNCTIONS_DISABLE, BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE)を 0 に設定した場合のみ使用できます。

RL78/G15 デバイスと、LLVM コンパイラの組み合わせでは本関数を使用しないでください。

6. プロジェクトのセットアップ

ここでは r_bsp をプロジェクトに追加する方法を説明します。

6.1 SIS モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。

(1) e^2 studio 上で Smart Configurator を使用して SIS モジュールを追加する場合
e^2 studio の Smart Configurator を使用して、自動的にユーザプロジェクトに SIS モジュールを追加します。詳細は、「RL78 スマート・コンフィグレータ ユーザーガイド: e^2 studio 編 (R20AN0579)」を参照してください。

(2) CS+上で Smart Configurator を使用して SIS モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに SIS モジュールを追加します。詳細は、「RL78 スマート・コンフィグレータ ユーザーガイド: CS+編 (R20AN0580)」を参照してください。

(3) IAREW 上で Smart Configurator を使用して SIS モジュールを追加する場合
スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに SIS モジュールを追加します。詳細は、「RL78 スマート・コンフィグレータ ユーザーガイド: IAREW 編 (R20AN0581)」を参照してください。

6.2　e2studio プロジェクトへの SIS モジュールの追加方法

ここでは e2studio プロジェクトに SIS モジュールを追加する方法を説明します。

6.2.1　e2studio 上で Smart Configurator を使用して SIS モジュールを追加する方法

この説明では e2studio(2021-01)を使用しています。

(1) e2studio で新規プロジェクトを作成します。

プロジェクト作成時、「Use Smart Configurator」のチェックを ON にすることで Smart Configurator を起動します。

(2) 「6.1 SIS モジュールの追加方法」の手順で e2studio プロジェクトに SIS モジュールを追加します。

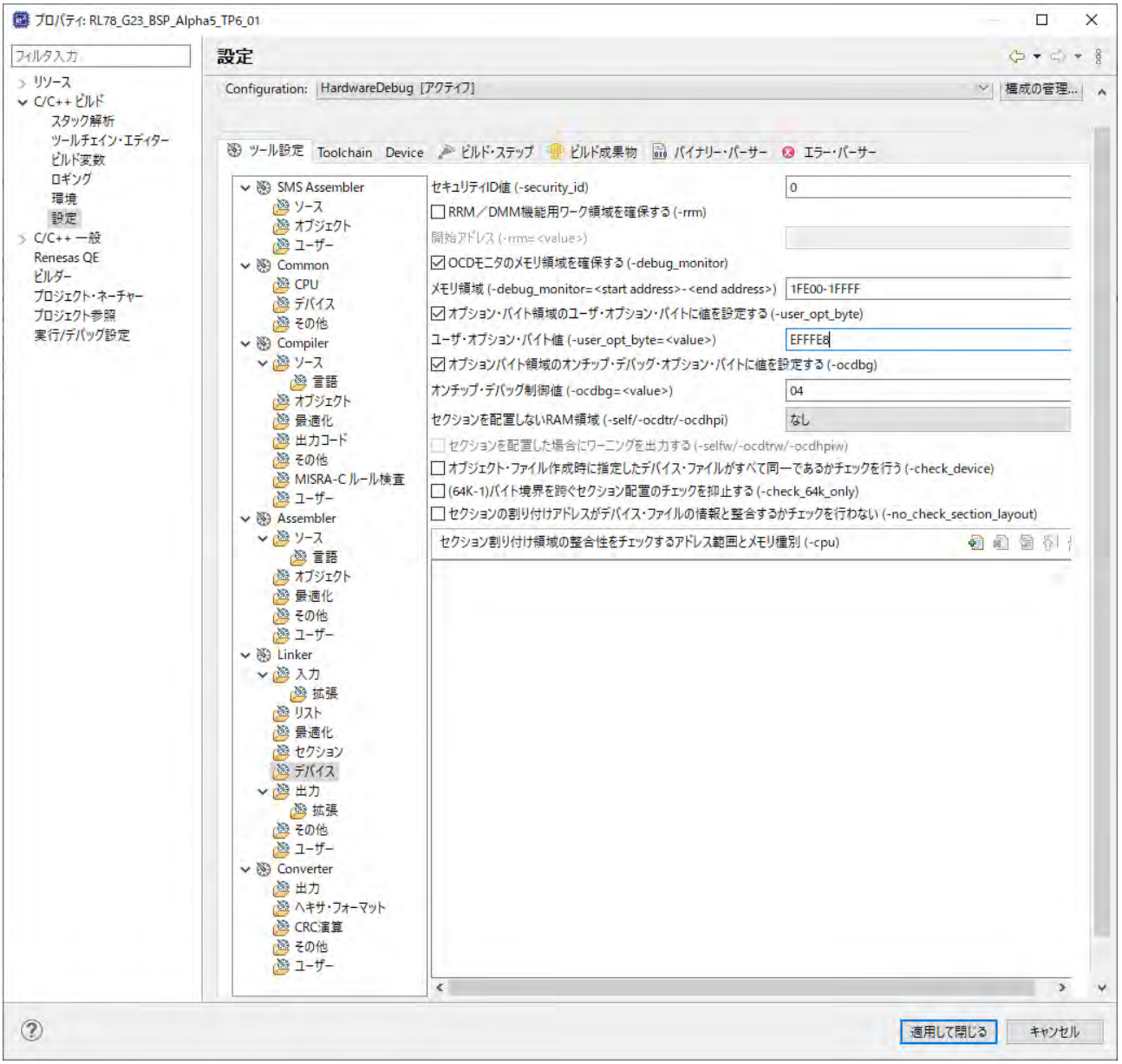
(3) プロジェクト上で右クリックをして、”プロパティ”をクリックします。

(4)　ツール設定タブの Compiler->ソースを選択します。

(5)　Smart Configulator で生成した SIS モジュールのインクルードパスが設定されています。

(6)　ツール設定タブの Linker->デバイスを選択します。

(7) オプション・バイト領域の設定を行う。



(8) プロジェクト上で右クリックをして、”プロジェクトのビルド”をクリックします。

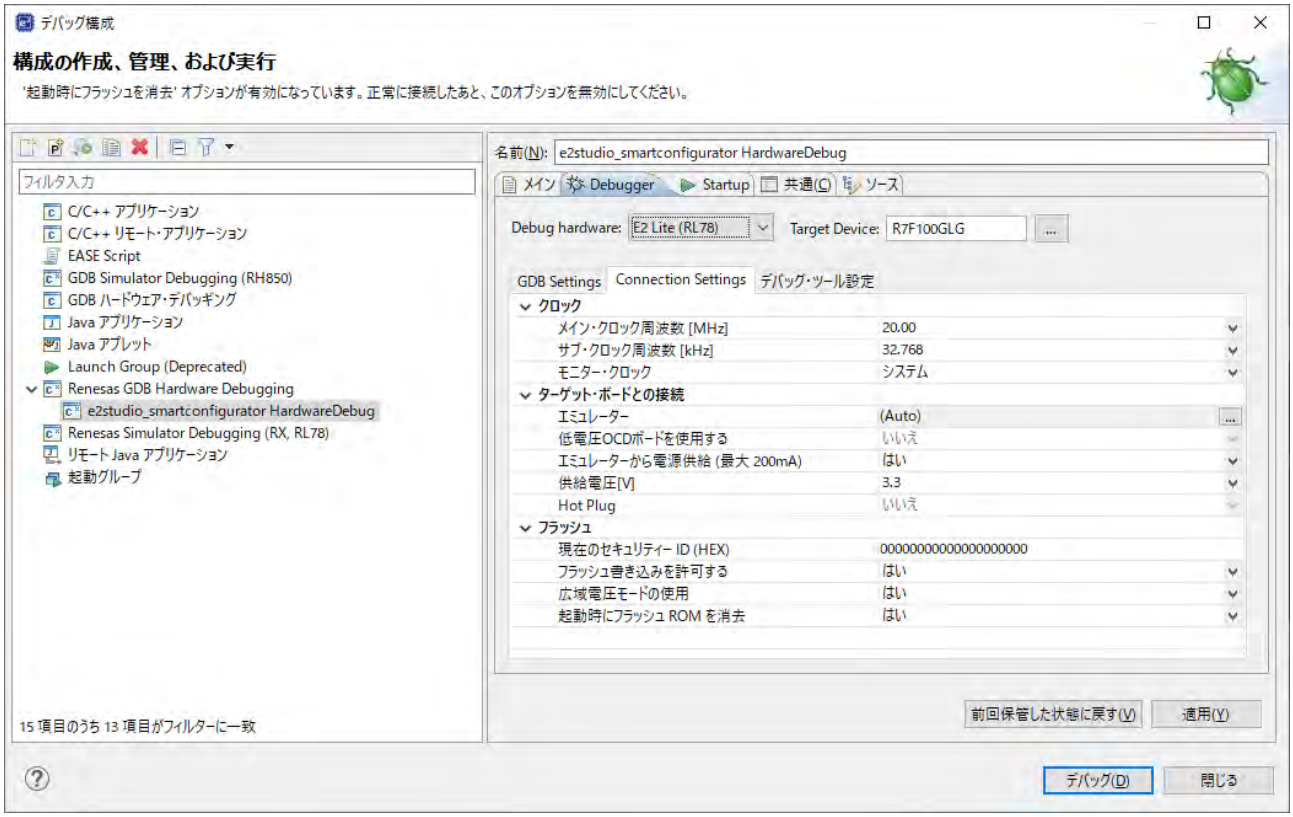
(9) プロジェクト上で右クリックして、デバッグ->デバッグの構成をクリックします。

(10) “Renesas GDB Hardware Debugging”->”プロジェクト名 HardwareDebug”をクリックします。

(11) Debugger タブで、Debug hardware に E2 Lite(RL78)を選択します。

(12) Connection Setting タグでメイン・クロック周波数とサブ・クロック周波数を設定します。

(13) Connection Setting タグでエミュレータからの電源供給を”はい”に設定します。



7. 付録

7.1 動作確認環境

本モジュールの動作確認環境を以下に示します。

表 7.1 動作確認環境 (Rev.1.00)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2021-01) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.20.1
C コンパイラ	ルネサスエレクトロニクス製 C compiler for R78 family V.1.09.0 LLVM for Renesas RL78 Build Support 0.1.0.v20200629-1555
モジュールのリビジョン	Rev.1.00
使用ボード	RL78/G23-64p Fast Prototyping Board(型名：RTK7RLG230CLG000BJ)

表 7.2 動作確認環境 (Rev.1.10)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2021-04) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.20.1
C コンパイラ	ルネサスエレクトロニクス製 C compiler for R78 family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.4.0.v20210325-1643
モジュールのリビジョン	Rev.1.10
使用ボード	RL78/G23-64p Fast Prototyping Board(型名：RTK7RLG230CLG000BJ)

表 7.3 動作確認環境 (Rev.1.11)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2021-04) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.20.1
C コンパイラ	ルネサスエレクトロニクス製 C compiler for R78 family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.4.0.v20210325-1643
モジュールのリビジョン	Rev.1.11
使用ボード	RL78/G23-64p Fast Prototyping Board(型名：RTK7RLG230CLG000BJ)

表 7.4 動作確認環境 (Rev.1.12)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2021-07) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.21.1
C コンパイラ	ルネサスエレクトロニクス製 C compiler for R78 family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.7.0.v20210630-0826
モジュールのリビジョン	Rev.1.12
使用ボード	RL78/G23-64p Fast Prototyping Board(型名：RTK7RLG230CLG000BJ)

表 7.5　動作確認環境 (Rev.1.13)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2021-10) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.21.1
C コンパイラ	ルネサスエレクトロニクス製 C compiler for R78 family V.1.10.0 GCC & LLVM for Renesas RL78 Build Support 21.7.0.v20210630-0826
モジュールのリビジョン	Rev.1.13
使用ボード	RL78/G23-64p Fast Prototyping Board(型名：RTK7RLG230CLG000BJ)

表 7.6　動作確認環境 (Rev.1.20)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2022-01) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.21.3
C コンパイラ	ルネサスエレクトロニクス製 C/C++ compiler for R78 family V.1.11.0 GCC & LLVM for Renesas RL78 Build Support 21.7.0.v20210630-0826
モジュールのリビジョン	Rev.1.20
使用ボード	RL78/F24 Target Board(型名：RTK7F124FPC0 1000BJ)

表 7.7　動作確認環境 (Rev.1.30)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2022-07) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.21.3
C コンパイラ	ルネサスエレクトロニクス製 C/C++ compiler for R78 family V.1.11.0 GCC & LLVM for Renesas RL78 Build Support 22.7.0.v20220419-1309
モジュールのリビジョン	Rev.1.30
使用ボード	RL78/G15 Target Board(型名：RTK5RLG150C00WS1BJ)

表 7.8　動作確認環境 (Rev.1.40)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2022-10) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.21.3
C コンパイラ	ルネサスエレクトロニクス製 C/C++ compiler for R78 family V.1.11.0 GCC & LLVM for Renesas RL78 Build Support 22.10.0.v20220621-1003
モジュールのリビジョン	Rev.1.40
使用ボード	RL78/G22 Target Board(型名：RTK7RLG220C00WS1BJ)

表 7.9　動作確認環境 (Rev.1.50)

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio (2023-01) IAR システムズ製 IAR Embedded Workbench for Renesas RL78 4.21.3
C コンパイラ	ルネサスエレクトロニクス製 C/C++ compiler for R78 family V.1.11.0 GCC & LLVM for Renesas RL78 Build Support 23.1.0.v20220922-1404
モジュールのリビジョン	Rev.1.50
使用ボード	RL78/G16 Target Board(型名：RTK5RLG160C00WS1BJ)

7.2 Rev1.30 から Rev1.40 へ更新時の API 関数の注意事項

API 関数を使用する場合、対応するマクロ定義を 0 に設定する必要があります。

マクロ定義は r_bsp_config.h に用意されています。

v1.30 以前のバージョンではマクロ定義のデフォルト値が全て 0(API 関数有効)ですが、v1.40 以降のバージョンではデフォルト値が BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE を除いて 1(API 関数無効)です。

デフォルト値を変更した理由は、デフォルト時のコードサイズを削減するためです。

7.3 Rev1.30 から Rev1.40 へ更新時の R_BSP_ChangeClockSetting 関数の注意事項

戻り値でエラーを返す条件を変更しています。

詳細については 5.6 R_BSP_ChangeClockSetting()を参照してください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Mar.08.2021	—	初版発行
1.10	Apr.22.2021	—	RTOS 対応を追加。
		15	RTOS のマクロ定義を追加 ・BSP_CFG_RTOS_USED
		19	オンチップ・デバッグ・セキュリティ ID の定義を追加 ・BSP_CFG_SECUID0_VALUE ・BSP_CFG_SECUID1_VALUE ・BSP_CFG_SECUID2_VALUE ・BSP_CFG_SECUID3_VALUE ・BSP_CFG_SECUID4_VALUE ・BSP_CFG_SECUID5_VALUE ・BSP_CFG_SECUID6_VALUE ・BSP_CFG_SECUID7_VALUE ・BSP_CFG_SECUID8_VALUE ・BSP_CFG_SECUID9_VALUE
		29	SIS モジュールを追加する際に参照するアプリケーションノートの名前を変更 ・RL78 スマート・コンフィグレータ ユーザーガイド: e^2 studio 編(R20AN0579) ・RL78 スマート・コンフィグレータ ユーザーガイド: CS+編(R20AN0580) ・RL78 スマート・コンフィグレータ ユーザーガイド: IAREW 編(R20AN0581)
		34	表 7.2 動作確認環境(Rev. 1.10)を追加
1.11	May.25.2021	—	マクロ定義を見直し
		12	バージョンによるプラットフォームの選択に関する記述を削除
		17,18,23	発振安定待ち時間のマクロ定義名を変更 ・BSP_CFG_SUBWAITTIME ・BSP_CFG_FIHWAITTIME ・BSP_CFG_FIMWAITTIME ・BSP_CFG_FILWAITTIME
		18	オプション・バイトのマクロ定義名を変更 ・BSP_CFG_OPTBYTE0_VALUE ・BSP_CFG_OPTBYTE1_VALUE ・BSP_CFG_OPTBYTE2_VALUE ・BSP_CFG_OPTBYTE3_VALUE
		34	表 7.3 動作確認環境(Rev1.11)を追加
1.12	Aug.04.2021	—	start.S にユーザ関数をコールする処理を追加 RL78/G23 iodefine.h(LLVM)を更新
		34	表 7.4 動作確認環境(Rev1.12)を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.13	Oct.29.2021	—	FSP 対応を追加 iodefine.h の名前を変更 MCU の属性を示すマクロ定義を追加 Smart Configurator のバージョンチェックを追加 ルネサスエレクトロニクス製コンパイラから C++を削除
		7	MCU の属性を示すマクロ定義を追加 ・BSP_MCU_FAMILY_<MCU_FAMILY> ・BSP_MCU_SERIES_<MCU_SERIES> ・BSP_MCU_GROUP_<MCU_GROUP>
		19	Smart Configurator のバージョンを示すマクロ定義を追加 ・BSP_CFG_CONFIGURATOR_VERSION
		35	表 7.5 動作確認環境(Rev1.13)を追加
1.20	Feb.28.2022	—	RL78/F23,F24 のサポートを追加 R_BSP_ChangeClockSetting 関数を追加 RAMSAR レジスタ設定機能を追加 API 関数の有効/無効切り替えを関数別に対応
		13	3.2.2 周辺 I/O リダイレクション・レジスタ 以下のマクロ定義を追加 ・BSP_CFG_PIORyy (yy=00～99)
		14	3.2.4 RAM 開始アドレス 章を追加し以下のマクロ定義を追加 ・BSP_CFG_ASM_RAMSAR_ADDRESS ・BSP_CFG_ASM_RAM_GUARD_START_ADDRESS
		16-19	3.2.8 クロックの設定 以下のマクロ定義を追加 ・BSP_CFG_ALLOW_FSL_IN_STOPHALT ・BSP_CFG_FIL_OPERATION ・BSP_CFG_PLL_DIVIDE ・BSP_CFG_FMAIN_DIVIDE ・BSP_CFG_CAN_CLOCK_OPERATION ・BSP_CFG_LIN1_CLOCK_SOURCE ・BSP_CFG_LIN1_CLOCK_OPERATION ・BSP_CFG_LIN0_CLOCK_SOURCE ・BSP_CFG_LIN0_CLOCK_OPERATION ・BSP_CFG_TRD_CLOCK_SOURCE ・BSP_CFG_LOCKUP_WAIT_COUNT_SEL ・BSP_CFG_PLL_MULTI ・BSP_CFG_PLL_MODE ・BSP_CFG_PLL_OPERATION ・BSP_CFG_FPLL_HZ ・BSP_CFG_FMP_DIVIDE ・BSP_CFG_ADC_ENABLE ・BSP_CFG_ADCLK_DIVIDE ・BSP_CFG_PLLWAITTIME
		19	3.2.9 オプション・バイト 以下のマクロ定義を追加 ・BSP_CFG_OPTBYTE4_VALUE

Rev.	発行日	改訂内容	
		ページ	ポイント
		20	3.2.10 オンチップ・デバッグ・セキュリティ ID 以下のマクロ定義を追加 ・BSP_CFG_SECUIDA_VALUE ・BSP_CFG_SECUIDB_VALUE ・BSP_CFG_SECUIDC_VALUE ・BSP_CFG_SECUIDD_VALUE ・BSP_CFG_SECUIDE_VALUE ・BSP_CFG_SECUIDF_VALUE
		21	3.2.13 API 関数無効 以下のマクロ定義を追加 ・BSP_CFG_CLOCK_OPERATION_API_FUNCTIONS_DISABLE ・BSP_CFG_GET_FREQ_API_FUNCTIONS_DISABLE ・BSP_CFG_SET_CLOCK_SOURCE_API_FUNCTIONS_DISABLE ・BSP_CFG_CHANGE_CLOCK_SETTING_API_FUNCTIONS_DISABLE
		25	4.9.1 クロックリソース クロックリソースを使用する関数に R_BSP_ChangeClockSetting 関数を追加
		26	4.11 コードサイズ コードサイズの測定結果を更新しました。
		28	5.1 概要 R_BSP_ChangeClockSetting 関数を追加
		29,30,32,33	API 関数の Special Note に使用できる条件を追加
		34,35	5.6 章 R_BSP_ChangeClockSetting を追加
		42	表 7.6 動作確認環境(Rev1.20)を追加
1.30	May.31.22	—	RL78/G15 のサポートを追加 R_BSP_SoftwareDelay 関数を追加
		21	3.2.13 API 関数無効 以下のマクロ定義を追加 ・BSP_CFG_SOFTWARE_DELAY_API_FUNCTIONS_DISABLE
		21	3.2.14 パラメータチェック 誤記訂正 BSP_CFG_PARAM_CHECKNIG_ENABLE　⇒ BSP_CFG_PARAM_CHECKING_ENABLE
		25	4.9.2 ソフトウェアディレイ単位 R_BSP_SoftwareDelay 関数用定義を追加
		29	5.1 概要 R_BSP_SoftwareDelay 関数を追加
		36	5.6 R_BSP_ChangeClockSetting() Parameters 欄に RL78/G15 で変更可能な設定値を追加
		37,38	5.7 章 R_BSP_SoftwareDelay()を追加
		45	表 7.7 動作確認環境(Rev1.30)を追加

Rev.	発行日	改訂内容	
		ページ	ポイント
1.40	Nov.11.22	—	RL78/G22 のサポートを追加 RL78/G15 のコードサイズを追加
		6	図 1.2 generic フォルダの構成 iccrl フォルダに格納されているファイルの名前を変更
		10	2.3 グローバル割り込み 固定ベクタテーブルの定義を iodefine.h に統一
		10	2.6 ID コード ID コードのサイズを削除
		16	3.2.8 クロックの設定 以下のマクロ定義を追加 ・BSP_CFG_MOCO_OPERATION
		27	4.11 コードサイズ RL78/G15 のコードサイズを追加
		38	5.6 R_BSP_ChangeClockSetting() Parameters 欄に RL78/G22 で変更可能な設定値を追加
		38	5.6 R_BSP_ChangeClockSetting() 戻り値でエラーを返す条件を変更
		48	表 7.8 動作確認環境(Rev1.40)を追加
		49	7.2 Rev1.30 から Rev1.40 へ更新時の API 関数の注意事項を追加
		49	7.3 Rev1.30 から Rev1.40 へ更新時の R_BSP_ChangeClockSetting 関数の注意事項を追加
1.50	Nov.30.22	—	RL78/G16 のサポート追加
		38	5.6 R_BSP_ChangeClockSetting() Parameters 欄に RL78/G16 で変更可能な設定値を追加
		39	5.6 R_BSP_ChangeClockSetting() Return values 欄に RL78/G16 でエラーを返す条件を追加
		49	表 7.9 動作確認環境(Rev1.50)を追加

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。