RX ファミリ
ボードサポートパッケージモジュール
Firmware Integration Technology

R01AN1685JJ0340
Rev.3.40
2016.10.01

要旨

Firmware Integration Technology（以下 FIT 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 FIT 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

- RX110、RX111、RX113 グループ
- RX130 グループ
- RX210、RX21A グループ
- RX220 グループ
- RX230、RX231、RX23T グループ
- RX24T グループ
- RX610 グループ
- RX62N、RX62T、RX62G グループ
- RX630、RX631、RX63N、RX63T グループ
- RX64M グループ
- RX71M グループ
- RX65N 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。

関連ドキュメント

- Firmware Integration Technology ユーザーズマニュアル (R01AN1833)

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、システムクロックの設定、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、例外処理用のコールバック関数や、割り込みの有効/無効を設定するための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すべて 1 つの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。

多くのユーザがルネサスの開発ボードを使って開発を行い、その後、独自のボード（ユーザボード）に移行されます。ユーザボードに移行される際には、r_bsp 内にユーザ仕様の BSP（カスタム BSP）を作成してください。提供される BSP と同様の規格や規則に従ってカスタム BSP を作成することによって、これから開発を行うボードにアプリケーションコードを簡単に移行できるので、開発をスムーズに開始できます。カスタム BSP の作成方法も本書で説明しています。

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。各ボードがそのボードに対応したソースファイルを持つ。
コールバック関数	イベント発生時に呼び出される関数のこと。例えば、バスエラー割り込み処理が r_bsp に組み込まれている場合、r_bsp にコールバック関数を持たせて、バスエラーの発生をユーザに通知することができる。バスエラー発生時、r_bsp は用意されたコールバック関数にジャンプし、ユーザはエラー処理が行える。 割り込みによってコールバック関数が呼び出されるとき、割り込みの処理内でコールバック関数を実行しているため、待機中の他の割り込みを遅延させる。そのため、割り込みによるコールバック関数では最低限の処理にし、かつ慎重に扱われなければならない。

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 フォルダにはボードごとに 1 フォルダが含まれ、各ボードのフォルダには、そのボードに対応したソースファイルが含まれます。board フォルダには他に、user フォルダがあります。これは任意のフォルダで、例えばユーザボード用フォルダとして使用できます。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。board フォルダがボード特定のソースファイルを有するのに対し、mcu フォルダは同じ MCU グループの MCU 間で共有できるソースを有します。例えば、ユーザが RX63N グループの 2 種類のボードを使用する場合、ボードごとに board フォルダが作成されます（例：「board >> my_board_1」および「board >> my_board_2」）が、これら 2 つのボードは同じ mcu フォルダを共有します（例：mcu >> rx63n）。これら 2 つの RX63N MCU のパッケージやメモリサイズが異なる場合でも、同じ mcu フォルダを共有します。

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。

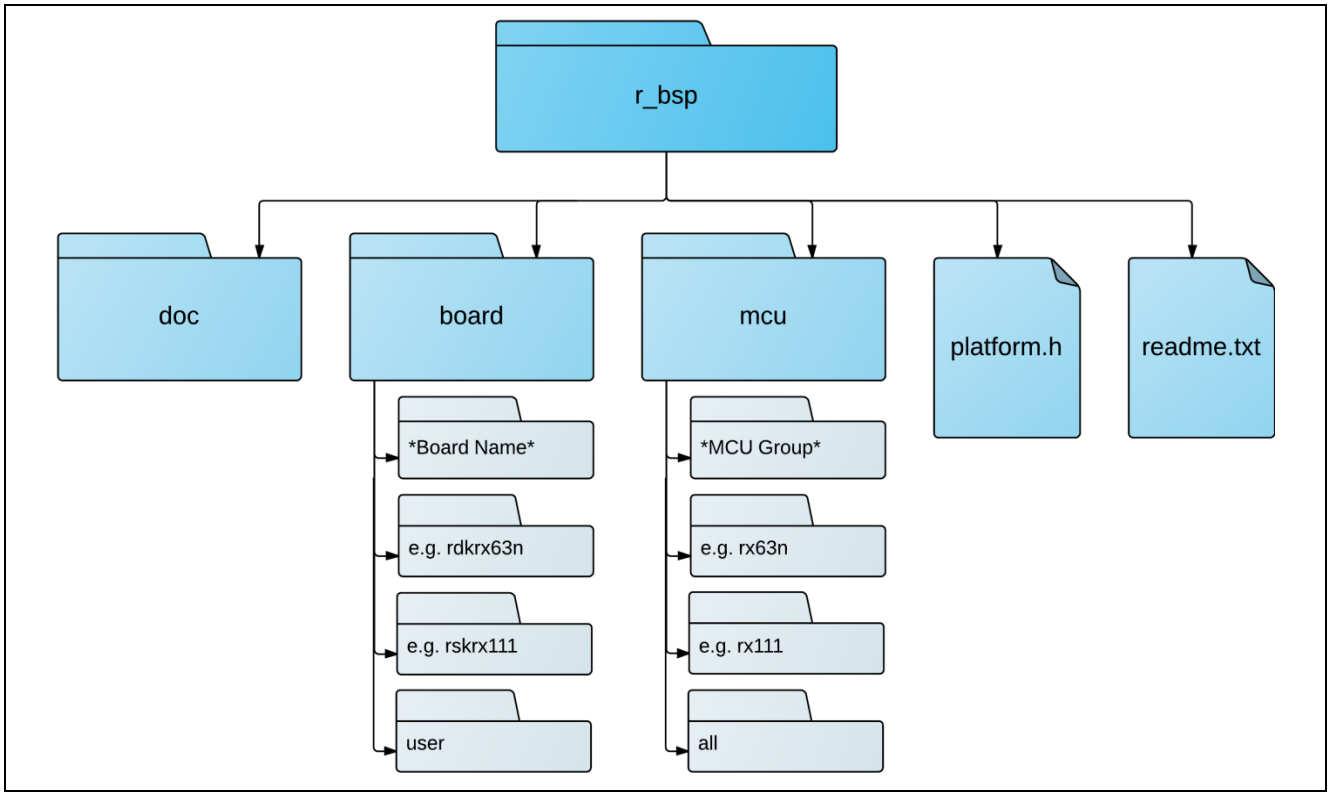


図 1.1　r_bsp ファイル構成

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、FIT モジュール、およびユーザコードで使用できます。FIT 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 FIT 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。各 MCU の mcu_info.h には、さまざまなマクロが存在する可能性がありますが、共通したいくつかの例を以下に示します。

定義	説明
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 RX62N、RX62T、RX630、RX63N などであれば、BSP_MCU_SERIES_RX600 が定義されます。
BSP_MCU_<MCU_GROUP>	この MCU が属している MCU グループ。たとえば、MCU が RX111 であれば、BSP_MCU_RX111 が定義されます。
BSP_PACKAGE_<PACKAGE_TYPE>	MCU のパッケージ。たとえば、MCU が 100 ピン LQFP パッケージの場合には、BSP_PACKAGE_LQFP100 が定義されます。
BSP_PACKAGE_PINS	この MCU のピン数
BSP_ROM_SIZE_BYTES	ユーザアプリケーション ROM 空間のサイズ（バイト）
BSP_RAM_SIZE_BYTES	ユーザが利用可能な RAM のサイズ（バイト）
BSP_DATA_FLASH_SIZE_BYTES	データフラッシュ領域のサイズ（バイト）
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_ICLK_HZ は、CPU クロック（Hz）を定義します。BSP_PCLKB_HZ は、周辺クロック B（Hz）を定義します。
BSP_MCU_IPL_MAX	MCU の最高割り込み優先レベル
BSP_MCU_IPL_MIN	MCU の最低割り込み優先レベル
FIT_NO_FUNC および FIT_NO_PTR	これらのマクロは、関数呼び出しの引数として使用することで、引数として何も提供されないことを示すことができます。たとえば、ある関数がコールバック関数用にオプションの引数を取る場合で、ユーザがコールバック関数を提供するつもりがなければ、FIT_NO_FUNC を使用することができます。これらのマクロは、リザーブアドレス空間を示すように定義されます。これにより、引数が誤って使用された場合に容易にこれを見つけることができます。つまり、MCU がリザーブ空間へのアクセスを試みた場合にバスエラーが生じるため、ユーザが直ちにこれを把握できるということです。代わりに NULL を使用した場合には、バスエラーは生じません。NULL は一般的に 0 と定義されますが、この値は RX 上の有効な RAM 位置であるからです。

2.2　初期設定

resetprg.c ファイルに含まれる PowerON_Reset_PC()関数を MCU のリセットベクタとして設定します。この関数は、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、PowerON_Reset_PC()関数の動作をフローチャートで示します。

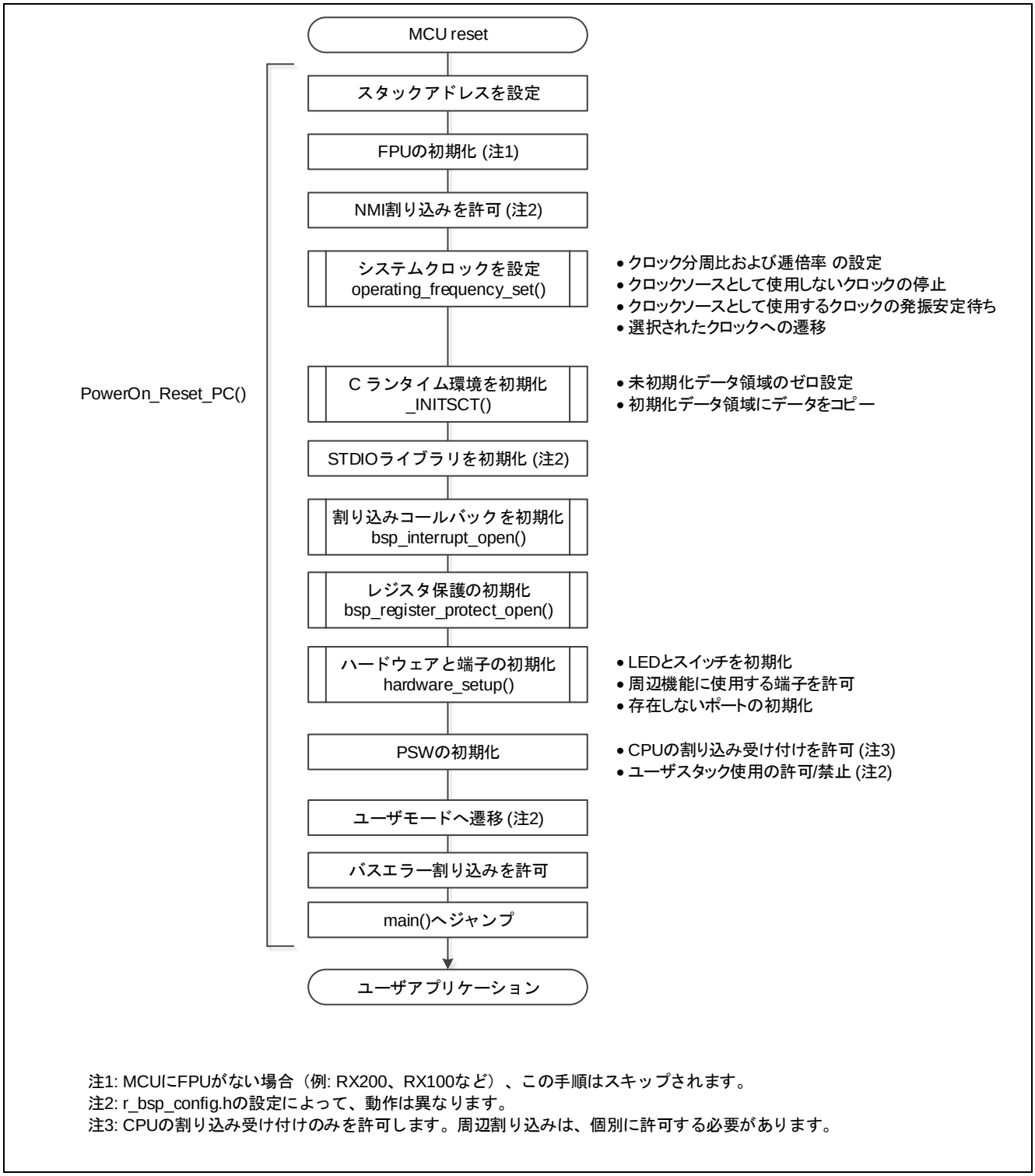


図 2.1　PowerON_Reset_PC()フローチャート

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。PowerON_Reset_PC()関数は、ユーザアプリケーションが呼び出される前に、割り込みを許可にします（2.2 参照）。

RX デバイスには可変ベクタテーブルと固定ベクタテーブルの 2 種類があります。可変ベクタテーブルはメモリのどこにでも配置することができますが、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。

可変ベクタテーブルでは、周辺機能の割り込みベクタが保持され、INTB レジスタによってポインタが指定されます。INTB レジスタは、PowerON_Reset_PC()関数にて、リセット後に初期化されます。可変ベクタテーブルのベクタは RX ツールチェーンによって挿入されます。RX ツールチェーンは、ユーザコードの‘#pragma interrupt’命令を使用して、ユーザの割り込みベクタ情報を取得します。

固定ベクタテーブルは、フラッシュ関連のオプションレジスタや、例外ベクタ、リセットベクタを保持します。固定ベクタテーブルは vecttbl.c で定義されます。vecttbl.c では、すべての例外、NMI 割り込み、バスエラー、未定義割り込みなどを処理するデフォルトの割り込み処理も定義されます。これらのベクタに対して、mcu_interrupts.c の機能を使ってコールバック関数を設定できます（2.4 参照）。また、ユーザブートリセットベクタが使用可能な状況であれば、vecttbl.c で設定が行えます。

固定ベクタテーブルのベクタはすべて vecttbl.c で操作されます。可変ベクタテーブルのベクタは、ユーザがベクタを定義すること、アプリケーションはそれぞれ異なることから、すべてが vecttbl.c で操作されるとは言えません。そのため、各アプリケーションのベクタがすべて埋まっているわけではないので、不用意に割り込み要求が生成されないように気を付けなければなりません。リンカの多くは、静的関数を使って、未使用のベクタを埋めることができます。これを行うために、vecttbl.c で undefined_interrupt_source_isr()関数が用意されており、この関数のアドレスを使ってリンカを設定し、未使用のベクタを埋めることが推奨されます。

2.4　割り込みコールバック

r_bsp では、複数の API 関数が用意されており、これらは割り込み要求生成のタイミングでユーザに通知されます（5.13、5.14 参照）。これは、ユーザが割り込みを選択し、それに対してコールバック関数を提供することによって行われます。割り込み要求が生成されると、r_bsp は提供されたコールバック関数を呼び出します。

ユーザは、固定ベクタテーブルにあるすべての例外割り込み、バスエラー割り込み、未定義割り込みに対応するコールバックを登録できます。ユーザ設定のコールバック関数が実行されると、r_bsp の割り込み処理は、必要に応じて割り込みフラグをクリアします。

2.5　存在しないポート端子

MCU グループには、ピン数の異なる様々なパッケージがあります。最大のピン数を持たないパッケージ（例: 最大 144 ピンの MCU グループで、64 ピンパッケージ）を使用する場合、接続されない端子の消費電力が低く抑えられるように初期化できます。r_bsp_config.h の設定をもとに、MCU の初期化処理中に、r_bsp が自動的にこれらの端子を初期化します。この機能は mcu_init.c 関数に組み込まれており、hardware_setup()によって呼び出されます。

2.6　クロックの設定

すべてのシステムクロックは 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（3.2.6 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。例えば、RX63x は 125 kHz の低速オンチップオシレータで起動するなど、RX MCU の中には、相対的に低速のクロックで起動するものがあります。このような MCU の起動を速めるために、クロックの設定が行われます。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。

一部の RX MCU では、フラッシュメモリや RAM にアクセスするためにウェイトサイクルの設定が必要です。アクセスするためのウェイトサイクルは、MEMWAIT レジスタか ROMWT レジスタによって設定されま

す。MEMWAIT レジスタか ROMWT レジスタの設定値は、システムクロックや動作電力制御モードに依存します。MEMWAIT レジスタか ROMWT レジスタの設定はユーザーズマニュアルの制限事項を確認してください。

2.7　STDIO とデバッグコンソール

STDIO が許可されている場合（3.2.3 参照）、STDIO ライブラリは、MCU 初期化処理の一部として初期化されます。STDIO の出力を、HEW または e² studio で表示できるデバッグコンソールに送信するように、r_bsp コードが設定されます。ソースファイル lowlvl.c によって、STDIO 関数のバイト単位での送受信が行われます。また、初期化処理では、デバッグコンソールが使用されるように設定されます。必要に応じて、r_bsp_config.h を変更し、BSP_CFG_USER_CHARGET_ENABLED や BSP_CFG_USER_CHARPUT_ENABLED を有効にし、さらに my_sw_charget_function や my_sw_charput_function 関数名を用意してそれぞれの関数の名前に置き換えることにより、STDIO の charget()や charput()関数をそれぞれの関数にリダイレクトすることができます。

2.8　スタック領域とヒープ領域

RX MCU では、ユーザスタックと割り込みスタックの 2 種類のスタックが使用できます。両スタックが使用される場合、通常の実行フロー実行時はユーザスタックが使用され、割り込み処理実行時には割り込みスタックが使用されます。これら 2 種類のスタックがあれば、割り込みが発生した場合に備えて余分なスペースをユーザスタックに確保する必要がなくなるので、スタックスペースの割り当てが簡単になります。しかし、アプリケーションによってはこのような考慮は必要がなく、RAM を浪費することにもなるので（スタック間の未使用スペース）、スタックを 2 種類持つことが好ましくない場合もあるでしょう。スタックを 1 種類のみ使用する場合は、必ず割り込みスタックを使用します。

ユーザスタック、割り込みスタック、ヒープはリセット後に r_bsp コード内で設定、および初期化されます。スタックやヒープのサイズ、また使用するスタック数などの設定は、r_bsp_config.h で行われます（3.2.2 参照）。また、ユーザはヒープを無効にすることもできます。

2.9　CPU モード

リセット解除後、RX MCU はスーパバイザ CPU モードで動作します。スーパバイザモードでは、CPU のリソースおよび命令がすべて使用できます。また、r_bsp のコードが main()にジャンプする前にユーザモードに遷移するオプションがあります（3.2.4 参照）。ユーザモードでは、以下に示す項目への書き込み命令に制限があります。

- プロセッサステータスワード（PSW）の次のビット：IPL[3:0]、PM、U、I
- 割り込みスタックポインタ（ISP）
- 割り込みテーブルレジスタ（INTB）
- バックアップ PSW（BPSW）
- バックアップ PC（BPC）
- 高速割り込みベクタレジスタ（FINTV）

ユーザモードで、MCU が上記のいずれかに対して書き込みを実行した場合、例外処理が発生します。コールバックが設定されている場合は、例外コールバック関数となります（2.4 参照）。

2.10　ID コード

RX MCU には ROM にある 16 バイトの 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、固定ベクタテーブルに配置され、r_bsp_config.h で簡単に設定できます（3.2.7 参照）。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「フラッシュメモリ」章や「ROM」章を参照ください。

2.11　パラレルライタのアクセス保護

RX MCU には、MCU のメモリをパラレルライタのアクセスから保護するために、ROM に 4 バイトのコードがあります。読み出し、および書き込みを許可、書き込みのみを許可、あるいはすべてのアクセスを禁止するなどのオプションがあります。本機能の詳細は、3.2.7 をご覧ください。

2.12　エンディアン

RX MCU では、ビッグエンディアン、リトルエンディアンのいずれで動作するかを選択できます。ご使用になる MCU によって、モードの選択方法が異なります。RX610 および RX62x MCU では端子レベルでエンディアンを決定します。RX100、RX200、RX63x、RX64x、RX65x、RX700 MCU では、ROM にあるレジスタ（MDE レジスタ）を使って、エンディアンを決定します。MDE レジスタを使用するデバイスでは、r_bsp は、ツールチェーンで選択されたエンディアンを検出し、それによってレジスタを正しく設定します。本バージョンの r_bsp では、以下のツールチェーンからエンディアンを検出します。

- Renesas RXC
- IAR
- KPIT GNU

2.13　オプション機能選択レジスタ

RX63x、RX200、RX100 MCU をご使用になる場合、ROM にオプション機能選択レジスタが配置されています。これらのレジスタを使って、ユーザコードを使用せずに、リセット時に特定の MCU 機能を許可に設定できます。これらの機能には、低電圧検出の許可、HOCO 発振の開始、IWDT の設定および開始などがあります。これらのレジスタに使用される値は r_bsp_config.h で設定できます（3.2.7 参照）。

2.14　Trusted Memory

RX64M、RX65N、RX71M には、Trusted Memory 機能に使用されるオプション設定メモリ（レジスタ）が 2 つあります。Trusted Memory 機能は、コードフラッシュメモリのブロック 8、ブロック 9 に対する不正リードを防止する機能で、初期設定では無効に設定されています。

Trusted Memory 機能は、r_bsp_config.h で定義される BSP_CFG_TRUSTED_MODE_FUNCTION を ‘0xF8FFFFFF’ に設定することによって、起動時に有効にできます。このオプションの値には‘0xF8FFFFFF’ または‘0xFFFFFFFF’のみを設定できます。

2.15　ボードごとの定義

各 board フォルダにはボードごとのヘッダファイルが含まれ、このファイルで、LED、スイッチ、SPI のスレーブ選択に使用される端子などが定義されています。ファイル名はボード名に ‘.h’ を付けたものになります。ボードが RSKRX111 の場合、ファイル名は rskrx111.h になります。

2.16　システム全般のパラメータチェック

デフォルトでは、FIT モジュールの入力パラメータチェックが有効になっています。この機能は開発時に役立ちますが、製品時には、実行時間の短縮のため、あるいはコードを配置する領域を確保するために、無効にできます。r_bsp_config.h には、システム全般に対してパラメータチェックの有効/無効を設定できるオプションがあります。各モジュールではデフォルトでこの設定を使用しますが、必要であれば、モジュールごとに無効にすることができます。本オプション設定の詳細は 3.2.9 をご覧ください。

2.17　ロック機能

r_bsp では、ロック機能を組み込む API 関数が用意されています。これらは、RTOS セマフォやミューテックスなどと同様に、重要なコードを保護するために使用されます。これらのロックには、RTOS が持つような拡張機能がないため、使用する際には注意が必要です。ロックを正しく使用しなければ、ユーザシステムがデッドロックされてしまう場合があります。

mcu フォルダにある mcu_locks.h には、mcu_lock_t という enum があり、MCU の周辺機能と周辺機能のチャネルごとにロックを持ちます。これらのロックを使って、予約された周辺機能を示すことができます。これは、FIT モジュールで周辺機能 3 チャネルを制御し、ユーザコードで 1 チャネルを制御したい場合などに使用できます。必要なチャネルにロックを予約することで、FIT モジュールの使用対象チャネルからそのチャネルを除外できます。また、これらのロックは、同一の周辺機能に対して複数の FIT モジュールを使用したい場合にも使用できます。例えば、調歩同期式の SCI を使用する FIT モジュールと I^2C モードの SCI を使用する FIT モジュールを使用する場合、ロック機能を使って、両 FIT モジュールによる同一の SCI チャネルの使用を防ぐことができます。ロック機能を組み込む API 関数は 4 つあり、5 章で説明されています。ハードウェアロック関数とソフトウェアロック関数の違いは、ハードウェアロック関数は mcu_locks.h で定義したロックのみを使用するという点です。ソフトウェアロック関数は、ロックを好きなところに配置でき、ユーザ定義のロックを作成できます。ロックが必要で、MCU の周辺機能を使用しない FIT モジュールの場合、それに対応したロックを作成し、ソフトウェアロックを使用します。

デフォルトの r_bsp ロック機能に替えてユーザ定義のロック機能を使用できます。詳細は 3.2.8 をご覧ください。

2.18　レジスタの保護

RX100、RX200、RX63x、RX64x、RX65x、RX700 グループの MCU にはプロテクトレジスタがあり、MCU のレジスタを不用意な書き込みから保護します。保護対象のレジスタには、クロック生成、消費電力低減機能、ソフトウェアリセット、低電圧検出関連のレジスタがあります。r_bsp には、これらのレジスタへの書き込みアクセスの許可/禁止が簡単に操作できる API 関数が用意されています。詳細は 5.7、5.8 をご覧ください。

2.19　CPU 機能

割り込みの許可/禁止や、CPU 割り込み優先レベルの設定など、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。

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 は様々なボード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。

プラットフォームを選択するには、使用するボードの ‘#include’ のコメントを解除します。RSK+RX63N ボードを使用する場合、 ‘./board/rskrx63n/r_bsp.h’の‘#include’ のコメントを解除します。その他のボードの ‘#include’ はコメント化しておいてください。

```
/***********************************************************************************
DEFINE YOUR SYSTEM - UNCOMMENT THE INCLUDE PATH FOR THE PLATFORM YOU ARE USING.
***********************************************************************************/
/* RSKRX610 */
//#include "./board/rskrx610/r_bsp.h"

/* RSKRX62N */
//#include "./board/rskrx62n/r_bsp.h"

/* RSKRX62T */
//#include "./board/rskrx62t/r_bsp.h"

/* RDKRX62N */
//#include "./board/rdkrx62n/r_bsp.h"

/* RSKRX630 */
//#include "./board/rskrx630/r_bsp.h"

/* RSKRX63N */
#include "./board/rskrx63n/r_bsp.h"
```

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 を作成するには、board フォルダから r_bsp_config_reference.h をコピーし、ファイル名を r_bsp_config.h に変更し、プロジェクトの適切な場所に置きます。r_bsp_config_reference.h は、ユーザが必要に応じて使用できる基本的なコンフィギュレーションファイルとして提供されます。r_bsp_config.h は r_config フォルダに格納することを推奨します。これは必須ではありませんが、全 FIT モジュールにそれぞれコンフィギュレーションファイルがありますので、1 カ所に集約することで、ファイルの検索やバックアップが容易になります。

r_bsp_config.h の内容はそれぞれのボード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。

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_PACKAGE	r_bsp_config.h にて、#define の上にあるコメントを参照。	使用するパッケージを定義します。パッケージサイズによって、MCU で使用できるピン数や周辺機能は異なります。
BSP_CFG_MCU_PART_MEMORY_SIZE		ROM、RAM、データフラッシュのサイズを定義します。
BSP_CFG_MCU_PART_GROUP		MCU グループを定義します（例：RX62N、RX63T）。
BSP_CFG_MCU_PART_SERIES		MCU シリーズを定義します（例：RX600、RX200、RX100）

3.2.2 スタックサイズとヒープサイズ

RX デバイスのスタックサイズは、RX ツールチェーンの#pragma ディレクティブを使って定義されます。

表 3.2　スタックサイズとヒープサイズの定義

定義	値	説明
BSP_CFG_USER_STACK_ENABLE	0＝割り込みスタックのみを使用 1＝割り込みスタックとユーザスタックを使用	スタックを 1 つ（割り込みスタック）使うか、2 つ（割り込みスタックとユーザスタック）使うかを選択します。詳細は 2.8 参照。
#pragma stacksize su=	ユーザスタックサイズ（単位：バイト）	ユーザスタックのサイズを定義します。エディタの設定によっては、本マクロは非表示になっていることがあります。
#pragma stacksize si=	割り込みスタックサイズ（単位：バイト）	割り込みスタックのサイズを定義します。エディタの設定によっては、本マクロは非表示になっていることがあります。
BSP_CFG_HEAP_BYTES	ヒープサイズ（単位：バイト）	ヒープサイズを定義します。ヒープを禁止にするには、この定義の ‘#define’ の上にあるコメントを参照ください。

3.2.3 STDIO 許可

STDIO ライブラリを使用するには、余分にコードスペース、RAM スペース、およびヒープの使用が必要になります。このため、STDIO を使用する必要がない場合、これを禁止にしてメモリを確保することを推奨します。

表 3.3　STDIO 許可の定義

定義	値	説明
BSP_CFG_IO_LIB_ENABLE	0＝STDIO の使用を禁止 1＝STDIO の使用を許可	起動時、STDIO の初期化関数を呼び出すかどうかを決定します。この初期化関数で、STDIO ライブラリが設定されます。

3.2.4　CPU モードとブートモード

RX MCU には、シリアルブートモード、ユーザブートモード、シングルチップモードなど、複数のブートモードがあります。RX610 および RX62x グループの MCU では、起動時に該当端子のレベルによって、使用するブートモードを選択します。RX63x および RX200 グループなどの MCU では、端子を設定するとともに、ROM に値（UB コード）を設定して選択します。

表 3.4　CPU モードとブートモードの定義

定義	値	説明
BSP_CFG_RUN_IN_USER_MODE	0＝スーパバイザモードに留まる 1＝ユーザモードに遷移	リセット後、RX MCU はスーパバイザモードで動作します。このオプションでユーザモードへの遷移を選択できます（ユーザモードでは、特定のレジスタへの書き込みアクセスが制限されます）。 できる限り、スーパバイザモードでの使用が推奨されます。
BSP_CFG_USER_BOOT_ENABLE	0＝ユーザブートモード禁止 1＝ユーザブートモード許可	RX63x および RX200 グループ MCU では、ユーザブートモードに遷移するために、ROM に値（UB コード）を設定する必要があります。本マクロでユーザブートモードを許可に定義した場合、r_bsp でそれに対応した UB コードが設定されます。

3.2.5　RTOS

表 3.5　RTOS の定義

定義	値	説明
BSP_CFG_RTOS_USED	0＝RTOS を使用しない 1＝RTOS 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。FIT モジュールによっては、この情報をその FIT モジュール自身の設定に使用することがあります。

3.2.6　クロックの設定

RX 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 3.6　クロック設定の定義 (1/2)

定義	値	説明
BSP_CFG_CLOCK_SOURCE	0＝低速オンチップオシレータ (LOCO) 1＝高速オンチップオシレータ (HOCO) 2＝メインクロック発振器 3＝サブクロック発振器 4＝PLL 回路	r_bsp コードが main()にジャンプする際に使用されるクロックソースを定義します。
BSP_CFG_MAIN_CLOCK_SOURCE	0=発振子 1=外部発振入力	メインクロックの発振器の発振源を定義します。
BSP_CFG_PLL_SOURCE	0＝メインクロック 1＝HOCO	PLL 回路に使用されるクロックソースを定義します。
BSP_CFG_USB_CLOCK_SOURCE	0＝システムクロック (ICLK) 1＝USB PLL 回路	USB 有効時に使用されるクロックソースを定義します。
BSP_CFG_LCD_CLOCK_SOURCE	0＝低速オンチップオシレータ (LOCO) 1＝高速オンチップオシレータ (HOCO) 2＝メインクロック発振器 3＝サブクロック発振器 4＝IWDT 専用クロック (IWDTCLK)	LCD 有効時に使用されるクロックソースを定義します。
BSP_CFG_LCD_CLOCK_ENABLE	0＝LCD クロックソース無効 1＝LCD クロックソース有効	LCD のクロックソースの有効／無効を定義します。
BSP_CFG_HOCO_FREQUENCY	0＝16MHz 1＝18MHz 2＝20MHz	HOCO の周波数を定義します。
BSP_CFG_LPT_CLOCK_SOURCE	0＝サブクロック 1＝IWDT 専用オンチップオシレータ	ローパワータイマ有効時に使用するクロックソースを定義します。
BSP_CFG_XTAL_HZ	入力クロック周波数 (単位: Hz)	入力クロック(発振子または外部クロック)の周波数を定義します。クロックの周波数の計算に使用されます。
BSP_CFG_PLL_DIV	PLL 入力周波数分周比	PLL 入力分周比を定義します。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_MUL	PLL 周波数逓倍率	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_UPLL_DIV	USB PLL 入力周波数分周比	USB PLL 入力分周比を定義します。USB 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_UPLL_MUL	USB PLL 周波数逓倍率	USB PLL 逓倍率を定義します。USB 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。

表 3.6　クロック設定の定義 (2/2)

定義	値	説明
BSP_CFG_<ClockAcronym>_DIV 例: BSP_CFG_ICK_DIV BSP_CFG_PCKA_DIV BSP_CFG_PCKB_DIV BSP_CFG_PCKC_DIV BSP_CFG_PCKD_DIV BSP_CFG_FCK_DIV	設定するクロックの分周比として使用する除数	RX MCU には様々なクロックドメインが内蔵されています。各クロックの消費電力を抑える一方で、パフォーマンスを最大限に引き出すために、個別に分周比を設定できます。 <ClockAcronym>には、設定対象のクロック名が入ります。 例えば、CPU クロック (ICLK)の分周比を設定するには、BSP_CFG_ICK_DIV マクロを設定します。
BSP_CFG_HOCO_WAIT_TIME	HOSCWTCR レジスタの設定値	高速オンチップオシレータウェイト時間を定義します。
BSP_CFG_MOSC_WAIT_TIME	MOSCWTCR レジスタの設定値	メインクロック発振器ウェイト時間を定義します。
BSP_CFG_SOSC_WAIT_TIME	SOSCWTCR レジスタの設定値	サブクロック発振器ウェイト時間を定義します。
BSP_CFG_ROM_CACHE_ENABLE	0=ROM キャッシュ動作禁止 1=ROM キャッシュ動作許可	ROM キャッシュ動作を定義します。
BSP_CFG_BCLK_OUTPUT	0＝BCLK を出力しない 1＝BCK 周波数を出力する 2＝BCK/2 周波数を出力する	BCLK を出力するかどうかを定義します。出力する場合、出力する周波数を定義します。
BSP_CFG_SDCLK_OUTPUT	0＝SDCLK を出力しない 1＝BCK 周波数を出力する	SDCLK を出力するかどうかを定義します。
BSP_CFG_USE_CGC_ MODULE	0＝r_bsp モジュール内部のクロック設定コードを使用する 1＝r_cgc_rx モジュールを使用してクロックを管理する	RX グループには、r_cgc_rx モジュールを使用できるものがあります。r_cgc_rx モジュールを使用する場合、r_bsp モジュール内部のクロック設定コードは削除され、r_cgc_rx モジュールの呼び出しに置き換えられます。r_cgc_rx モジュールは、実行時にクロックの設定を変更できるなど、内部クロックコードよりも高性能で、多くの機能を有します。アプリケーションで、動的にクロックを変更する必要がない場合は、このオプションに 0 を設定すると、コードサイズを小さくできます。

3.2.7　ROM 上のレジスタ、および外部メモリアクセスの保護

メモリ保護関連のレジスタやオプション機能選択レジスタなど、レジスタの中には ROM に配置されているものがあり、それらはコンパイル時に設定する必要があります。

RX MCU には製品化後に、MCU メモリが不用意に読み出されないように保護する 2 種類の方法があります。1 つめは ID コードを使用する方法です。RX ID コードは 16 バイトの数値で、デバッガから、またはシリアルブートモードから接続されないように MCU を保護します。ID コードの設定については、MCU ごとに異なる場合がありますので、ご使用の MCU のユーザーズマニュアル ハードウェア編をご覧ください。2 つめの方法は、ROM コードプロテクトという 4 バイトの数値を使用します。この数値によって、パラレルライタで行える MCU への書き込み/読み込み動作を決定します。

オプション機能選択レジスタ (OFS0、OFS1)を設定して、リセット時の動作を選択できます。例えば、IWDT の設定および許可、電圧検出の許可、HOCO の発振許可などが行えます。オプション機能選択レジスタが設定されている場合、MCU のリセットベクタが取得され、実行される前に、レジスタに設定された動作を完了します。

表 3.7　ROM 上のレジスタ定義

定義	値	説明
BSP_CFG_ID_CODE_LONG_1 BSP_CFG_ID_CODE_LONG_2 BSP_CFG_ID_CODE_LONG_3 BSP_CFG_ID_CODE_LONG_4	ID コードの設定 (4 バイト単位)	MCU の ID コードを定義します。初期値は全て 0xFF で、保護は解除されています。 **注**：ID コードを設定した場合はコードを記録しておいてください。デバッガやシリアルブートモードで接続が必要なときにコードを使用します。
BSP_CFG_ROM_CODE_ PROTECT_VALUE	0＝読み出し/書き換えアクセスを禁止 1＝読み出しアクセスを禁止 Else＝読み出し/書き換えアクセスを許可	パラレルライタに許可される読み出し/書き換えアクセスを定義します。
BSP_CFG_OFS0_REG_VALUE	OFS0 レジスタに書き込む値	ROM の OFS0 番地に書き込む 4 バイトの値を定義します。
BSP_CFG_OFS1_REG_VALUE	OFS1 レジスタに書き込む値	ROM の OFS1 番地に書き込む 4 バイトの値を定義します。
BSP_CFG_TRUSTED_MODE_FUNCTION	TMEF レジスタに書き込む値	RX64M、RX65N および RX71M グループ MCU にて、Trusted Memory の有効／無効を定義します。
BSP_CFG_FAW_REG_VALUE	FAW レジスタに書き込む値	RX65N グループ MCU にて、ROM の FAW 番地に書き込む 4 バイトの値を定義します。
BSP_CFG_ROMCODE_REG_VALUE	ROMCODE レジスタに書き込む値	RX65N グループ MCU にて、ROM の ROMCODE 番地に書き込む 4 バイトの値を定義します。

3.2.8　ロック機能

r_bsp のロック機能については、2.17 で紹介しています。ここで紹介するマクロを使って、デフォルトのロック機能を無効にして、ユーザ定義のロック機能を使用することができます。ユーザは、シンプルな r_bsp デフォルトのロック機能に替えて、RTOS のセマフォやミューテックスなど、より機能強化したオブジェクトを使用することができます。そのためにはまず、ユーザ定義のロック機能を使用するように r_bsp を設定します（以下の BSP_CFG_USER_LOCKING_ENABLED 参照）。次に、BSP_CFG_USER_LOCKING_TYPE を、ユーザ定義のロック機能に対応した型に定義します。RTOS のセマフォを使用する場合、ここでセマフォに対応した型を定義します。その後、4 種類のロック関数を定義する必要があります（表 3.8 の下から 4 つの関数）。これらのユーザ定義関数への引数は、デフォルトのロック関数に渡される引数と一致する必要があります。関数定義が変更されると、ユーザプロジェクトのすべてのロック機能が、ユーザ定義のロック機能に変換されます。ユーザコード、または FIT モジュールによって r_bsp ロック関数が呼び出される場合、ユーザ定義のロック関数が呼び出されます。ユーザ定義のロック機能を使用する場合は、ユーザご自身の責任において行ってください。ユーザ定義のロック関数では、RTOS のより強化された拡張機能を使用することもできます。

表 3.8　ロック機能の定義

定義	値	説明
BSP_CFG_USER_LOCKING_ENABLED	0＝デフォルトのロック機能を使用 1＝ユーザ定義のロック機能を使用	r_bsp で提供されるデフォルトのロック機能は RTOS を使用しないため、RTOS のセマフォやミューテックスなどの拡張機能は提供されません。
BSP_CFG_USER_LOCKING_TYPE	ロックに使用されるデータ型 (デフォルトは bsp_lock_t)	ユーザ定義のロック機能を使用する場合、ここでデータ型を定義します。デフォルトのロックに替えて RTOS のセマフォやミューテックスを使用する場合、そのデータ型を指定します。
BSP_CFG_USER_LOCKING_HW_LOCK_FUNCTION	ユーザ定義のロック機能を使用する場合、呼び出したいユーザ定義のハードウェアロック関数を設定してください。	ユーザ定義のロック機能を使用する場合、R_BSP_HardwareLock()が呼び出されると、本マクロで定義される関数が呼び出されます。
BSP_CFG_USER_LOCKING_HW_UNLOCK_FUNCTION	ユーザ定義のロック機能を使用する場合、呼び出したいユーザ定義のハードウェアアンロック関数を設定してください。	ユーザ定義のロック機能を使用する場合、R_BSP_HardwareUnlock()が呼び出されると、本マクロで定義される関数が呼び出されます。
BSP_CFG_USER_LOCKING_SW_LOCK_FUNCTION	ユーザ定義のロック機能を使用する場合、呼び出したいユーザ定義のソフトウェアロック関数を設定してください。	ユーザ定義のロック機能を使用する場合、R_BSP_SoftwareLock()が呼び出されると、本マクロで定義される関数が呼び出されます。
BSP_CFG_USER_LOCKING_SW_UNLOCK_FUNCTION	ユーザ定義のロック機能を使用する場合、呼び出したいユーザ定義のソフトウェアアンロック関数を設定してください。	ユーザ定義のロック機能を使用する場合、R_BSP_SoftwareUnlock()が呼び出されると、本マクロで定義される関数が呼び出されます。

3.2.9　パラメータチェック

本マクロはシステム全般のパラメータチェックの有効/無効を設定します。また、各 FIT モジュールには、同様の機能を果たすローカルマクロがあります。デフォルトで、ローカルマクロはグローバル設定された数値を取得しますが、グローバル設定された値は無効にできます。ローカル設定は、本マクロのグルーバル設定より優先されます。パラメータチェックは、入力値が正しいと判断できるとき、処理速度を上げたいとき、コードの容量を小さくしたいときにのみ、無効にするようにしてください。

表 3.9　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	システム全般のパラメータチェックの有効/無効を定義します。ローカルのモジュールはデフォルトでこの定義の設定値を取りますが、その値は無効にできます。

3.2.10　MCU 電圧

定義	値	説明
BSP_CFG_MCU_VCC_MV	MCU に提供される電圧 (Vcc) (単位：mV)	FIT モジュールには、LVD など、MCU に提供される電圧情報が必要なものがあります。この関数を定義して電圧情報を取得できます。

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

なし

4.5 ツールチェーン

本ドライバは以下のツールチェーンで動作を確認しています。

- Renesas RX Toolchain v.2.01.00 (RX110, RX111, RX113, RX210, RX21A, RX220, RX231, RX610, RX62N, RX62T, RX62G, RX630, RX631, RX63N, RX63T, RX64M, RX71M)
- Renesas RX Toolchain v.2.03.00 (RX130, RX230, RX23T, RX24T)
- Renesas RX Toolchain v.2.05.00 (RX65N)

4.6 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。

4.7 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。

4.8 コンフィギュレーションの概要

コンフィギュレーションに関する情報は、3 章をご覧ください。

4.9 API データ構造体

4.9.1 ソフトウェアロック

このデータ構造体は、RX MCU にロック機能を組み込むために使用されます。lock メンバは、RX の XCHG 命令を使用するために、4 バイトである必要があります。この構造体は、BSP_CFG_USER_LOCKING_TYPE マクロにて、デフォルトで定義されている型です。

```
typedef struct
{
    /* The actual lock. int32_t is used because this is what the xchg()
       instruction takes as parameters. */
    int32_t     lock;
} bsp_lock_t;
```

4.9.2　割り込みコールバックのパラメータ

このデータ構造体は、割り込みコールバック関数を呼び出すときに使用されます。割り込み処理で、この構造体に‘ (void *)’として埋め込まれ、コールバック関数に引数として渡されます。

```
typedef struct
{
    bsp_int_src_t vector;        //Which vector caused this interrupt
} bsp_int_cb_args_t;
```

4.10　API Typedef

4.10.1　レジスタの保護

この typedef はレジスタのタイプごとに保護オプションを定義し、オプションの設定対象は切り替えが可能です。それぞれのオプションでは、レジスタをグループで扱います。例えば、LPC、CGC、およびソフトウェアリセット関連のレジスタは同じビットによって保護されます。レジスタの分類や数は、使用する MCU によって異なります。MCU で利用できるオプションについては、ご使用の MCU の cpu.h をご確認ください。以下に RX111 を使用した場合の typedef を示します。

```
/* The different types of registers that can be protected. */
typedef enum
{
    /* Enables writing to the registers related to the clock generation circuit:
       SCKCR, SCKCR3, PLLCR, PLLCR2, MOSCCR, SOSCCR,LOCOCR, ILOCOCR, HOCOCR,
       OSTDCR, OSTDSR, CKOCR. */
    BSP_REG_PROTECT_CGC = 0,
    /* Enables writing to the registers related to operating modes, low power
       consumption, the clock generation circuit, and software reset: SYSCR1,
       SBYCR, MSTPCRA, MSTPCRB, MSTPCRC, OPCCR, RSTCKCR, SOPCCR, MOFCR, MOSCWTCR,
       SWRR. */
    BSP_REG_PROTECT_LPC_CGC_SWR,
    /* Enables writing to the HOCOWTCR register. */
    BSP_REG_PROTECT_HOCOWTCR,
    /* Enables writing to the registers related to the LVD: LVCMPCR, LVDLVLR,
       LVD1CR0, LVD1CR1, LVD1SR, LVD2CR0, LVD2CR1, LVD2SR. */
    BSP_REG_PROTECT_LVD,
    /* Enables writing to MPC's PFS registers. */
    BSP_REG_PROTECT_MPC,
    /* This entry is used for getting the number of enum items. This must be the
       last entry. DO NOT REMOVE THIS ENTRY!*/
    BSP_REG_PROTECT_TOTAL_ITEMS
} bsp_reg_protect_t;
```

4.10.2　ハードウェアリソースロック

この typedef はハードウェアリソースロックを定義します。ハードウェアロック配列には、enum のエントリごとにソフトウェアロックが 1 つ割り当てられます。ロックの項目や数は選択された MCU により異なります。以下に RX111 を使用した場合の typedef を示します。

```
typedef enum
{
    BSP_LOCK_BSC = 0,
    BSP_LOCK_CAC,
    BSP_LOCK_CMT,
    BSP_LOCK_CMT0,
    BSP_LOCK_CMT1,
    BSP_LOCK_CRC,
    BSP_LOCK_DA,
    BSP_LOCK_DOC,
    BSP_LOCK_DTC,
    BSP_LOCK_ELC,
    BSP_LOCK_FLASH,
    BSP_LOCK_ICU,
    BSP_LOCK_IRQ0,
    BSP_LOCK_IRQ1,
    BSP_LOCK_IRQ2,
    BSP_LOCK_IRQ3,
    BSP_LOCK_IRQ4,
    BSP_LOCK_IRQ5,
    BSP_LOCK_IRQ6,
    BSP_LOCK_IRQ7,
    BSP_LOCK_IWDT,
    BSP_LOCK_MPC,
    BSP_LOCK_MTU,
    BSP_LOCK_MTU0,
    BSP_LOCK_MTU1,
    BSP_LOCK_MTU2,
    BSP_LOCK_MTU3,
    BSP_LOCK_MTU4,
    BSP_LOCK_MTU5,
    BSP_LOCK_POE,
    BSP_LOCK_RIIC0,
    BSP_LOCK_RSPI0,
    BSP_LOCK_RTC,
    BSP_LOCK_RTCB,
    BSP_LOCK_S12AD,
    BSP_LOCK_SCI1,
    BSP_LOCK_SCI5,
    BSP_LOCK_SCI12,
    BSP_LOCK_SMCI1,
    BSP_LOCK_SMCI5,
    BSP_LOCK_SMCI12,
    BSP_LOCK_SYSTEM,
    BSP_LOCK_USB0,
    BSP_NUM_LOCKS   /* This entry is not a valid lock. It is used for sizing
                       g_bsp_Locks[] array below. Do not touch! */
} mcu_lock_t;
```

4.10.3　割り込みエラーコード

この typedef は、R_BSP_InterruptWrite()、 R_BSP_InterruptRead()、R_BSP_InterruptControl()関数から返すエラーコードを定義します。

以下に RX111 を使用した場合の typedef を示します。

他の RX MCU では、その他の割り込み制御コマンドがサポートされている場合があります。

```
typedef enum
{
   BSP_INT_SUCCESS = 0,
   BSP_INT_ERR_NO_REGISTERED_CALLBACK, //There is not a registered callback
                                       //for this interrupt source
   BSP_INT_ERR_INVALID_ARG,            //Illegal argument input
   BSP_INT_ERR_UNSUPPORTED             //Operation is not supported by this API
} bsp_int_err_t;
```

4.10.4　割り込み制御コマンド

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

以下に RX111 を使用した場合の typedef を示します。

他の RX MCU では、その他の割り込み制御コマンドがサポートされている場合があります。

```
typedef enum
{
   BSP_INT_CMD_CALL_CALLBACK = 0, //Calls registered callback function
                                  //if one exists
   BSP_INT_CMD_INTERRUPT_ENABLE,  //Enables a give interrupt (Available for NMI
                                  //pin, FPU, and Bus Error)
   BSP_INT_CMD_INTERRUPT_DISABLE  //Disables a given interrupt (Available for
                                  //FPU, and Bus Error)
} bsp_int_cmd_t;
```

4.10.5　割り込みコールバック関数

この typedef は、コールバック関数の型を定義します。コールバック関数は戻り値の型が‘void’、引数の型が‘void *’でなければなりません。

```
typedef void (*bsp_int_cb_t)(void *);
```

4.10.6　割り込み要因

このtypedefは、割り込みベクタを定義します。各割り込みベクタには、コールバックが登録されます。typedefで選択可能なオプションは、使用するMCUにより異なります。以下にRX111を使用した場合のtypedefを示します。他のRX MCUでは、その他の割り込みソースがサポートされている場合があります。

```
typedef enum
{
    BSP_INT_SRC_EXC_SUPERVISOR_INSTR = 0, //Occurs when privileged instruction
                                          //is executed in User Mode
    BSP_INT_SRC_EXC_UNDEFINED_INSTR,      //Occurs when MCU encounters an
                                          //unknown instruction
    BSP_INT_SRC_EXC_NMI_PIN,              //NMI Pin interrupt
    BSP_INT_SRC_EXC_FPU,                  //FPU exception
    BSP_INT_SRC_OSC_STOP_DETECT,          //Oscillation stop is detected
    BSP_INT_SRC_WDT_ERROR,                //WDT underflow/refresh error has
                                          //occurred
    BSP_INT_SRC_IWDT_ERROR,               //IWDT underflow/refresh error has
                                          //occurred
    BSP_INT_SRC_LVD1,                     //Voltage monitoring 1 interrupt
    BSP_INT_SRC_LVD2,                     //Voltage monitoring 2 interrupt
    BSP_INT_SRC_UNDEFINED_INTERRUPT,      //Interrupt has triggered for a vector
                                          //that user did not write a handler
                                          //for
    BSP_INT_SRC_BUS_ERROR,                //Bus error: illegal address access or
                                          //timeout
    BSP_INT_SRC_TOTAL_ITEMS               //DO NOT MODIFY! This is used for
                                          //sizing the interrupt callback array.
} bsp_int_src_t;
```

4.10.7　グループ割り込み

RX64M、RX65NおよびRX71M MCUはグループ割り込みをサポートしており、複数の（最大32の）周辺割り込み要求が1つの割り込み要求としてグループ化されます。割り込みは、周辺動作クロック（PCLKAまたはPCLKB）および割り込み要求の検出方法（エッジ検出またはレベル検出）に応じてグループ化されます。グループ割り込み要求が生成されるとき、それぞれの（AまたはB、エッジまたはレベルの）グループ割り込み要求レジスタを調べることで割り込みのソースが識別されます。

4.10.8　選択型割り込み

RX64M、RX65NおよびRX71M MCUでは、周辺割り込みソースを128～255のベクタ番号に動的に割り当てることができます。これらは、周辺動作クロックに基づいて選択型割り込みAと選択型割り込みBに分類されます。選択型割り込みBは、PCLKBと同期して動作する周辺機能で使用でき、割り込み番号128～207に割り当てることができます。選択型割り込みAは、PCLKAと同期して動作する周辺機能で使用でき、割り込み番号208～255に割り当てることができます。

4.11　戻り値

なし

4.12　ドライバをプロジェクトに追加する

詳細は、6章をご覧ください。

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_GetVersion	r_bsp のバージョンを返す。
R_BSP_InterruptsDisable	割り込みを全般的に禁止する。
R_BSP_InterruptsEnable	割り込みを全般的に許可する。
R_BSP_CpuInterruptLevelRead	CPU の割り込み優先レベルを読み出す。
R_BSP_CpuInterruptLevelWrite	CPU の割り込み優先レベルを書き込む。
R_BSP_RegisterProtectEnable	選択したレジスタの書き込み保護を有効にする。
R_BSP_RegisterProtectDisable	選択したレジスタの書き込み保護を無効にする。
R_BSP_SoftwareLock	ロックを予約する。
R_BSP_SoftwareUnlock	ロックを解除する。
R_BSP_HardwareLock	ハードウェアロックを予約する。
R_BSP_HardwareUnlock	ハードウェアロックを解除する。
R_BSP_InterruptWrite	割り込みに使用するコールバック関数を登録する。
R_BSP_InterruptRead	コールバックの登録がある場合、対象の割り込みに使用するコールバックを取得する。
R_BSP_InterruptControl	その他、様々な割り込み機能を実行する。
R_BSP_SoftwareDelay	指定した時間だけ遅延させる。

5.2　R_BSP_GetVersion

r_bsp のバージョンを返します。

フォーマット

```
uint32_t R_BSP_GetVersion(void);
```

パラメータ

なし

戻り値

r_bsp のバージョン

プロパティ

r_bsp_common.h でプロトタイプ宣言

r_bsp_common.c に組み込まれる

説明

本関数は、現在インストールされている r_bsp のバージョンを返します。バージョン番号はコード化されています。最初の 2 バイトがメジャーバージョン番号で、後の 2 バイトがマイナーバージョン番号です。例えば、バージョンが 4.25 の場合、戻り値は ‘0x00040019’ となります。

リエントラント

可

例

```
uint32_t cur_version;

/* Get version of installed r_bsp. */
cur_version = R_BSP_GetVersion();

/* Check to make sure version is new enough for this application's use. */
if (MIN_VERSION > cur_version)
{
   /* This r_bsp version is not new enough and does not have XXX feature
      that is needed by this application. Alert user. */
   ....
}
```

参考情報

本関数は、r_bsp_common.c に、インライン関数として設定されます。

5.3　R_BSP_ InterruptsDisable

割り込みを全般的に禁止します。

フォーマット

```
   void R_BSP_InterruptsDisable(void);
```

パラメータ

なし

戻り値

なし

プロパティ

cpu.h でプロトタイプ宣言

cpu.c に組み込まれる

説明

本関数は、割り込みを全般的に禁止します。この関数では、CPU のプロセッサステータスワード（PSW）レジスタの I ビットをクリアします。

リエントラント

可

例

```
/* Disable interrupts so that accessing this critical area will be guaranteed
   to be atomic. */
R_BSP_InterruptsDisable();

/* Access critical resource while interrupts are disabled */
....

/* End of critical area. Enable interrupts. */
R_BSP_InterruptsEnable();
```

参考情報

PSW の I ビットは、スーパバイザモードでのみ変更できます。CPU がユーザモードのときに、この関数が呼び出されると、“特権命令例外”が発生します。

5.4　R_BSP_InterruptsEnable

割り込みを全般的に許可します。

フォーマット

```
void R_BSP_InterruptsEnable(void);
```

パラメータ

なし

戻り値

なし

プロパティ

cpu.h でプロトタイプ宣言

cpu.c に組み込まれる

説明

本関数は、割り込みを全般的に許可します。この関数は、CPU のプロセッサステータスワード（PSW）レジスタの I ビットを設定します。

リエントラント

可

例

```
/* Disable interrupts so that accessing this critical area will be guaranteed
   to be atomic. */
R_BSP_InterruptsDisable();

/* Access critical resource while interrupts are disabled */
....

/* End of critical area. Enable interrupts. */
R_BSP_InterruptsEnable();
```

参考情報

PSW の I ビットは、スーパバイザモードでのみ変更できます。CPU がユーザモードのときに、この関数が呼び出されると、“特権命令例外”が発生します。

5.5　R_BSP_CpuInterruptLevelRead

CPU の割り込み優先レベルを読み出します。

フォーマット

```
uint32_t R_BSP_CpuInterruptLevelRead(void);
```

パラメータ

なし

戻り値

CPU の割り込み優先レベル

プロパティ

cpu.h でプロトタイプ宣言

cpu.c に組み込まれる

説明

本関数は、CPU の割り込み優先レベルを読み出します。割り込み優先レベルは、CPU のプロセッサステータスワード（PSW）レジスタの IPL ビットに格納されています。

リエントラント

可

例

```
uint32_t cpu_ipl;

/* Read the CPU's Interrupt Priority Level. */
cpu_ipl = R_BSP_CpuInterruptLevelRead();
```

参考情報

なし

5.6　R_BSP_CpuInterruptLevelWrite

CPU の割り込み優先レベルを書き込みます。

フォーマット

```
bool R_BSP_CpuInterruptLevelWrite(uint32_t level);
```

パラメータ

level: CPU の IPL の書き込みレベル

戻り値

true: CPU の IPL の書き込み成功。

false: level で渡された IPL は無効な値。

プロパティ

cpu.h でプロトタイプ宣言

cpu.c に組み込まれる

説明

本関数は、CPU の割り込み優先レベルを書き込みます。割り込み優先レベルは、CPU のプロセッサステータスワード（PSW）レジスタの IPL ビットに格納されています。また、本関数は、IPL に書き込まれた値が有効であるか確認します。IPL ビットの設定として有効な最大値および最小値は、BSP_MCU_IPL_MAX マクロ、BSP_MCU_IPL_MIN マクロを使って、mcu_info.h に定義されます。

リエントラント

可

例

```
/* Response time is critical during this portion of the application. Set the
   CPU's Interrupt Priority Level so that interrupts below the set
   threshold are disabled. Interrupt vectors with IPLs higher than this
   threshold will still be accepted and will not have to contend with the
   lower priority interrupts. */
if (false == R_BSP_CpuInterruptLevelWrite(HIGH_PRIORITY_THRESHOLD))
{
    /* Error in setting CPU's IPL. Invalid IPL was provided. */
    ....
}

/* Only high priority interrupts (as defined by user) will be accepted during
   this period. */
....

/* Time sensitive period is over. Set CPU's IPL back to lower value so that
   lower priority interrupts can now be serviced again. */
if (false == R_BSP_CpuInterruptLevelWrite(LOW_PRIORITY_THRESHOLD))
{
    /* Error in setting CPU's IPL. Invalid IPL was provided. */
    ....
}
```

参考情報

CPU の IPL ビットは、スーパバイザモードでのみ変更できます。CPU がユーザモードのときに、この関数が呼び出されると、“特権命令例外”が発生します。

5.7　R_BSP_RegisterProtectEnable

選択されたレジスタの書き込み保護を有効にします。

フォーマット

```
void R_BSP_RegisterProtectEnable(bsp_reg_protect_t regs_to_protect);
```

パラメータ

regs_to_protect: 書き込み保護を有効にするレジスタを指定

戻り値

なし

プロパティ

cpu.h でプロトタイプ宣言

cpu.c に組み込まれる

説明

本関数は、指定した入力レジスタの書き込み保護を有効にします。限られた MCU レジスタに対してのみ、書き込み保護を設定できます。本関数を適用できるレジスタについては、ご使用の MCU の cpu.h で、bsp_reg_protect_t enum をご確認ください。

本関数、および R_BSP_RegisterProtectDisable()は、エントリごとに、bsp_reg_protect_t enum のカウンタを使用します。これによって、これらの関数を複数回呼び出すことが可能になります。カウンタの使用については、本セクションの参考情報で説明します。

リエントラント

不可

例

```
/* Write access must be enabled before writing to MPC registers. */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

/* MPC registers are now writable. */
/* Setup Port 2 Pin 6 as TXD1 for SCI1. */
MPC.P26PFS.BYTE = 0x0A;

/* Setup Port 4 Pin 2 as AD input for potentiometer. */
MPC.P42PFS.BYTE = 0x80;

/* More pin setup. */
....

/* Enable write protection for MPC registers to protect against accidental
   writes. */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);
```

参考情報

レジスタ保護にカウンタを使用する理由を以下の例で説明します。

1. ユーザアプリケーションで、r_module1 の open 関数を呼び出します。
2. 本モジュールの初期化処理中に書き込みが必要なレジスタについて、r_module1 は R_BSP_RegisterProtectDisable()を使って、書き込み保護を無効にします。この時点で、対象のレジスタのカウンタがインクリメントされて 1 になります。
3. r_module1 は、上記の手順で書き込みが許可されたレジスタに書き込みます。
4. r_module1 は r_module2 を使う必要がある場合、r_module2 の open 関数である R_MODULE2_Open()を呼び出します。
5. r_module2 の関数では、r_module1 で書き込み許可したのと同じレジスタに書き込みが必要です。r_module2 は、r_module 1 でそれらのレジスタの書き込み保護を無効にしていることが認識されていないので、R_BSP_RegisterProtectDisable()を呼び出します。ここで、対象のレジスタのカウンタがインクリメントされ、カウンタ値が 2 になります。
6. r_module2 は、上記の手順で書き込みが許可されたレジスタに書き込みます。
7. r_module2 は書き込みが完了すると、書き込みを許可したレジスタの書き込み保護を有効にするために R_BSP_RegisterProtectEnable()を呼び出します。ここで書き込み保護が有効になったレジスタのカウンタがデクリメントされ 1 になります。コードは、カウンタが 0 ではないことから、実際に書き込み保護を有効にしてはならないと判断します。
8. プログラムはレジスタの書き込みを続行している R_MODULE1_Open()に戻ります。このとき、カウンタが使用されていなければ、r_module2 が R_BSP_RegisterProtectEnable()を呼び出したことによって（手順 7）、r_module1 のレジスタへの書き込み動作ができなくなり、問題となります。
9. r_module1 では、書き込み許可したレジスタへの書き込みが完了すると、R_BSP_RegisterProtectEnable()を呼び出し、それらのレジスタの書き込み保護を有効にします。カウンタはデクリメントされ、カウンタ値は 0 になります。カウンタ値が 0 になることで、API コードはレジスタの書き込み保護を有効にしてもよいと判断します。

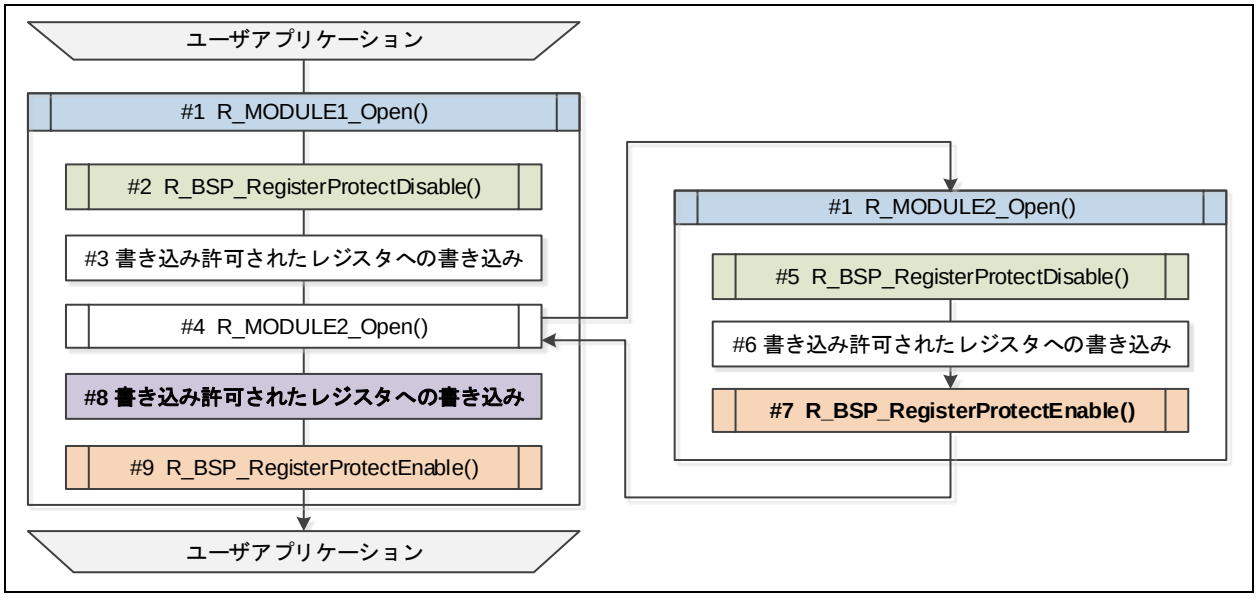


図 5.1　レジスタの書き込み保護設定例

5.8　R_BSP_RegisterProtectDisable

選択されたレジスタの書き込み保護を無効にします。

フォーマット

```
void R_BSP_RegisterProtectDisable(bsp_reg_protect_t regs_to_unprotect);
```

パラメータ

regs_to_unprotect: 書き込み保護を無効にするレジスタを指定

戻り値

なし

プロパティ

cpu.h でプロトタイプ宣言

cpu.c に組み込まれる

説明

本関数は、入力レジスタの書き込み保護を無効にします。限られた MCU レジスタのみが、書き込み保護を設定できます。本関数を適用できるレジスタについては、ご使用の MCU の cpu.h で、bsp_reg_protect_t enum をご確認ください。

本関数、および R_BSP_RegisterProtectEnable()は、エントリごとに、bsp_reg_protect_t enum のカウンタを使用します。これによって、これらの関数を複数回呼び出すことが可能になります。カウンタの使用については、5.7 の参考情報をご覧ください。

リエントラント

不可

例

```
/* Write access must be enabled before writing to CGC registers. */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_CGC);
/* CGC registers are spread amongst two protection bits. */
R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_LPC_CGC_SWR);

/* CGC registers are now writable. */
/* Select PLL as clock source. */
SYSTEM.SCKCR3.WORD = 0x0400;

/* More clock setup. */
....

/* Enable write protection for CGC registers to protect against accidental
   writes. */
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_CGC);
R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_LPC_CGC_SWR);
```

参考情報

なし

5.9　R_BSP_SoftwareLock

ロックを予約します。

フォーマット

```
bool R_BSP_SoftwareLock(BSP_CFG_USER_LOCKING_TYPE * const plock);
```

パラメータ

plock: ロックを予約、設定するためのロック構造体へのポインタ

戻り値

true: ロックの使用が可能で設定に成功。

false: ロックは既に設定されていて、使用不可。

プロパティ

locking.h でプロトタイプ宣言

locking.c に組み込まれる

説明

本関数は、ロック機能を組み込みます。ロックは様々な方法で使用できます。一般的には、重要なコードの保護、リソースの割り当ての重複を防ぐことを目的によく使用されます。重要なコードを保護するには、プログラムの実行前に重要箇所をロックする必要があります。リソースの割り当ての重複回避は、2 つの FIT モジュールで、同じ周辺機能を使用する場合などに必要になります。例えば、一方の FIT モジュールが UART モードで SCI を使用し、他方が I²C モードで SCI を使用する場合がこれに当たります。両方の FIT モジュールが同じ SCI チャネルを使用できないようにするために、ロックを使用できます。

ロックを使用する場合、RTOS のセマフォやミューテックスなどにある拡張機能はご使用になれません。ロックが正しく行われなければ、システムのデッドロックに繋がりますのでご注意ください。

デフォルトのロック機能は無効にできます。詳細は 3.2.8 をご覧ください。

リエントラント

可

例

ここでは、Virtual EEPROM コードの例を使って、ロックの使用例を示します。この FIT モジュールは周辺機能に直接アクセスはしませんが、リエントラントを防ぐためにロックされる必要があります。

```
/* Used for locking state of VEE */
static BSP_CFG_USER_LOCKING_TYPE g_vee_lock;

/***********************************************************************
* Function Name: vee_lock_state
* Description  : Tries to lock the VEE state
* Arguments    : state -
*                  Which state to try to transfer to
* Return value : VEE_SUCCESS -
*                  Successful, state taken
*              VEE_BUSY -
*                  Data flash is busy, state not taken
***********************************************************************/
static uint8_t vee_lock_state (vee_states_t state)
{
    /* Local return variable */
    uint8_t ret = VEE_SUCCESS;
```

```
    /* Try to lock VEE to change state. */
    /* Check to see if lock was successfully taken. */
    if(false == R_BSP_SoftwareLock(&g_vee_lock))
    {
        /* Another operation is on-going */
        return VEE_BUSY;
    }


    /* Check VEE status to make sure we are not interfering with another
       thread */
    if( state == VEE_READING )
    {
        /* If another read comes in while the state is reading then we are OK */
        if( ( g_vee_state != VEE_READY ) && ( g_vee_state != VEE_READING) )
        {
            /* VEE is busy */
            ret = VEE_BUSY;
        }
    }
    else
    {
        /* If we are doing something other than reading then we must be in the
           VEE_READY state */
        if( g_vee_state != VEE_READY )
        {
            /* VEE is busy */
            ret = VEE_BUSY;
        }
    }

    if( ret == VEE_SUCCESS )
    {
        /* Lock state */
        g_vee_state = state;
    }

    /* Release lock. */
    R_BSP_SoftwareUnlock(&g_vee_lock);

    return ret;
}
```

参考情報

なし

5.10　R_BSP_SoftwareUnlock

ロックを解除します。

フォーマット

```
bool R_BSP_SoftwareUnlock(BSP_CFG_USER_LOCKING_TYPE * const plock);
```

パラメータ

plock: 解除するロックの構造体へのポインタ

戻り値

true: ロックの解除に成功。

false: ロックの解除に失敗。

プロパティ

locking.h でプロトタイプ宣言

locking.c に組み込まれる

説明

本関数は、R_BSP_SoftwareLock()関数を使って設定されたロックを解除します。ロックに関する詳細は 5.9 をご覧ください。

リエントラント

可

例

以下に、重要なコードに対し、ロックを使用する例を示します。

```
/* Used for locking critical section of code. */
static BSP_CFG_USER_LOCKING_TYPE g_critical_lock;

static bool critical_area_example (void)
{
    /* Try to acquire lock for executing critical section below. */
    if(false == R_BSP_SoftwareLock(&g_critical_lock))
    {
        /* Lock has already been acquired. */
        return false;
    }

    /* BEGIN CRITICAL SECTION. */

    /* Execute critical section. */
    ....

    /* END CRITICAL SECTION. */

    /* Release lock. */
    R_BSP_SoftwareUnlock(&g_critical_lock);

    return true;
}
```

参考情報

なし

5.11　R_BSP_HardwareLock

ハードウェアロックを予約します。

フォーマット

```
bool R_BSP_HardwareLock(mcu_lock_t const hw_index);
```

パラメータ

hw_index: ハードウェアロック配列から設定するロックへのポインタ

戻り値

true: ロックの使用が可能で設定に成功。

false: ロックは既に設定されていて、使用不可。

プロパティ

locking.h でプロトタイプ宣言

locking.c に組み込まれる

説明

本関数は、MCU のハードウェアリソースのロックを予約します。ロックへのポインタを送信する R_BSP_SoftwareLock()関数とは違って、MCU のハードウェアリソースごとに 1 つのロックを持つ配列へのインデックスを送信します。この配列はすべての FIT モジュールおよびユーザコード間で共有されますので、複数の FIT モジュール（およびユーザコード）で同じロックを使用することができます。使用可能なハードウェアリソースは、mcu_locks.h の mcu_lock_t enum で確認できます。これらの enum の数値も、ハードウェアロック配列へのインデックスです。本関数と R_BSP_SoftwareLock()関数では、同じメカニズムのロック機能が使用されます。

リエントラント

可

例

ここでは、RSPI チャネルへのアクセスを制御するために使用されたハードウェアロックの設定例を示します。

```
/******************************************************************************
* Function Name: R_RSPI_Send
* Description  : Send data over RSPI channel.
* Arguments    : channel -
*                   Which channel to use.
*              pdata -
*                   Pointer to data to transmit
*              bytes -
*                   Number of bytes to transmit
* Return Value : true -
*                   Data sent successfully.
*              false -
*                   Could not obtain lock.
******************************************************************************/
bool R_RSPI_Send(uint8_t channel, uint8_t * pdata, uint32_t bytes)
{
    mcu_lock_t rspi_channel_lock;

    /* Check and make sure channel is valid. */
    ...

    /* Use appropriate RSPI channel lock. */
    if (0 == channel)
    {
        rspi_channel_lock = BSP_LOCK_RSPI0;
    }
```

```
    else
    {
        rspi_channel_lock = BSP_LOCK_RSPI1;
    }

    /* Attempt to obtain lock so we know we have exclusive access to RSPI
       channel. */
    if (false == R_BSP_HardwareLock(rspi_channel_lock))
    {
        /* Lock has already been acquired by another task. Need to try again
           later. */
        return false;
    }

    /* Else, lock was acquired. Continue on with send operation. */
    ...

    /* Now that send operation is completed, release hold on lock so that other
       tasks may use this RSPI channel. */
    R_BSP_HardwareUnlock(rspi_channel_lock);

    return true;
}
```

参考情報

mcu_locks.h の mcu_lock_t enum の各エントリにはロックが割り当てられます。RX MCU では、各ロックに 4 バイトを必要とします。RAM の容量に問題がある場合、mcu_lock_t enum から不要なエントリを削除できます。例えば、CRC を使用しない場合、BSP_LOCK_CRC エントリを削除できます。1 つエントリを削除するごとに 4 バイトの容量を確保できます。

5.12　R_BSP_HardwareUnlock

ハードウェアロックを解除します。

フォーマット

```
bool R_BSP_HardwareUnlock(mcu_lock_t const hw_index);
```

パラメータ

hw_index: ハードウェアロック配列から解除するロックへのポインタ

戻り値

true: ロックの解除に成功。

false: ロックの解除に失敗。

プロパティ

locking.h でプロトタイプ宣言

locking.c に組み込まれる

説明

本関数は、R_BSP_HardwareLock()関数を使って設定されたハードウェアリソースのロックを解除します。ハードウェアロックに関する詳細は 5.11 をご覧ください。

リエントラント

可

例

以下の例は、ハードウェアリソースの配置の重複を防ぐために使用されたハードウェアロックを示しています。R_SCI_Open()では、全モジュールが SCI チャネルが使用されていることを確認できるようにロックを設定しています。R_SCI_Close()では、モジュールの使用を可能にするために、ロックを解除しています。

```
bool R_SCI_Open(uint8_t channel, ...)
{
    mcu_lock_t sci_channel_lock;

    /* Check and make sure channel is valid. */
    ...

    /* Use appropriate RSPI channel lock. */
    if (0 == channel)
    {
        sci_channel_lock = BSP_LOCK_SCI0;
    }
    else if (1 == channel)
    {
        sci_channel_lock = BSP_LOCK_SCI1;
    }
    ... continue for other channels ...

    /* Attempt to obtain lock so we know we have exclusive access to SCI
       channel. */
    if (false == R_BSP_HardwareLock(sci_channel_lock))
    {
        /* Lock has already been acquired by another task or another FIT module.
           Need to try again later. */
        return false;
    }

    /* Else, lock was acquired. Continue on initialization. */
    ...
}
```

```
bool R_SCI_Close(uint8_t channel, ...)
{
   mcu_lock_t sci_channel_lock;

   /* Check and make sure channel is valid. */
   ...

   /* Use appropriate RSPI channel lock. */
   if (0 == channel)
   {
      sci_channel_lock = BSP_LOCK_SCI0;
   }
   else if (1 == channel)
   {
      sci_channel_lock = BSP_LOCK_SCI1;
   }
   ... continue for other channels ...

   /* Clean up and turn off this SCI channel. */
   ....

   /* Release hardware lock for this channel. */
   R_BSP_HardwareUnlock(sci_channel_lock);
}
```

参考情報

mcu_locks.h の mcu_lock_t enum の各エントリにはロックが割り当てられます。RX MCU では、各ロックに 4 バイトを必要とします。RAM の容量に問題がある場合、mcu_lock_t enum から不要なエントリを削除できます。例えば、CRC を使用しない場合、BSP_LOCK_CRC エントリを削除できます。1 つエントリを削除するごとに 4 バイトの容量を確保できます。

5.13　R_BSP_InterruptWrite

割り込み用のコールバック関数を登録します。

フォーマット

```
bsp_int_err_t R_BSP_InterruptWrite(bsp_int_src_t vector,
                                   bsp_int_cb_t callback);
```

パラメータ

vector: コールバックを登録する割り込みを指定（4.10.6 参照）。

callback: 割り込み発生時にコールされる関数へのポインタ（4.10.5 参照）。

戻り値

BSP_INT_SUCCESS: コールバックの登録に成功。

BSP_INT_ERR_INVALID_ARG: 無効な関数のアドレス入力。以前に登録された関数の登録は解除。

プロパティ

mcu_interrupts.h でプロトタイプ宣言

mcu_interrupts.c に組み込まれる

説明

割り込み用にコールバック関数を登録します。FIT_NO_FUNC、NULL や、無効な関数のアドレスがコールバックの引数として渡された場合、それ以前に登録されたコールバックは全て登録が解除されます。

本関数で処理される割り込み要求が発生した場合、割り込み処理は、有効なコールバック関数が登録されているかどうかを確認します。有効であることが確認されると、コールバック関数が呼び出されます。有効であることが確認されなかった場合は、該当のフラグをクリアし、処理を終了します。

割り込み処理に必要のなくなった登録済みのコールバック関数がある場合、ベクタのパラメータに FIT_NO_FUNC を指定して、本関数を再度呼び出します。

リエントラント

不可

例

```
/* Prototype for callback function. */
void bus_error_callback(void * pdata);

void main (void)
{
   bsp_int_err_t err;

   /* Register bus_error_callback() to be called whenever a bus error occurs */
   err = R_BSP_InterruptWrite(BSP_INT_SRC_BUS_ERROR, bus_error_callback);

   if (BSP_INT_SUCCESS != err)
   {
      /* Error in registering callback. Alert user. */
      ...
   }
}

void bus_error_callback (void * pdata)
{
   /* Bus error has occurred. Handle accordingly. */
   ...
}
```

参考情報

FIT_NO_FUNC で定義されたアドレスへのアクセスはバスエラーを発生させ、ユーザが認識しやすいので、NULL よりも FIT_NO_FUNC を使用する方が適しています。NULL は多くの場合 0 と解釈されますが、0 は RX MCU では有効なアドレスです。

5.14　R_BSP_InterruptRead

割り込み用のコールバック関数が登録されている場合、それを読み出します。

フォーマット

```
bsp_int_err_t R_BSP_InterruptRead(bsp_int_src_t vector,
                                  bsp_int_cb_t * callback);
```

パラメータ

vector: コールバックを読み出す割り込みを指定（4.10.6 参照）。

callback: コールバックのアドレスの格納先へのポインタ（4.10.5 参照）。

戻り値

BSP_INT_SUCCESS: コールバックのアドレスが正しく戻された。

BSP_INT_ERR_NO_REGISTERED_CALLBACK: 割り込み要因に対して、有効なコールバック関数が登録されていない。

プロパティ

mcu_interrupts.h でプロトタイプ宣言

mcu_interrupts.c に組み込まれる

説明

該当の割り込みに対してコールバック関数が登録済みの場合、そのコールバック関数のアドレスを戻します。コールバック関数が登録されていない場合、エラーが戻され、callback のアドレスには何も格納されません。

リエントラント

不可

例

```
/* This function handles bus error interrupts. The address for this function
   is located in the bus error interrupt vector. */
void bus_error_isr (void)
{
   bsp_int_err_t  err;
   bsp_int_cb_t * user_callback;

   /* Bus error has occurred, see if a callback function has been registered */
   err = R_BSP_InterruptRead(BSP_INT_SRC_BUS_ERROR, user_callback);

   if (BSP_INT_SUCCESS == err)
   {
      /* Valid callback function found. Call it. */
      user_callback ();
   }

   /* Clear bus error flags. */
   ...
}
```

参考情報

なし

5.15　R_BSP_SoftwareDelay

指定した単位の時間だけ遅延させてから戻ります。

フォーマット

```
bool R_BSP_SoftwareDelay(uint32_t delay, bsp_delay_units_t units)
```

パラメータ

delay: 遅延させる単位の数値。

units: 指定した単位のベース。有効な値は次のとおりです。

BSP_DELAY_MICROSECS, BSP_DELAY_MILLISECS, BSP_DELAY_SECS

戻り値

true: 遅延が実行された場合。

false: 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

プロパティ

r_bsp_common.h でプロトタイプ宣言

r_bsp_common.c に組み込まれる

説明

これは、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。

実際の遅延時間は指定した時間にオーバヘッドを加えたものになります。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。

リエントラント

不可

例

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

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

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

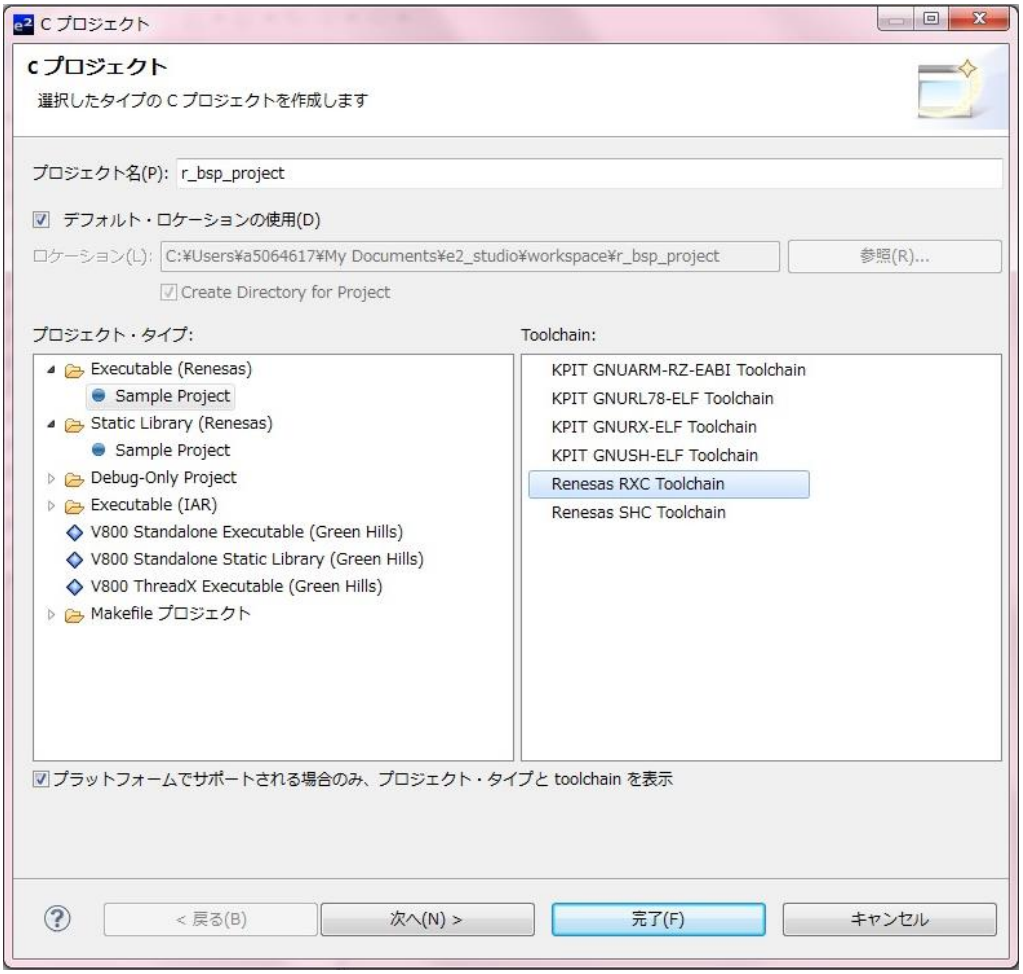
6. プロジェクトのセットアップ

ここでは e² studio プロジェクトの作成方法と r_bsp をプロジェクトに追加する方法を説明します。

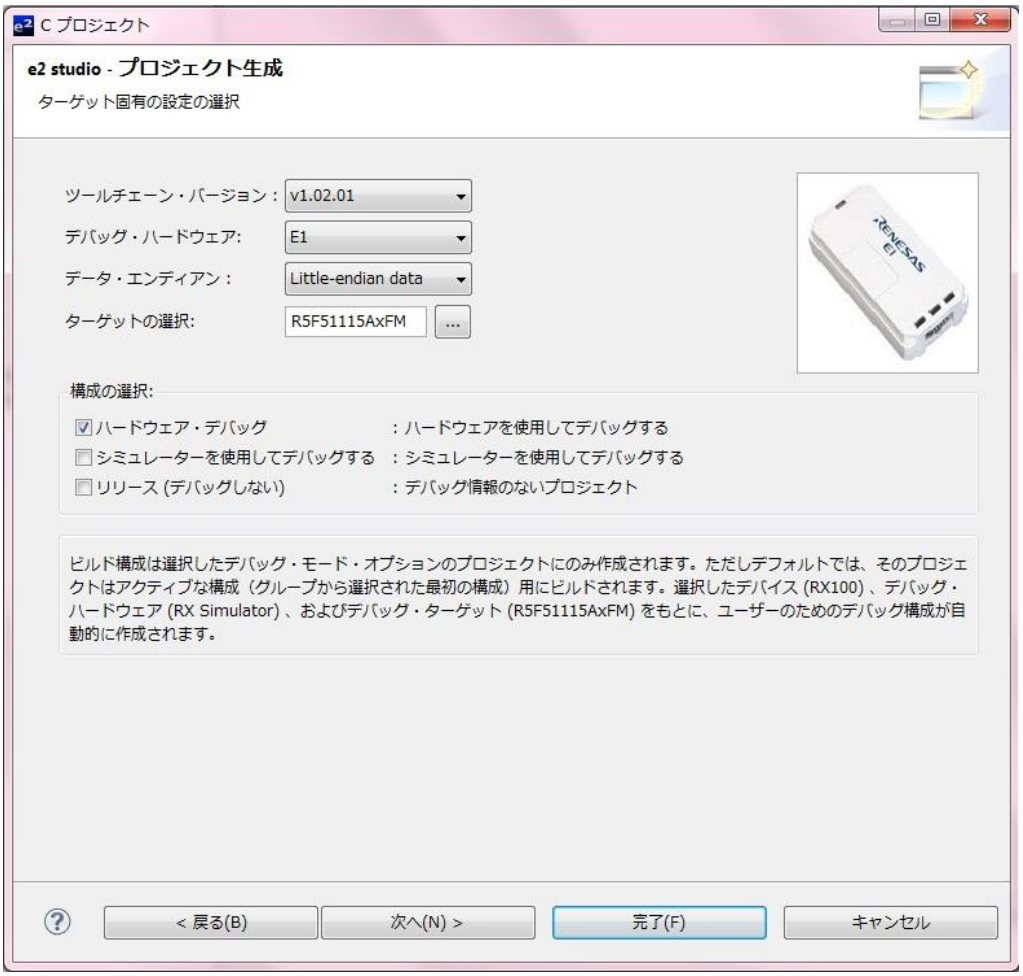
6.1 空プロジェクトを作成する

e² studio を開始するには、プロジェクトを作成し、それを変更します。ここでは、RSK RX111 を例に、プロジェクトを作成します。

1. e² studio のワークスペースを開きます。
2. 「ファイル(F) >> 新規(N) >> C プロジェクト」をクリックします。
3. プロジェクト名を入力します。「プロジェクト・タイプ」で「Sample Project」を選択します。「ツールチェーン」で、「Renesas RXC Toolchain」を選択します。「次へ(N)」をクリックします。

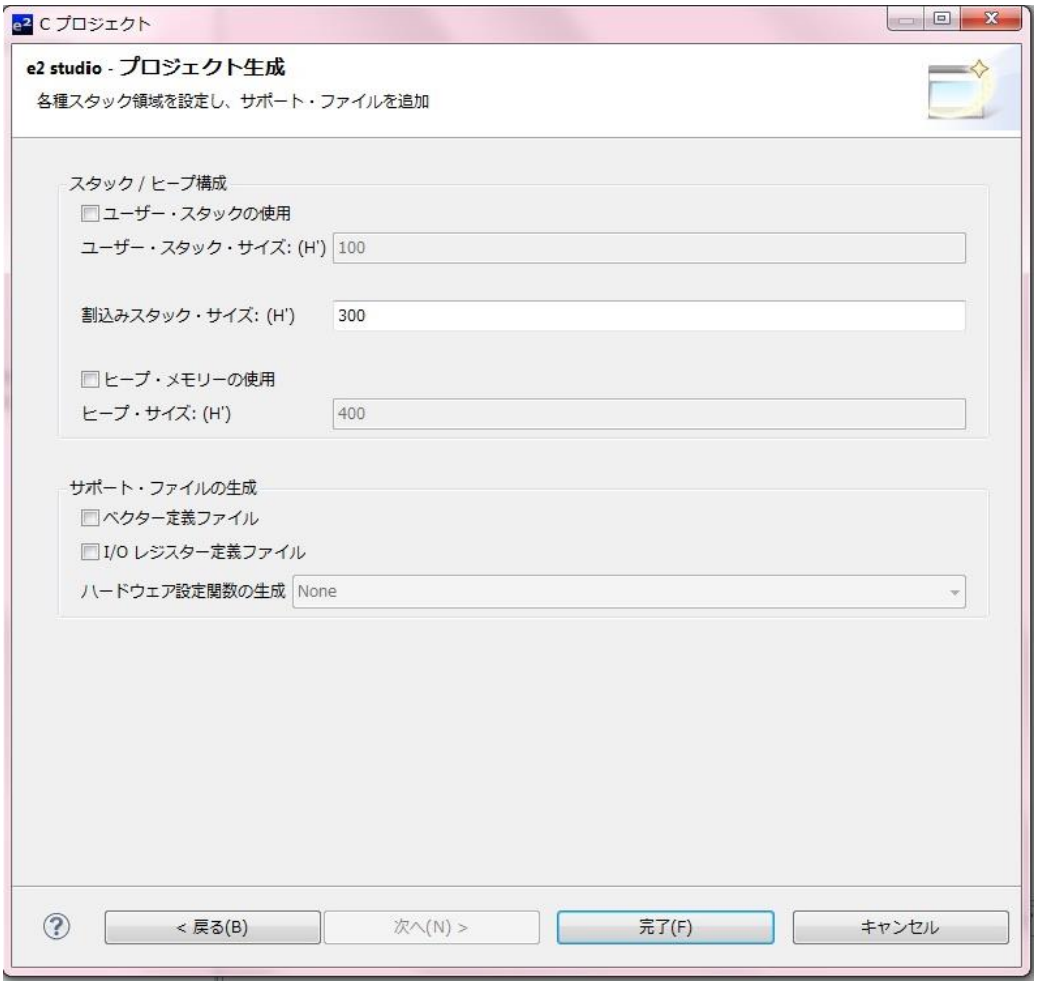


4. 使用するデバッグハードウェアと MCU を選択します。

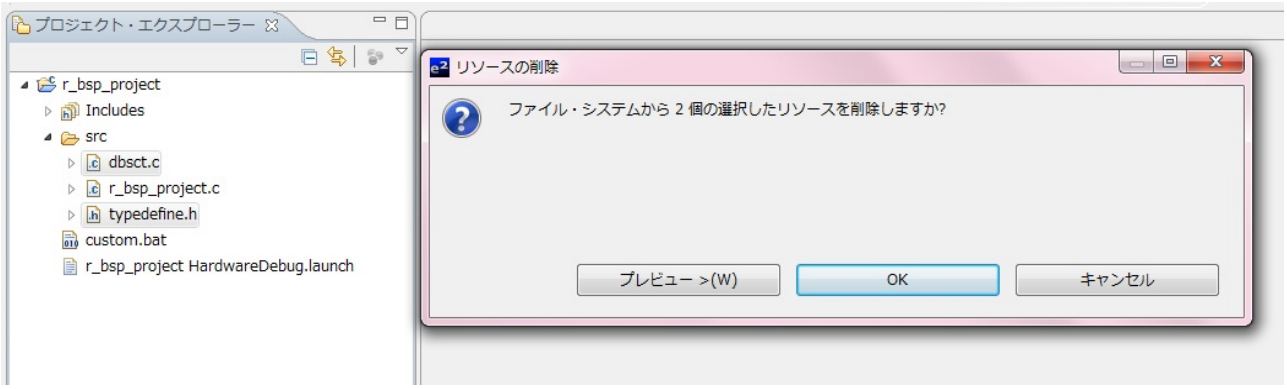


5. 「コード生成設定」ウィンドウでは、「コード生成を使用する」にチェックをしないでください。「次へ(N)」をクリックしてください。
6. 「追加 CPU オプションの選択」ウィンドウで、必要に応じて設定を行い、「次へ(N)」をクリックします。
7. 「グローバル・オプション設定」ウィンドウで、必要に応じて設定を行い、「次へ(N)」をクリックします。

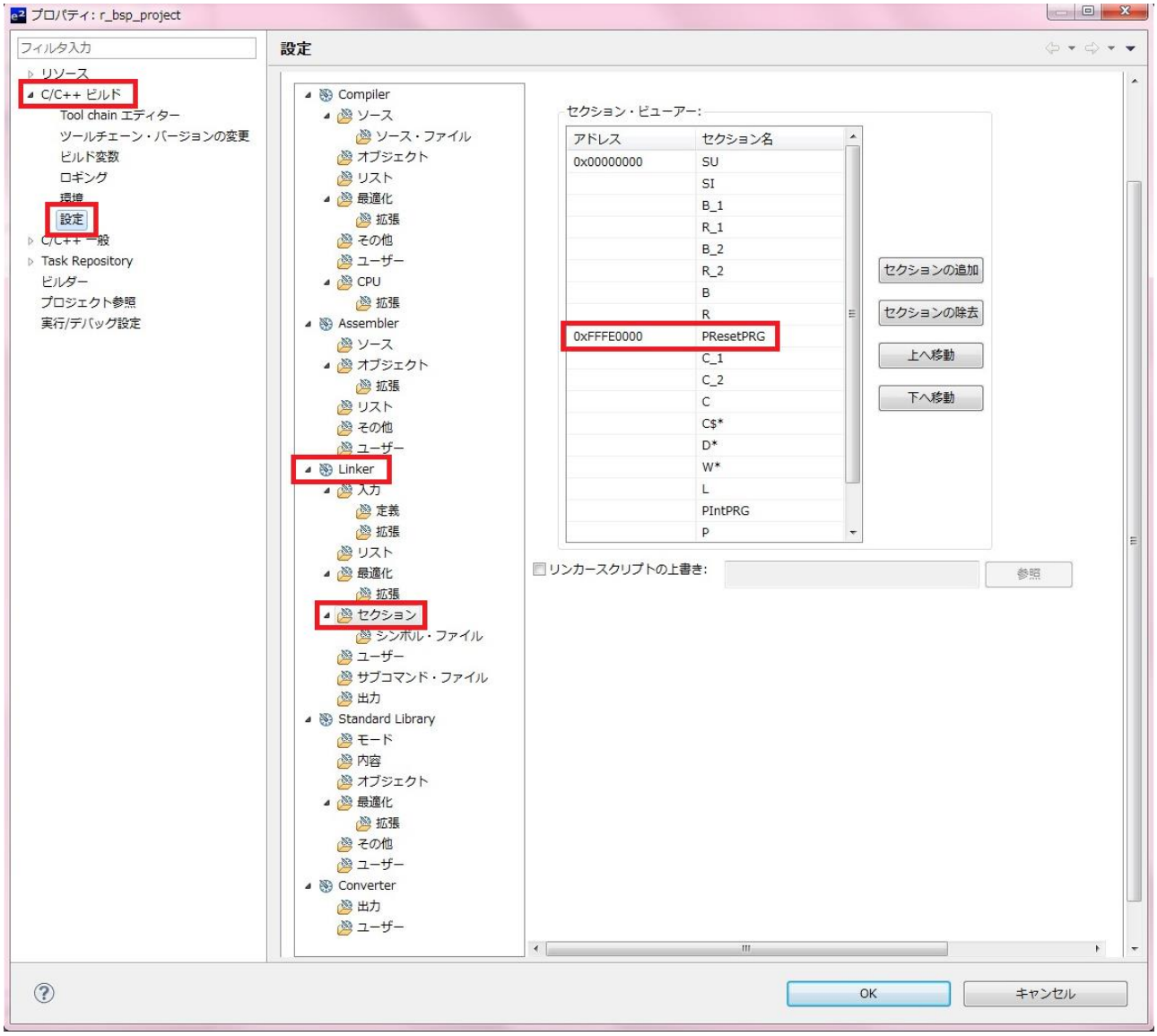
8. 「標準ヘッダー・ファイル」ウィンドウの「ライブラリー構成」で、「C(C99)」を選択します。使用するライブラリを設定し「次へ(N)」をクリックします。
9. 以下に示すウィンドウで、すべてのチェックボックスの選択を外します。



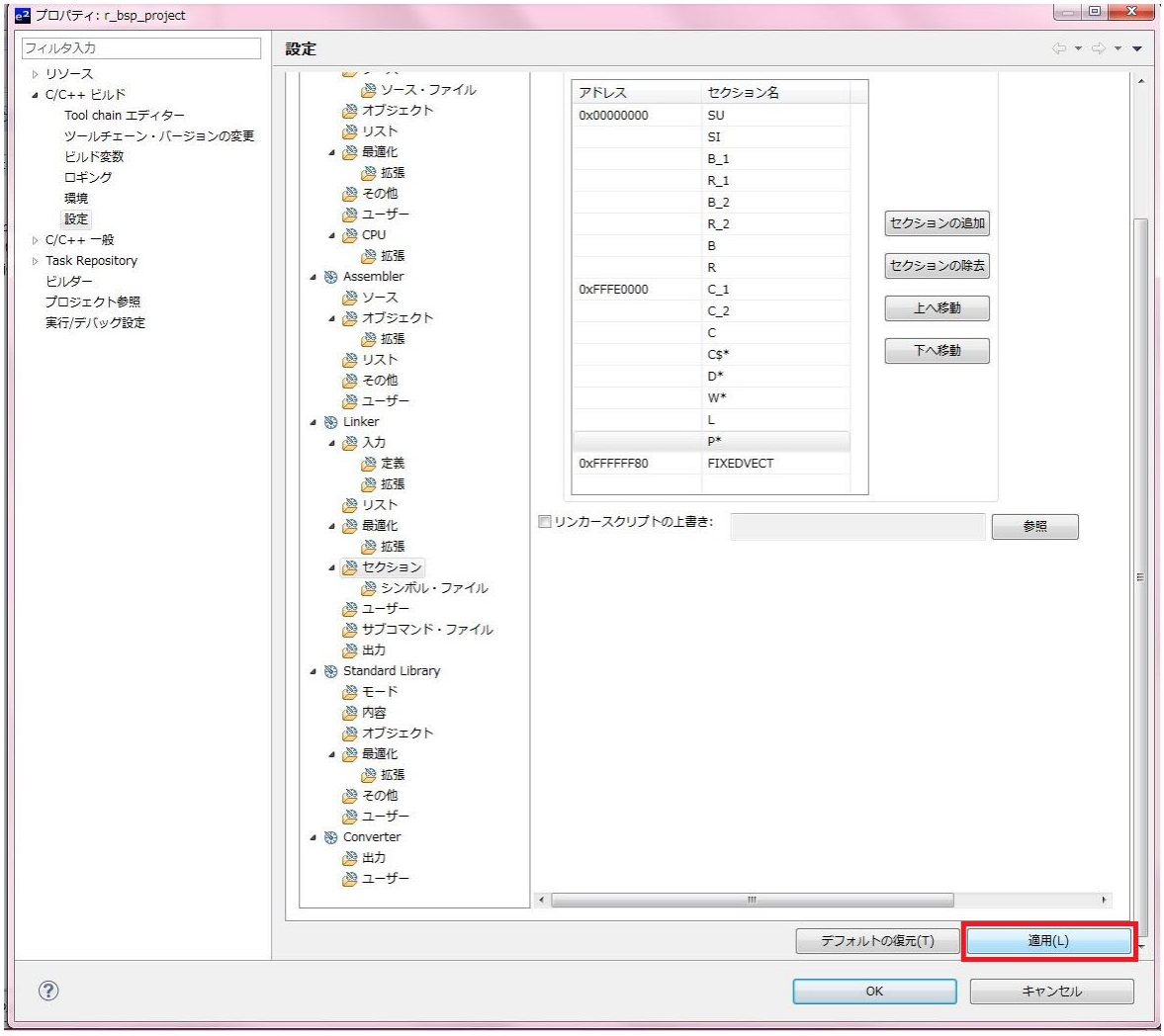
10. 「完了(F)」ボタンをクリックします。「要約」ウィンドウが開きますので OK をクリックします。
11. 「プロジェクト・エクスプローラー」ペインで、新規に作成したプロジェクトを開きます。src ディレクトリを開き、main()関数を含むファイルを除いて全てのファイルを削除します。ここでは、dbsct.c と typedefine.h の 2 ファイルを削除します。



12. 「プロジェクト・エクスプローラー」ペインでプロジェクトを右クリックし、「プロパティ(R)」をクリックします。
13. ここからはリンカセクションの設定を行います。ここで行う主な変更は、r_bsp で使用しないデフォルトのリンカセクションを削除することです。
14. 「C/C++ビルド」を開き、「設定」をクリックします。
15. 「ツール設定」で、「Linker >> セクション」を選択します。



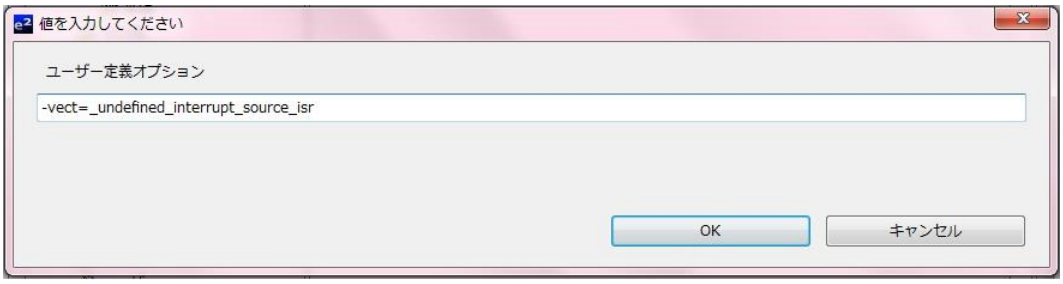
16. 「セクション・ビューアー」ペインで、「PResetPRG」セクションに割り当てられたアドレスを書き留めておきます。このアドレスがご使用の MCU の ROM の先頭アドレスになります。アドレスを書き留めたら、「PResetPRG」セクションをクリックして、「セクションの除去」をクリックします。
17. 「PResetPRG」のすぐ下にあったセクション（ここでは C_1）をクリックして、アドレスを先ほど書き留めたアドレスに変更します。
18. 「PIntPRG」セクションをクリックして、「セクションの除去」をクリックします。
19. 「P」セクションをクリックし、「上へ移動」ボタンをクリックします。これによって、セクションからアドレスが削除され、前のセクションブロックと結合されます。
20. 「P」セクションをクリックし、「P*」に変更します。‘*’を付けることでワイルドカードのように扱われ、プロジェクトで使用される「P」セクションがすべてピックアップされます。
21. 「FIXEDVECT」セクションのアドレスが 0xFFFFFF80 になっていることを確認します。
22. ここで「適用(L)」ボタンを必ず押してください。「適用(L)」ボタンが表示されていない場合は、スクロールバーを使ってウィンドウを右にずらして表示させてください。
23. ここまでで、リンカ画面は以下のようになっているはずです。



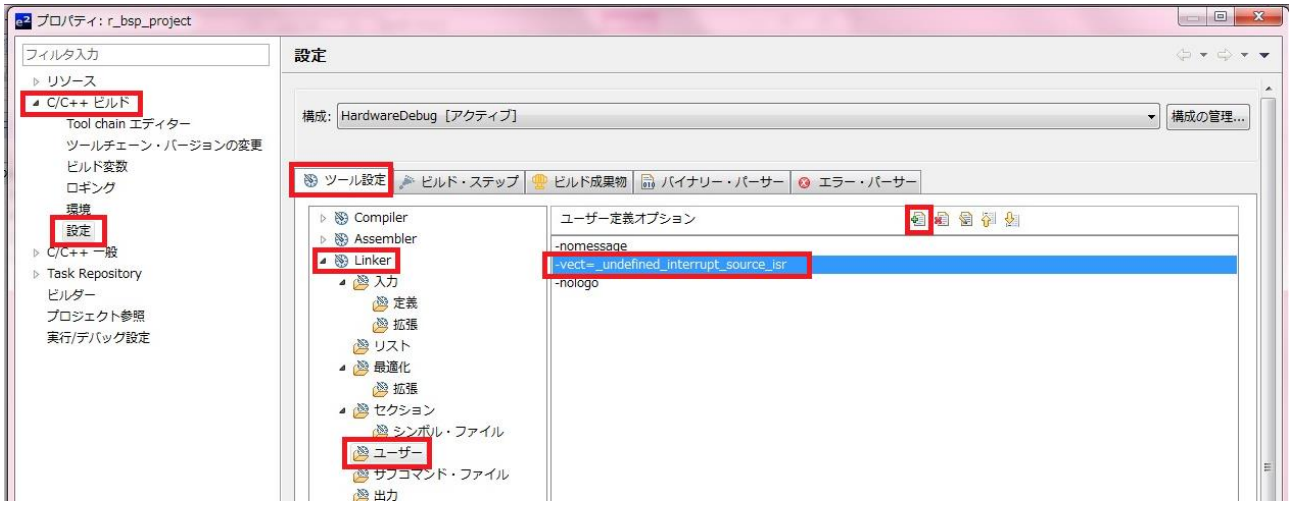
24. リンカを設定して、undefined_interrupt_source_isr()関数のアドレスで未使用の割り込みベクタを埋めます。「ツール設定」から、「Linker >> ユーザー」を選択します。

25. 「追加」ボタン（緑の‘+’）をクリックし、開いたウィンドウで以下を入力します。

```
-vect=_undefined_interrupt_source_isr
```



26. OK をクリックして、ウィンドウを閉じます。リストにオプションが追加されていることを確認して「適用(L)」をクリックします。

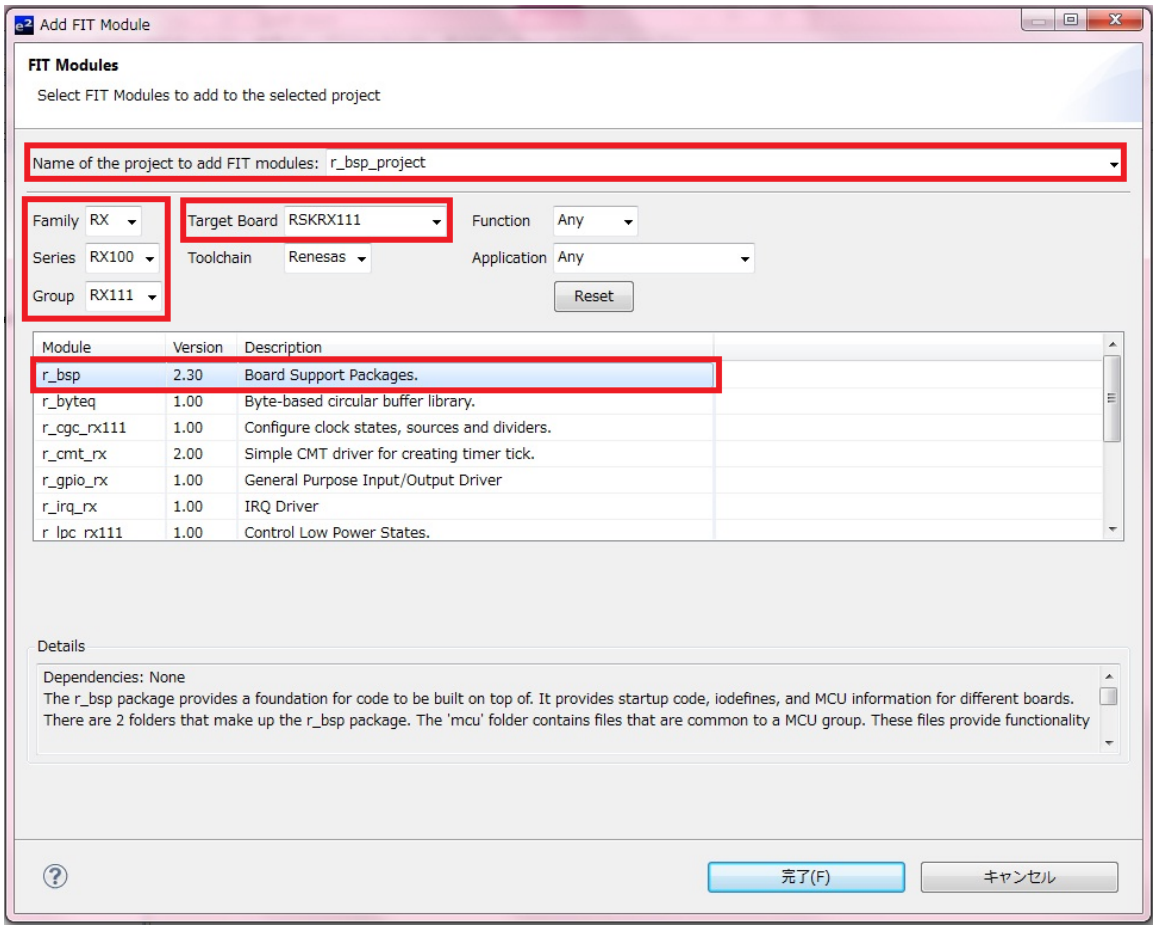


27. OK をクリックして、プロジェクトに戻ります。

6.2　e² studio FIT プラグインを使って r_bsp を追加する

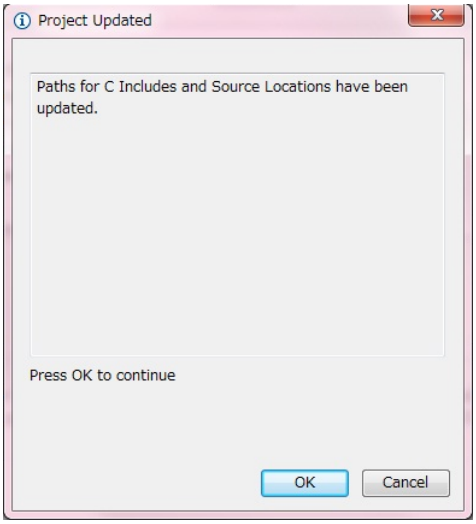
ここまでで、r_bsp コードを追加できる空の e² studio プロジェクトができました。追加する方法は 2 種類あります。1 つめは、FIT プラグインを使用する方法で、このセクションで説明します。2 つめは r_bsp を手動で追加する方法で、セクション 7 で説明します。

1. 「ファイル(F) >> 新規(N) >> Renesas FIT Module」をクリックして、FIT プラグインを開きます。
2. 「Name of the project to add FIT modules」で、使用するプロジェクトが表示されていることを確認します。
3. 「Family」、「Series」、「Group」、「Target」、「Board」ドロップダウンリストからオプションを選択して、使用するボードと MCU を選択します。例では RSKRX111 を使用しています。
4. 「Module」リストから使用する r_bsp のバージョンをクリックします。

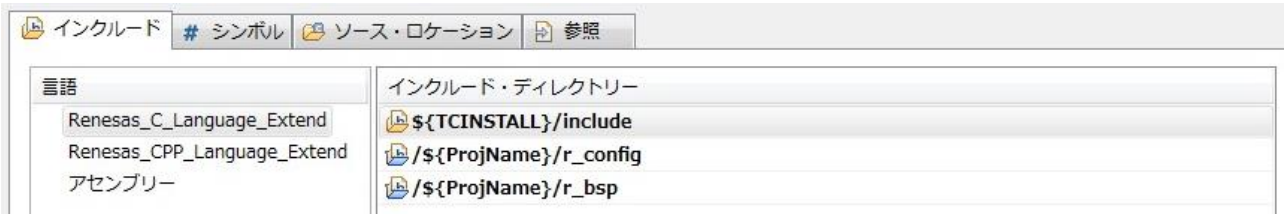


5. 「完了（F)）ボタンをクリックします。

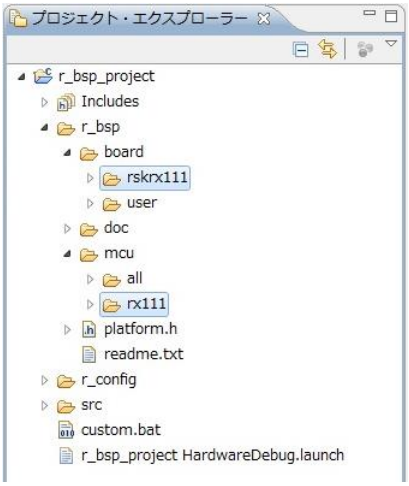
6. プラグインが新規モジュール用にインクルードパスを更新したことを示すポップアップウィンドウが表示されますので、OK をクリックします。



7. プラグインがプロジェクトのインクルードパスを表示します。r_bsp と r_config フォルダのインクルードがあることを確認します。「適用(L)」をクリックし、OK をクリックします。



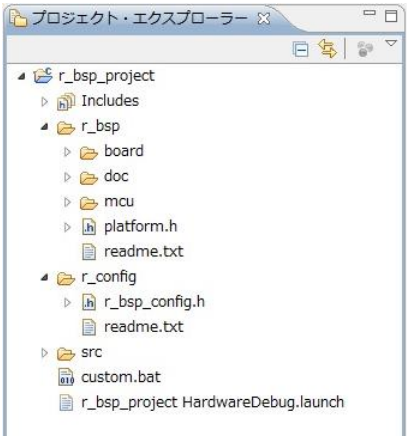
8. プロジェクトに r_bsp フォルダと r_config フォルダがあることを確認します。
9. r_bsp フォルダを開いて、正しいボードと MCU がコピーされていることを確認します。



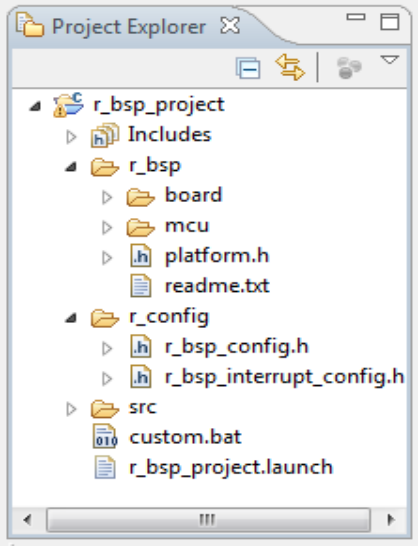
10. 使用するボードは platform.h ヘッダファイルにて選択する必要があります。platform.h を開き、使用するボードの#include のコメントを解除します。ここでは、RSKRX111 を使用しますので、“./board/rskrx111/r_bsp.h”の#include のコメントを解除します。

```
/* RSKRX63N */
//#include "./board/rskrx63n/r_bsp.h"

/* RSKRX63T_64PIN */
//#include "./board/rskrx63t_64pin/r_bsp.h"

/* RSKRX63T_144PIN */
//#include "./board/rskrx63t_144pin/r_bsp.h"

/* RDKRX63N */
//#include "./board/rdkrx63n/r_bsp.h"

/* RSKRX210 */
//#include "./board/rskrx210/r_bsp.h"

/* RSKRX111 */
#include "./board/rskrx111/r_bsp.h"
```

11. r_bsp を設定するためには、r_bsp_config.h ファイルを作成する必要があります。board フォルダから r_bsp_config_reference.h ファイルをコピーし、r_config フォルダにペーストします。r_config フォルダのファイルを右クリックし、「名前変更(M)」をクリックします。ファイル名を r_bsp_config.h に変更します。



12. r_bsp_config.h で必要な箇所を変更し、r_bsp をご使用のボードに合わせて設定します。
13. RX64M、RX65N および RX71M グループの MCU の場合、bsp を設定するには、r_bsp_interrupt_config.h ファイルも作成する必要があります。board フォルダから r_bsp_interrupt_config_reference.h ファイルをコピーし、r_config フォルダにペーストします。r_config フォルダのファイルを右クリックし、「名前変更(M)」をクリックします。ファイル名を r_bsp_interrupt_config.h に変更します。

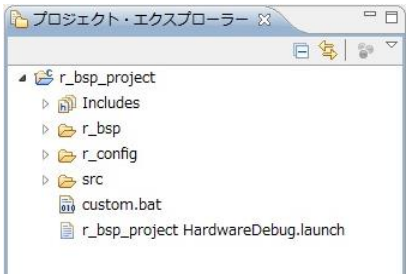


14. r_bsp_interrupt_config.h ファイルで必要な箇所を変更し、ご使用の RX64M、RX65N または RX71M ボードに合わせて選択型割り込みを設定します。
15. プロジェクトをビルドします。

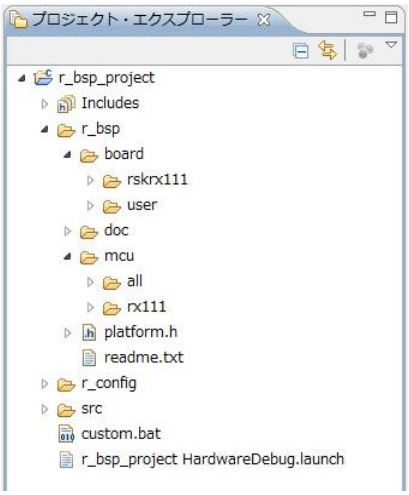
7. 手動で r_bsp を追加する

本セクションでは、e² studio プロジェクトに手動で r_bsp を追加する方法（FIT プラグインを使用しない方法）を説明します。

1. r_bsp フォルダを e² studio プロジェクトの直下にコピーします。Windows 上で r_bsp モジュールのフォルダを右クリックし「コピー(C)」をクリックしたら、e² studio のプロジェクトを右クリックし、「貼り付け(P)」をクリックします。



2. r_bsp 内の board フォルダを開き、使用するボード以外のフォルダをすべて削除します。user ディレクトリは残しておいて、カスタム BSP を作成するときに使用しても構いません。
3. r_bsp 内の mcu フォルダを開き、使用する MCU グループおよび all フォルダ以外のフォルダをすべて削除します。



4. すべての FIT モジュールのコンフィギュレーションファイルを格納するディレクトリを作成することが推奨されます。これらのファイルを 1 カ所に集約することで、ファイルの検索やバックアップが容易になります。このフォルダはデフォルトでは r_config というフォルダ名です。r_config フォルダが r_bsp zip ファイルに含まれていなかった場合は、ここで作成します。r_config フォルダを作成する場合は、プロジェクトを右クリックし、「新規(N) >> フォルダ」を選択します。ポップアップウィンドウが表示されますので、フォルダ名に‘r_config’と入力し「完了」ボタンをクリックします。
5. r_bsp フォルダと r_config フォルダのインクルードパスを設定します。プロジェクトを右クリックし、「プロパティ(R)」をクリックします。
6. 「設定」タブで、「Compiler >> ソース」を選択します。

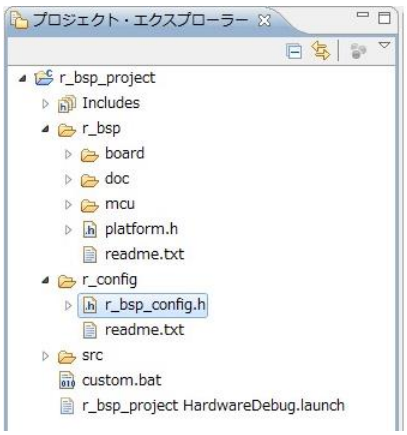
7. 「インクルード・ファイル・ディレクトリー」ボックスで、「追加」ボタンをクリックします。

8. 「ディレクトリー・パスの追加」ウィンドウが表示されますので、「ワークスペース」ボタンをクリックします。
9. 「フォルダの選択」ウィンドウで、r_bsp フォルダを選択し OK をクリックします。

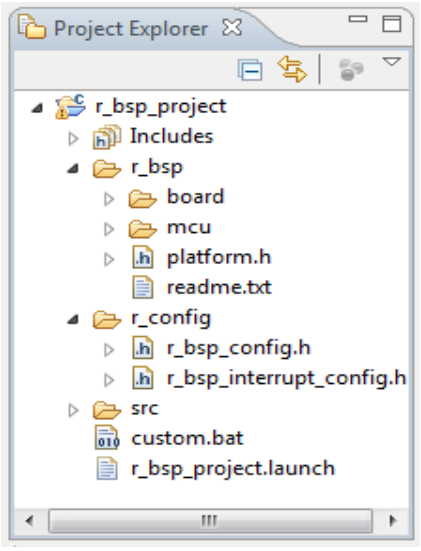
10. ウィンドウの表示が上記のようになっていることを確認して OK をクリックします。
11. メインの「プロパティ」ウィンドウに戻り、r_bsp フォルダのインクルードパスがあることを確認します。
12. 同様の方法で、r_config フォルダのインクルードパスも追加します。
13. メインの「プロパティ」ウィンドウに戻り、r_config フォルダのインクルードパスがあることを確認し、「適用(L)」をクリックします。OK をクリックして、プロジェクトに戻ります。
14. 使用するボードは platform.h ヘッダファイルにて選択する必要があります。platform.h を開き、使用するボードの#include のコメントを解除します。ここでは、RSKRX111 を使用しますので、“./board/rskrx111/r_bsp.h”の#include のコメントを解除します。

```
/* RSKRX63N */
//#include "./board/rskrx63n/r_bsp.h"

/* RSKRX63T_64PIN */
//#include "./board/rskrx63t_64pin/r_bsp.h"

/* RSKRX63T_144PIN */
//#include "./board/rskrx63t_144pin/r_bsp.h"

/* RDKRX63N */
//#include "./board/rdkrx63n/r_bsp.h"

/* RSKRX210 */
//#include "./board/rskrx210/r_bsp.h"

/* RSKRX111 */
#include "./board/rskrx111/r_bsp.h"
```

15. r_bsp を設定するためには、r_bsp_config.h ファイルを作成する必要があります。board フォルダから r_bsp_config_reference.h ファイルをコピーし、r_config フォルダにペーストします。r_config フォルダにあるファイルを右クリックし、「名前変更(M)」をクリックします。ファイル名を r_bsp_config.h に変更します。



16. r_bsp_config.h で必要な箇所を変更し、r_bsp をご使用のボードに合わせて設定します。
17. RX64M、RX65N および RX71M MCU の場合、bsp を設定するには、r_bsp_interrupt_config.h ファイルも作成する必要があります。board フォルダから r_bsp_interrupt_config_reference.h ファイルをコピーし、r_config フォルダにペーストします。r_config フォルダのファイルを右クリックし、「名前変更(M)」をクリックします。ファイル名を r_bsp_interrupt_config.h に変更します。

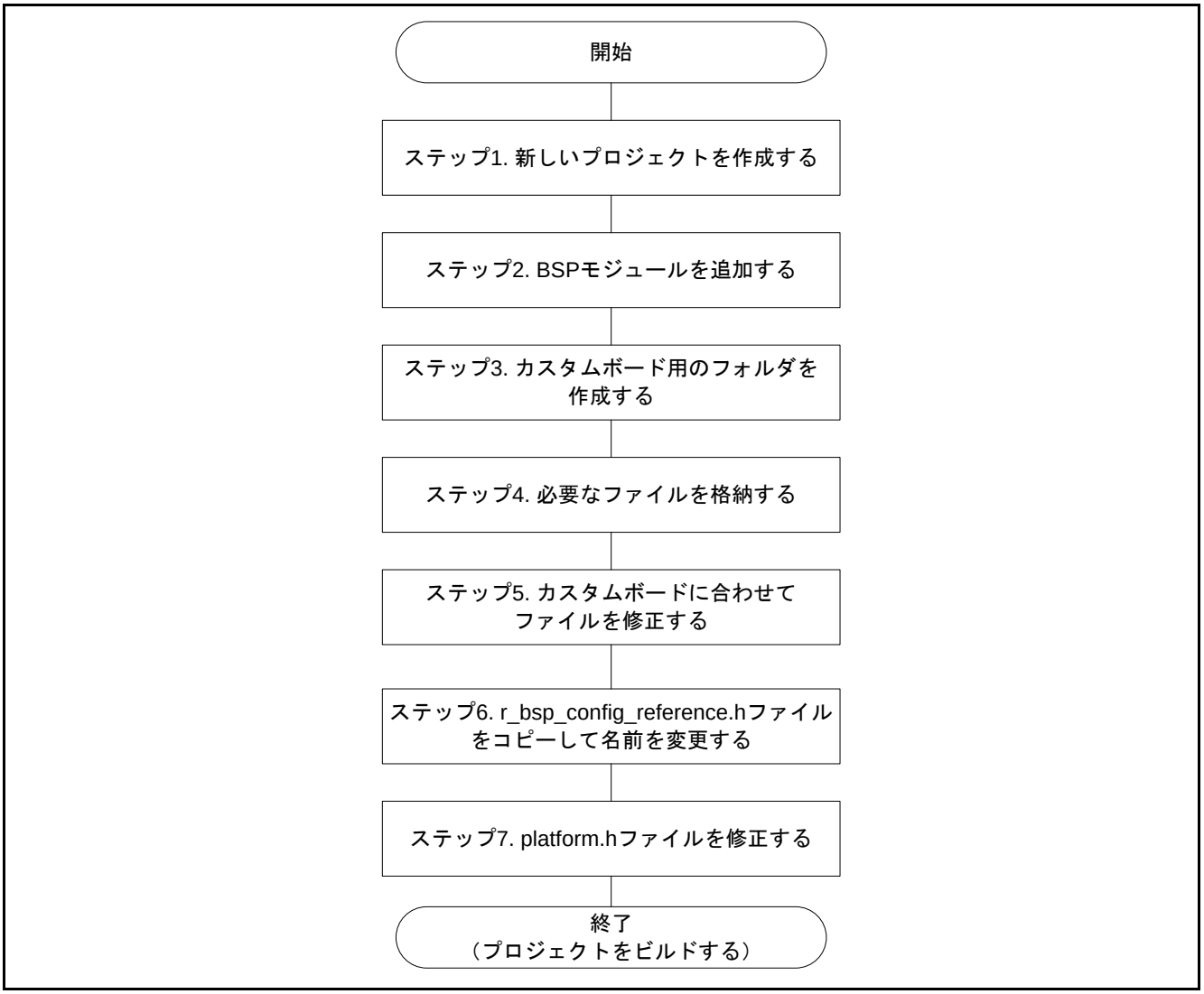


18. r_bsp_interrupt_config.h ファイルで必要な箇所を変更し、ご使用の RX64M、RX65N または RX71M ボードに合わせて選択型割り込みを設定します。
19. プロジェクトをビルドします。

7.1　カスタムボード用の BSP モジュールを作成する

ユーザがカスタムボード用の専用 r_bsp（カスタム BSP）を作成できるように r_bsp が提供されます。このセクションでは、カスタム BSP を使用するときに新しいプロジェクトを作成およびビルドする方法について説明します。本書では、RX111 MCU を例として使用します。

以下の図は、新しい bsp を作成するための手順を示しています。



ステップ 1.　新しいプロジェクトを作成する（必須）

新しいプロジェクトを作成するには、「ボードサポートパッケージモジュール Firmware Integration Technology（R01AN1685）」アプリケーションノートの「空プロジェクトを作成する」を参照してください。

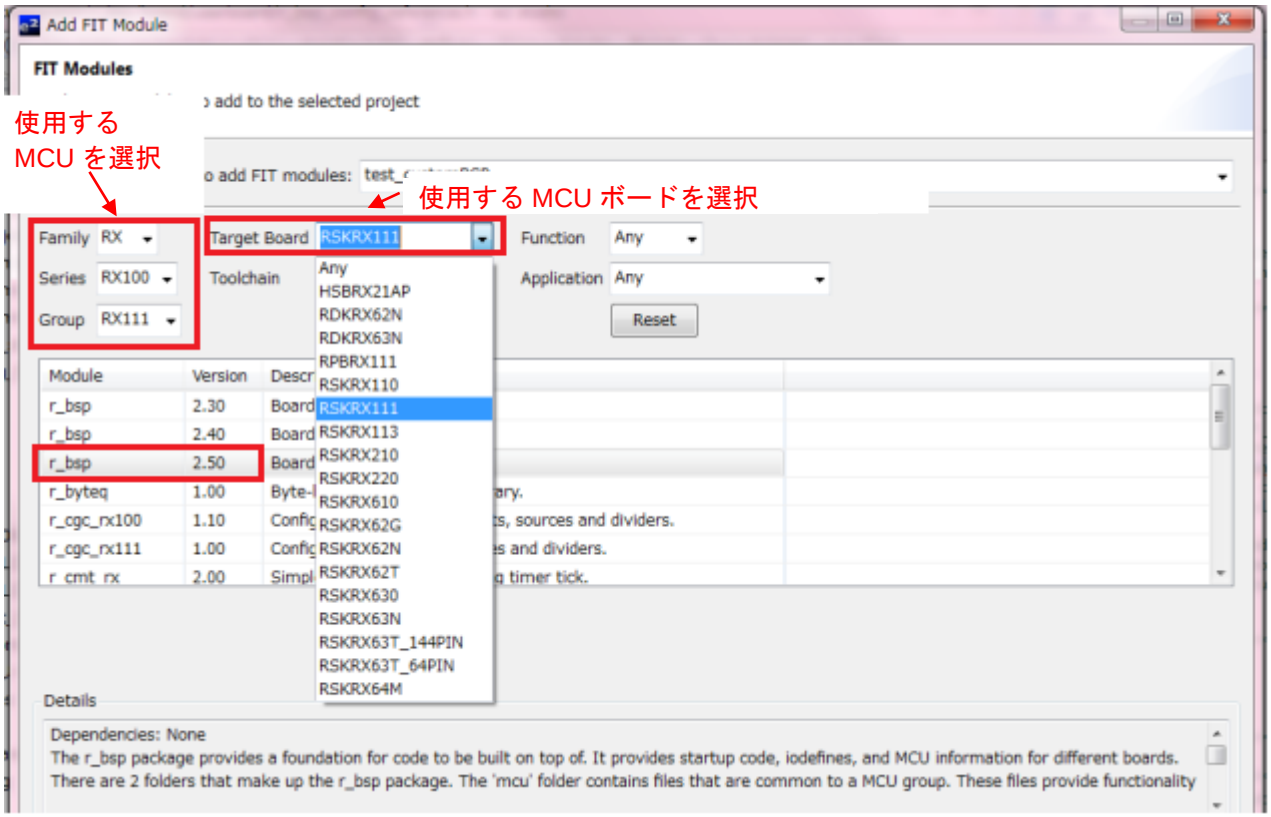
ステップ 2.　BSP モジュールを追加する（必須）

ステップ 1 で作成した新しいプロジェクト(ユーザプロジェクト)に BSP モジュールを追加するには、「ボードサポートパッケージモジュール Firmware Integration Technology（R01AN1685）」アプリケーションノートの「e² studio FIT プラグインを使って r_bsp を追加する」を参照してください。

FIT プラグイン上で BSP モジュールを追加するときには以下のオプションを選択してください。

- ファミリ、シリーズ、グループ：使用する MCU
- ターゲットボード：使用する MCU ボード

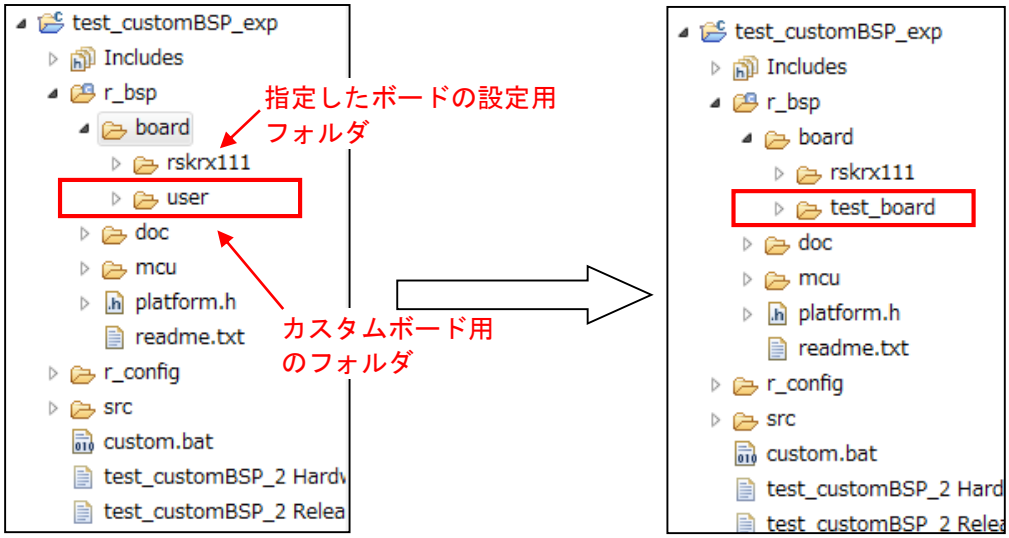
たとえば、ユーザボードを作成するのに RX111 を使用するときには、RSKRX111 を選択します。ここで適切なオプションを選択することで、カスタムボード用のボードフォルダを簡単に作成することができます。



ステップ 3.　カスタムボード用のフォルダを作成する

これで、r_bsp フォルダがユーザプロジェクトに表示されるようになります。以下では、r_bsp フォルダの下にあるボードフォルダを変更してカスタム BSP を作成しています。mcu フォルダ内のコードは変更を必要としません。

1. ステップ 2 で指定したボードフォルダ（ここでは rskrx111）とユーザフォルダが、r_bsp フォルダの下のボードフォルダに生成されていることを確認します。
2. カスタムボード用のフォルダとしてユーザフォルダを使用します（オプション）。
   フォルダの名前を変更します（オプション）。フォルダ名の変更は必須ではありません。



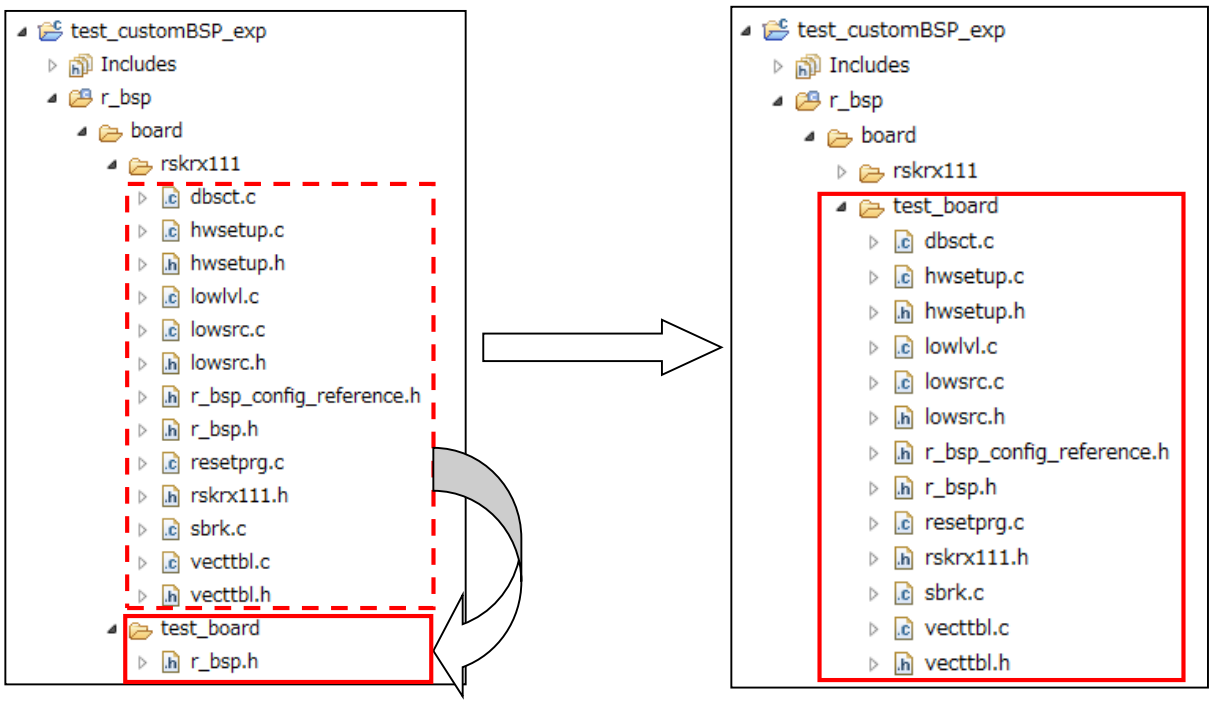
BSP モジュールが追加された後のフォルダ構造

フォルダ名をカスタムボード用に変更したときの例

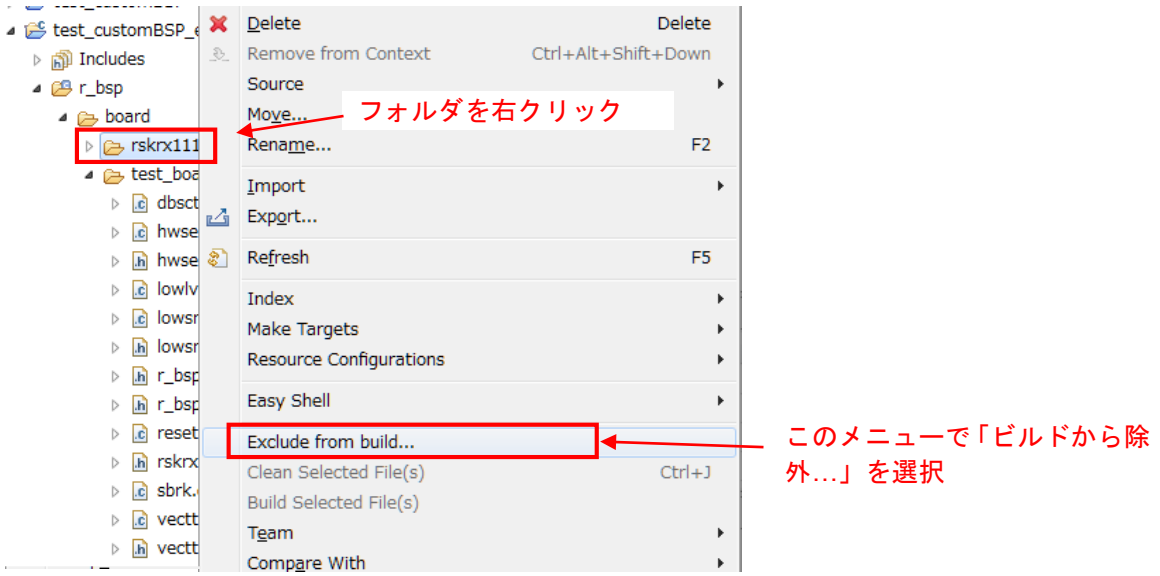
ステップ 4.　必要なファイルを格納する（必須）

ステップ 3 で作成したフォルダに必要なファイルを格納します。

1. rskrx111 フォルダ内のすべてのファイルをコピーしてカスタムボード用のフォルダに貼り付けます。これで r_bsp.h ファイルは上書きされます。



2. rskrx111 フォルダをビルドから除外します。
（このフォルダは、カスタムボード用のフォルダを作成した後、不要であれば削除できます。）



ステップ 5.　カスタムボードに合わせてファイルを修正する（必須）

カスタムボードに合わせて次の 4 つのファイルを修正します。

1.　hwsetup.c

このファイルは次の 4 つの関数を実行します。

- **関数：output_ports_configure**

この関数は、LED、スイッチ、SCI、および ADC で使用するポートを初期化します。
使用するボードに応じて、以下の手順のいずれかでポートを設定する必要があります。

この関数で端子を設定しない場合

1)　output_ports_configure 関数の関数宣言をコメントアウトするか削除します。
2)　hardware_setup 関数で呼び出される output_ports_configure 関数を削除します。
3)　output_ports_configure 関数をコメントアウトするか削除します。
　次に「2. *board_specific_defines*.h」に記載した設定も行います。

この関数で端子を設定する場合

1)　output_ports_configure 関数のソースコードをコメントアウトするか削除します。
2)　使用するボードに応じて端子を設定します。

- **関数：bsp_non_existent_port_init**

この関数は、存在しないポートを初期化します。この関数では追加の処理は不要です。

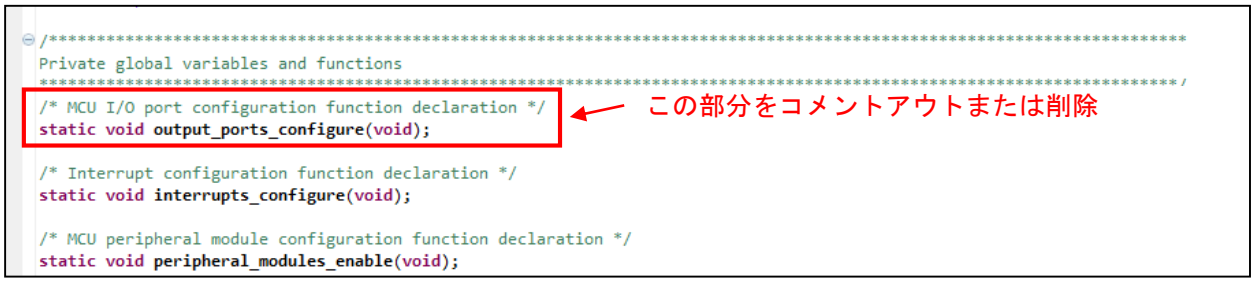
- **関数：interrupts_configure**

この関数は、メイン関数に先立って実施される割り込みの設定を行います。
このような設定が必要なとき、この関数でその設定を追加します。

- **関数：peripheral_modules_enable**

この関数は、メイン関数に先立って実施される周辺関数の設定を行います。
このような設定が必要なとき、この関数でその設定を追加します。

output_ports_configure 関数で端子を設定しないときの処理の例を以下に示します。



```
/***********************************************************************************************************************
* Function name: hardware_setup
* Description  : Contains setup functions called at device restart
* Arguments    : none
* Return value : none
***********************************************************************************************************************/
void hardware_setup(void)
{
    output_ports_configure();
    interrupts_configure();
    peripheral_modules_enable();
    bsp_non_existent_port_init();
}
```

この行をコメントアウトまたは削除

```
static void output_ports_configure(void)
{
    /* Enable LEDs. */
    /* Start with LEDs off. */
    LED0 = LED_OFF;
    LED1 = LED_OFF;
    LED2 = LED_OFF;
    LED3 = LED_OFF;

    /* Set LED pins as outputs. */
    LED0_PDR = 1;
    LED1_PDR = 1;
    LED2_PDR = 1;
    LED3_PDR = 1;

    /* Enable switches. */
    /* Set pins as inputs. */
    SW1_PDR = 0;
    SW2_PDR = 0;
    SW3_PDR = 0;

    /* Set port mode registers for switches. */
    SW1_PMR = 0;
    SW2_PMR = 0;
    SW3_PMR = 0;

    /* Unlock MPC registers to enable writing to them. */
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

    /* TXD1 is output. */
    PORT1.PMR.BIT.B6 = 0;
    MPC.P16PFS.BYTE  = 0x0A;
    PORT1.PDR.BIT.B6 = 1;
    PORT1.PMR.BIT.B6 = 1;
    /* RXD1 is input. */
    PORT1.PMR.BIT.B5 = 0;
    MPC.P15PFS.BYTE  = 0x0A;
    PORT1.PDR.BIT.B5 = 0;
    PORT1.PMR.BIT.B5 = 1;

    /* Configure the pin connected to the ADC Pot as an analog input */
#if   (BSP_CFG_BOARD_REVISION == 0)
    PORT4.PMR.BIT.B4 = 0;
    MPC.P44PFS.BYTE = 0x80;     //Set ASEL bit and clear the rest
    PORT4.PDR.BIT.B4 = 0;
#elif (BSP_CFG_BOARD_REVISION == 1)
    PORT4.PMR.BIT.B0 = 0;
    MPC.P40PFS.BYTE = 0x80;     //Set ASEL bit and clear the rest
    PORT4.PDR.BIT.B0 = 0;
#endif
```

この部分をコメントアウトまたは削除

2. *board_specific_defines*.h

使用するボードがこのファイルの名前になります（たとえば rskrx111.h）。このファイルには、スイッチや LED などに使用する端子の定義が記されており、その設定は使用するボードによって異なります。
ただし、カスタムボードを使用するときには、このファイルは不要です。以下の手順を実施してください。

1) カスタムボード用のフォルダから*board_specific_defines*.h ファイルを削除します。
2) r_bsp.h ファイルの以下の行を削除します。

```
#include    "board/rskrx111/rskrx111.h"
```

3. r_bsp.h

このヘッダファイルは platform.h に含まれており、ボードと MCU に必要なすべての#include を含んでいます。ボードに関連するインクルードパスを修正する必要があります。

1) 以下に示すように、"board/"で始まるインクルードパスを修正します。
パスを"board/*カスタムボード用のフォルダ名*/ファイル名"に変更します。

例
修正前: `#include    "board/rskrx111/rskrx111.h"`

修正後: `#include    "board/test_board/rskrx111.h"`

```
/***********************************************************************************************************************
INCLUDE APPROPRIATE MCU AND BOARD FILES
***********************************************************************************************************************/
#include    "mcu/all/r_bsp_common.h"
#include    "r_bsp_config.h"
#include    "mcu/rx111/register_access/iodefine.h"
#include    "mcu/rx111/mcu_info.h"
#include    "mcu/rx111/mcu_locks.h"
#include    "mcu/rx111/locking.h"
#include    "mcu/rx111/cpu.h"
#include    "mcu/rx111/mcu_init.h"
#include    "mcu/rx111/mcu_interrupts.h"
#include    "board/test_board/rskrx111.h"
#include    "board/test_board/hwsetup.h"
#include    "board/test_board/lowsrc.h"
#include    "board/test_board/vecttbl.h"

#endif /* BSP_BOARD_RSKRX111 */
```

この部分をカスタムボード用のフォルダ名に変更

4. r_bsp_config_reference.h

このヘッダファイルには、ボードのデフォルトオプションを提供するための設定が含まれます。このファイルに含まれていて、カスタムボードに応じて修正が必要なマクロ定義を下表に示します。必要に応じて設定を変更してください。

たとえば、コピーしたボードフォルダの設定がシステムクロックに PLL を使用しているが、ユーザシステムは HOCO を使用している場合、BSP_CFG_CLOCK_SOURCE のクロック設定を PLL から HOCO に変更してください。

また、下表にないマクロについては、その使用条件を確認し、必要に応じて修正してください。

表 7.1　カスタムボードに合わせて修正すべきマクロ

マクロ	説明
BSP_CFG_CLOCK_SOURCE	ボード上の水晶発振子とクロックソースを選択します。
BSP_CFG_XTAL_HZ	ボード上の水晶発振子に応じて値を指定します(デフォルト値: RSK 設定)。
BSP_CFG_PLL_DIV	PLL 使用時: ボード上の水晶発振子を使用して利用可能な設定値を指定します。
BSP_CFG_PLL_MUL	PLL 使用時: ボード上の水晶発振子を使用して利用可能な設定値を指定します。
BSP_CFG_ICK_DIV	ボード上の水晶発振子を使用して利用可能な設定値を指定します。
BSP_CFG_PCKB_DIV	ボード上の水晶発振子を使用して利用可能な設定値を指定します。
BSP_CFG_PCKD_DIV	ボード上の水晶発振子を使用して利用可能な設定値を指定します。
BSP_CFG_FCK_DIV	ボード上の水晶発振子を使用して利用可能な設定値を指定します。

ステップ 6.　r_bsp_config_reference.h ファイルをコピーして名前を変更する（必須）

ステップ 5 の後、r_bsp_config_reference.h ファイルをコピーし、これを r_config folder に貼り付けてから、コピーしたファイルの名前を"r_bsp_config.h"に変更します。

ステップ 7.　platform.h ファイルを修正する（必須）

このヘッダファイルは、新しく作成したカスタムボード用のフォルダ内の r_bsp.h ファイルを指定するように修正する必要があります。以下の手順に従って修正を行います。

1. コメント"/* User Board - Define your own board here. */ "の下にある行のコメントを解除します。
2. "board/"の後のフォルダ名をカスタムボード用のフォルダ名に変更します。

修正前:

```
/* User Board - Define your own board here. */
//#include "./board/user/r_bsp.h"
```

修正後:

```
/* User Board - Define your own board here. */
#include "./board/test_board/r_bsp.h"
```

テクニカルアップデートの対応について

本モジュールは以下のテクニカルアップデートの内容を反映しています。

- TN-RX*-A021A/J

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

Rev.	発行日	改訂内容	
		ページ	ポイント
2.30	2013.11.15	—	初版発行
2.40	2014.02.18	—	RX21A、RX220、RX110 のサポートを追加。「MCU 情報」項の内容を拡張
2.50	2014.07.09	—	RX64M のサポートを追加
2.60	2014.08.18	—	「カスタムボード用の BSP モジュールを作成する」のセクションを追加
2.70	2014.08.05	—	RX113 のサポートを追加
2.80	2015.01.21	—	RX71M のサポートを追加
2.81	2015.03.31	—	RX71M の動作周波数で 240MHz をデフォルトとしてサポート
2.90	2015.06.30	—	RX231 のサポートを追加
3.00	2015.09.30	—	RX23T のサポートを追加
3.01	2015.09.30	プログラム	**クロック関連** ソフトウェア不具合のため、BSP FIT モジュールを改修 ■内容 リセット直後のクロック切り替えにおいて、中速動作モードの許容周波数を超える場合、高速動作モードに切り替える処理の条件判断に誤りがあり、許容周波数を超えて中速動作モードが設定される場合があります。 ■発生条件 次の 3 つの条件に該当したとき ・BSP FIT モジュール Rev3.00 以前のバージョンで、RX231 または RX23T をご使用されている ・最も高いクロック周波数が 12MHz を超えかつ 32MHz 以下で初期定義されている（RX231) ・ICLK が 12MHz を超え、かつ 32MHz 以下で初期定義されている（RX23T） ■対策 BSP FIT モジュール Rev3.01 以降をご使用ください。 **スタック関連** ソフトウェア不具合のため、BSP FIT モジュールを改修 ■内容 BSP で定義しているスタックサイズが大きく、スタックおよびヒープ以外の RAM 領域が不足する場合があります。 ■発生条件 次の 2 つの条件に該当したとき ・BSP FIT モジュール Rev3.00 で、RX23T をご使用されている ・BSP_CFG_USER_STACK_ENABLE = 1 ■対策 BSP FIT モジュール Rev3.01 以降をご使用ください。

Rev.	発行日	改訂内容	
		ページ	ポイント
3.01	2015.09.30	プログラム	**ロック関連** ソフトウェア不具合のため、BSP FIT モジュールを改修 ■内容 ロック機能において、あらかじめ定義したハードウェア機能に準じたインデックスに過不足があり、一部ハードウェア機能に対し、ロック機能を使用できない場合があります。 ■発生条件 次の 3 つの条件に該当したとき ・BSP FIT モジュール Rev3.00 以前のバージョンで、RX231 または RX23T をご使用されている ・R_BSP_HardwareLock 関数もしくは R_BSP_HardwareUnlock 関数をご使用されている ・BSP_CFG_USER_LOCKING_ENABLED = 0 ■対策 BSP FIT モジュール Rev3.01 以降をご使用ください。 本改修により、次の定義を変更しています。 ・追加した定義（RX23T） BSP_LOCK_CMPC0, CMPC1, CMPC2 BSP_LOCK_SMCI1, SMCI5 ・追加した定義（RX231） BSP_LOCK_CMPB0, CMPB1, CMPB2, CMPB3 BSP_LOCK_LPT ・削除した定義（RX231） BSP_LOCK_CMPB BSP_LOCK_SMCI2, SMCI3, SMCI4, SMCI7, SMCI10, SMCI11
3.10	2015.12.01	— 1,6,8 64	RX130 のサポートを追加 ・誤記訂正 「動作確認デバイス」,「2.6 クロック設定」,「2.14 Trusted Memory」 ・セクション追加 「テクニカルアップデートの対応について」
3.20	2016.02.01	— 13,14 プログラム	RX24T のサポートを追加 3.2.6 クロックの設定 以下のマクロ定義を追加 ・BSP_CFG_MAIN_CLOCK_SOURCE ・BSP_CFG_MOSC_WAIT_TIME ・BSP_CFG_ROM_CACHE_ENABLE **クロック関連** イーサネットコントローラ(ETHERC)のクロック制約(ICLK=PCLKA)を満たすため、PCLKA の初期値を変更。(RX63N)

Rev.	発行日	改訂内容	
		ページ	ポイント
3.30	2016.02.29	—	RX230 のサポートを追加
		—	RX113 iodefine.h を V1.0A に更新
		42	5.15 R_BSP_SoftwareDelay 説明変更
		プログラム	**API 関数関連** BSP FIT モジュールを改修 ■内容 R_BSP_SoftwareDelay 関数でオーバヘッドの減算が必要以上にされているため、指定した時間を確保できない場合があります。 ■対策 次の定義(オーバヘッドのサイクル数)を変更しています。 OVERHEAD_CYCLES OVERHEAD_CYCLES_64 ■注意 本改修により、BSP FIT モジュール Rev3.20 以前のバージョンと比べて、R_BSP_SoftwareDelay 関数の処理時間が長くなっています。
3.31	2016.04.13	—	RX230、RX231 iodefine.h を V1.0F に更新
		—	RX23T iodefine.h を V1.1 に更新
		—	RX24T iodefine.h を V1.0A に更新
		—	RX64M iodefine.h を V1.0 に更新
		14	3.2.6 クロックの設定 以下のマクロ定義の内容を修正 ・BSP_CFG_MOSC_WAIT_TIME 以下のマクロ定義を追加 ・BSP_CFG_HOCO_WAIT_TIME ・BSP_CFG_SOSC_WAIT_TIME
		プログラム	**メモリ関連** RAM 容量追加に伴い、以下のマクロ定義の設定値を変更。(RX23T) ・BSP_RAM_SIZE_BYTES **クロック関連** 以下の内容をサポート。一部プログラムを変更。(RX23T、RX64M、RX71M) ■内容 ・システムクロックのクロックソース選択に HOCO を追加 (RX23T のみ) ・メインクロック発振器の発振源が選択可能 ・メインクロック発振器のウェイト時間が選択可能 ・サブクロック発振器のウェイト時間が選択可能(RX64M、RX71M のみ) ■注意 本変更により、RX64M、RX71M のメインクロック発振器およびサブクロック発振器のウェイト時間のデフォルト値にはユーザーズマニュアルのリセット後の値を設定しています。 Rev3.30 以前の BSP FIT モジュールのデフォルト値とは異なるので、ご注意ください。

Rev.	発行日	改訂内容	
		ページ	ポイント
3.31	2016.04.13	プログラム	**クロック関連** オプション機能選択レジスタ 1 で HOCO 発振を有効にした場合(OFS1.HOCOEN = 1)の HOCO 発振設定が適切ではないため、HOCO 発振設定を改修。(RX64M、RX71M) ■内容 ・オプション機能選択レジスタ 1 で HOCO 発振を有効にし(OFS1.HOCOEN = 1)、HOCO をシステムクロックのクロックソースに選択した場合、HOCO を停止しないように変更。 ・オプション機能選択レジスタ 1 で HOCO 発振を無効にし(OFS1.HOCOEN = 0)、HOCO をシステムクロックのクロックソースに選択しなかった場合、HOCO の電源を OFF。 **割り込み関連** ユーザーズマニュアルの IPR 設定手順に従うため、bsp_interrupt_group_enable_disable 関数内のプログラムを変更。 (RX64M、RX71M) ■内容 該当する IERm.IENj ビットが“0”のときに、IPRr レジスタを書くように変更。 **STDIO/デバッグコンソール関連** 以下の内容を改修。(RX23T、RX64M、RX71M) ■内容 BSP_CFG_USER_CHARGET_ENABLED や BSP_CFG_USER_CHARPUT_ENABLED を有効(“1”)にしても、正しく動作しなかったため、正常動作するように修正。 **API 関数関連** R_BSP_RegisterProtectEnable 関数と R_BSP_RegisterProtectDisable 関数の不要な enum の定数を削除、HOCO の enum の定数を追加。(RX23T) ■内容 R_BSP_RegisterProtectEnable 関数と R_BSP_RegisterProtectDisable 関数の引数である bsp_reg_protect_t enum の定数 「BSP_REG_PROTECT_VRCR」を削除。 「BSP_REG_PROTECT_HOCOWTCR」を追加。

Rev.	発行日	改訂内容	
		ページ	ポイント
3.40	2016.10.01	—	RX65N のサポートを追加
		15	3.2.7 ROM 上のレジスタ、および外部メモリアクセスの保護 以下のマクロ定義を追加 ・BSP_CFG_FAW_REG_VALUE ・BSP_CFG_ROMCODE_REG_VALUE
		プログラム	**クロック関連** (1) LPT モジュールにおいてコンパイルエラーの要因となるため、以下の定義のデフォルト値を変更。(RX130) ・BSP_CFG_LPT_CLOCK_SOURCE (2) ⇒ (0) (2) 以下の定義の誤りを修正。(RX230、RX231) [BSP_CFG_LPT_CLOCK_SOURCE = 1 の場合] ・BSP_LPTSRCCLK_HZ (15360) ⇒ (15000) [BSP_CFG_LPT_CLOCK_SOURCE = 2 の場合] ・定義を削除。 (3) 以下の定義を追加。(RX130) ・BSP_LPTSRCCLK_HZ

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. 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（予約領域）のアクセス禁止
   【注意】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、将来の機能拡張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。
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、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、内部ROM、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、ノイズ輻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。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/