

RX ファミリ

RX FSP への移行ガイド

要旨

本アプリケーションノートでは、ルネサス RX Flexible Software Package (RX FSP) の概要とルネサスから提供してきたソフトウェアである FIT モジュールやスマート・コンフィグレータのコード生成機能を使用したプロジェクトから、RX FSP を使用したプロジェクトへの移行方法について説明します。

対象デバイス

- ・RX ファミリ

目次

1. RX ファミリ向けのソフトウェアについて	6
2. RX FSP とは	6
2.1 FSP モジュール.....	6
2.2 FSP 開発環境と FSP スタック	7
2.3 FSP のプロジェクト作成手順.....	8
2.4 FSP プロジェクトのウィンドウ	14
2.4.1 プロジェクト・エクスプローラー	15
2.4.2 RX プロジェクトコンフィグレーションエディタビュー	15
2.4.3 Visualization ビュー.....	16
2.4.4 プロパティビュー.....	17
2.5 FSP プロジェクトにおける設定箇所	17
2.5.1 基本設定	17
2.5.2 クロック設定	19
2.5.3 端子設定	20
2.5.4 割り込み設定	21
2.5.5 イベントリンク設定	21
2.5.6 リンカセクションのマッピング設定.....	22
2.5.7 コンポーネント(モジュール)設定とスタック設定	22
2.6 FSP プロジェクトのファイル構成.....	24
2.7 FSP モジュールの API の情報の確認方法	34
3. FSP 開発環境への移行	36
3.1 スマート・コンフィグレータと FSP Configuration パースペクティブの操作の違い.....	36
3.1.1 概要とコンポーネントの状態の確認.....	37
3.1.2 ボード、デバイス選択	38
3.1.3 クロック設定	39
3.1.4 デバッグインタフェース設定.....	40
3.1.5 コンポーネント設定	41
3.1.6 端子設定	44
3.1.7 割り込み設定	45
3.1.8 イベントリンク設定	46
4. ソフトウェアの仕様比較	48
4.1 スマート・コンフィグレータによるコード生成/FIT モジュール/FSP モジュール対応機能比較 ...	48
4.1.1 周辺機能の対応機能比較.....	48
4.1.2 ミドルウェアの対応機能比較.....	51
4.2 初期設定の仕様比較	53
4.2.1 FIT モジュールと FSP モジュールの BSP の仕様比較	53
4.2.2 FSP がない場合の対応方法.....	54
4.2.2.1 CPU 関連.....	54
4.2.2.2 三角関数演算器	55
4.2.2.3 VBATT 電圧安定待ち	55
4.2.2.4 クロックソースとして使用しないクロックの停止	55
4.2.2.5 STDIO ライブラリの初期化	56

4.2.2.6	バス優先度の初期化	56
4.2.2.7	ユーザスタック使用の許可・禁止	56
4.2.2.8	ユーザモードへの遷移	56
4.2.2.9	STDIO ライブラリのクローズ	57
4.2.3	FSP BSP の初期設定	57
4.3	各モジュールの仕様比較	59
4.3.1	コンペアマッチタイマ(CMT)	60
4.3.1.1	CMT FSP モジュール	60
4.3.1.2	CMT FIT モジュールと FSP モジュールの比較	61
4.3.1.3	コード生成 CMT 用 API 関数と FSP モジュールの比較	62
4.3.2	シリアルコミュニケーションインタフェース(RSCI)	63
4.3.2.1	SCI UART FSP モジュール	63
4.3.2.2	RSCI FIT モジュールと UART FSP モジュールの比較	64
4.3.2.3	コード生成 SCI/SCIF 調歩同期式モード用 API 関数と UART FSP モジュールの比較	65
5.	FSP 以前のプロジェクトから FSP プロジェクトに移行する手順	67
5.1	FSP プロジェクトの新規作成	67
5.2	基本設定の移行	67
5.3	クロック設定の移行	71
5.4	モジュールの移行	72
5.5	割り込みの移行	76
5.5.1	モジュール追加による割り込み設定	77
5.5.2	直接追加による割り込み設定	79
5.6	端子設定の移行	80
5.7	ソースコードの移植	83
5.8	iodefine.h の追加	83
5.9	FSP モジュールの処理への置き換え	84
5.9.1	FIT モジュールの処理の FSP モジュールの処理への置き換え	84
5.9.2	コード生成の処理の FSP モジュールの処理への置き換え	85
6.	FIT アプリケーションの FSP プロジェクトへの移行例	86
6.1	CMT FIT のサンプルプログラムを使用した移行例	86
6.1.1	移行元プロジェクトのフォルダ・ファイル構成と移行対象	86
6.1.2	移行元のプロジェクトの処理概要	86
6.1.3	移行先のプロジェクトの作成	89
6.1.4	CMT モジュールの追加	89
6.1.5	移行後のプロジェクトのフォルダ・ファイル構成と変更対象	91
6.1.6	移行のポイント	92
6.1.7	変更対象のファイルの置き換え後の処理	92
6.1.8	本移行例における FIT モジュールの処理の FSP モジュールへの処理の置き換え	95
6.1.8.1	メイン処理の比較	95
6.1.8.2	コールバック関数の比較	95
6.2	RSCI FIT のサンプルプログラムを使用した移行例	96
6.2.1	移行元プロジェクトのフォルダ・ファイル構成と移行対象	96
6.2.2	移行元のプロジェクトの処理概要	96
6.2.3	移行先のプロジェクトの作成	102
6.2.4	SCI モジュールの追加	102

6.2.5	PORT 設定の追加.....	108
6.2.6	移行後のプロジェクトのフォルダ・ファイル構成と変更対象.....	110
6.2.7	移行のポイント	110
6.2.8	変更対象のファイルの置き換え後の処理	110
6.2.9	本移行例における FIT モジュールの処理の FSP モジュールへの処理の置き換え	115
6.2.9.1	メイン処理の比較.....	115
6.2.9.2	コールバック関数の比較.....	116
7.	コード生成アプリケーションの FSP プロジェクトへの移行例	117
7.1	CMT コード生成のサンプルプログラムを使用した移行例.....	117
7.1.1	移行元プロジェクトのフォルダ・ファイル構成と移行対象	117
7.1.2	移行元のプロジェクトの処理概要	117
7.1.3	移行先のプロジェクトの作成.....	122
7.1.4	CMT モジュールの追加	122
7.1.5	ICU モジュールの追加.....	124
7.1.6	PORT 設定の追加.....	129
7.1.7	移行後のプロジェクトのフォルダ・ファイル構成と変更対象.....	134
7.1.8	移行のポイント	134
7.1.9	変更対象のファイルの置き換え後の処理	135
7.1.10	本移行例におけるコード生成の処理の FSP モジュールへの処理の置き換え.....	137
7.1.10.1	モジュールの初期化処理の比較.....	137
7.1.10.2	メイン処理の比較.....	138
7.1.10.3	コールバック関数の比較.....	138
7.2	SCI コード生成のサンプルプログラムを使用した移行例	139
7.2.1	移行元プロジェクトのフォルダ・ファイル構成と移行対象	139
7.2.2	移行元のプロジェクトの処理概要	139
7.2.3	移行先のプロジェクトの作成.....	143
7.2.4	SCI モジュールの追加.....	144
7.2.5	PORT 設定の追加.....	149
7.2.6	移行後のプロジェクトのフォルダ・ファイル構成と変更対象.....	151
7.2.7	移行のポイント	152
7.2.8	変更対象のファイルの置き換え後の処理	152
7.2.9	本移行例におけるコード生成の処理の FSP モジュールへの処理の置き換え.....	154
7.2.9.1	モジュールの初期化処理の比較.....	155
7.2.9.2	メイン処理の比較.....	155
7.2.9.3	コールバック関数の比較.....	156
8.	スマート・コンフィグレータを使用しないプロジェクトの FSP プロジェクトへの移行例	157
8.1	割り込み関数を含まないサンプルプログラムを使用した移行例	157
8.1.1	移行元プロジェクトのフォルダ・ファイル構成と移行対象	157
8.1.2	移行元のプロジェクトの処理概要	157
8.1.3	移行先のプロジェクトの作成.....	160
8.1.4	移行後のプロジェクトのフォルダ・ファイル構成と変更対象.....	160
8.1.5	移行のポイント	161
8.1.6	変更対象のファイルの置き換え後の処理	161
8.2	割り込み関数を含むサンプルプログラムを使用した移行例.....	165
8.2.1	移行元プロジェクトのフォルダ・ファイル構成と移行対象	165

8.2.2	移行元のプロジェクトの処理概要	165
8.2.3	移行先のプロジェクトの作成.....	170
8.2.4	割り込み設定の追加	170
8.2.5	移行後のプロジェクトのフォルダ・ファイル構成と変更対象.....	173
8.2.6	移行のポイント	173
8.2.7	変更対象のファイルの置き換え後の処理	173
9.	FSP プロジェクトへの移行時の注意点	178
9.1	BSP の組み込みと競合.....	178
9.2	IO レジスタ定義ファイルの違いと競合	178
9.3	同じ周辺機能が搭載されていないデバイスでの移行	182
10.	参考ドキュメント	183
	改訂記録	184

1. RX ファミリ向けのソフトウェアについて

ルネサスの RX ファミリでは、お客様のシステム開発にご利用可能なソフトウェアパッケージとして、「Code generator」と「FIT ソフトウェアモジュール」の二種類を提供していました。「Code generator」は [RX スマート・コンフィグレータ](#) によるドライバコード自動生成、FIT ソフトウェアモジュールは [Firmware Integration Technology](#)（以下 FIT）の仕様に基づくモジュール群で [RX Driver Package](#) としてパッケージングしています。2026 年 1 月以降にリリースされる RX ファミリ新デバイス向けには、より一貫性と競争力のあるエクスペリエンスをお客様に提供するために [RX Flexible Software Package](#) を提供していきますので、FSP をご活用ください。

2. RX FSP とは

Renesas RX Flexible Software Package(以下 FSP)は、ルネサスの RX ファミリの MCU を用いた組み込みシステム設計向けに、ユーザフレンドリーでスケーラブル、かつ高品質なソフトウェアを提供するために設計されたソフトウェアパッケージです。FSP には、ボードサポート パッケージ(BSP)、周辺機能にアクセスするためのハードウェア抽象化レイヤー(HAL)ドライバ、マルチタスクをサポートする FreeRTOS、基本的なミドルウェアスタックなど、様々なソフトウェアコンポーネントが含まれています。

- ✓ FSP は、重要なデータやプログラムの漏洩を防ぐ新しい Protect Zone 機能および革新的なセキュリティ機能をサポートしており、HAL ドライバ、FreeRTOS およびその他の多様なミドルウェアスタックを用いて、セキュアな IoT デバイスを迅速かつ多彩な方法で開発できます。
- ✓ マルチスレッドアプリケーション環境をサポートする FreeRTOS をベースとして使用しますが、ベアメタルでの使用、お好みの RTOS の使用、既存のソフトウェア資産、サードパーティのエコシステムソリューションの使用など、FSP はユーザの開発に柔軟性をもたらします。
- ✓ 統合開発環境 e2 studio は、FSP を使用する際の直感的なコンフィグレータおよびインテリジェントなコード生成をサポートします。

FreeRTOS は Amazon Web Services, Inc. の商標です。

<http://www.openrtos.net/copyright.html>

2.1 FSP モジュール

FSP は、ボードサポート パッケージ(BSP)、周辺機能モジュールおよび、ミドルウェアモジュールで構成されます。モジュールは、FSP の中核的な構成要素です。最も単純な FSP アプリケーションは、BSP とユーザアプリケーションを上配置した 1 つのモジュールで構成されています。なお、FSP の BSP は FIT とは異なる BSP 構造になっています。

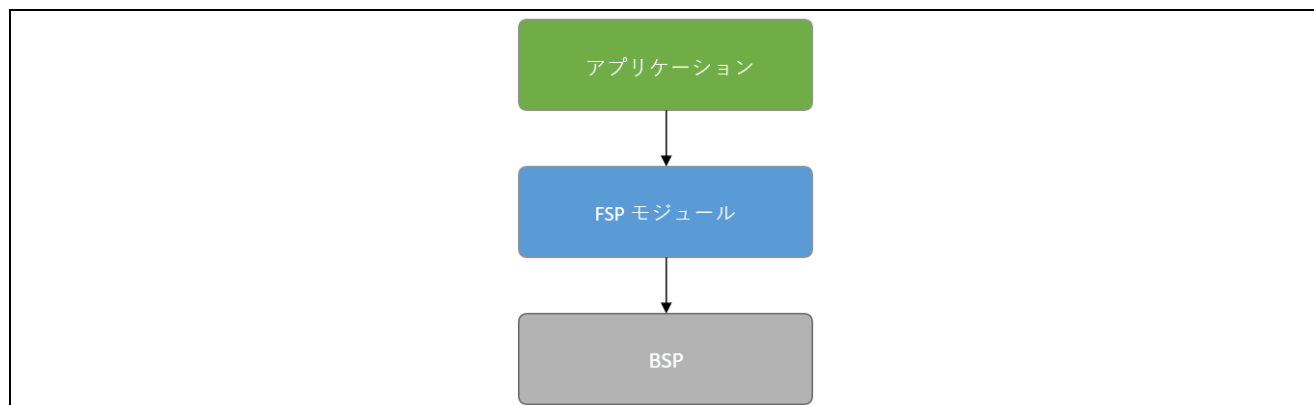


図 2-1 FSP アプリケーションの最小構成

モジュールはさまざまなことを行うことができますが、すべてのモジュールは、上位層のソフトウェアに機能を提供し、下位層のソフトウェアまたは周辺機能のハードウェアに対して要求を実行するという基本的な概念を共有しています。

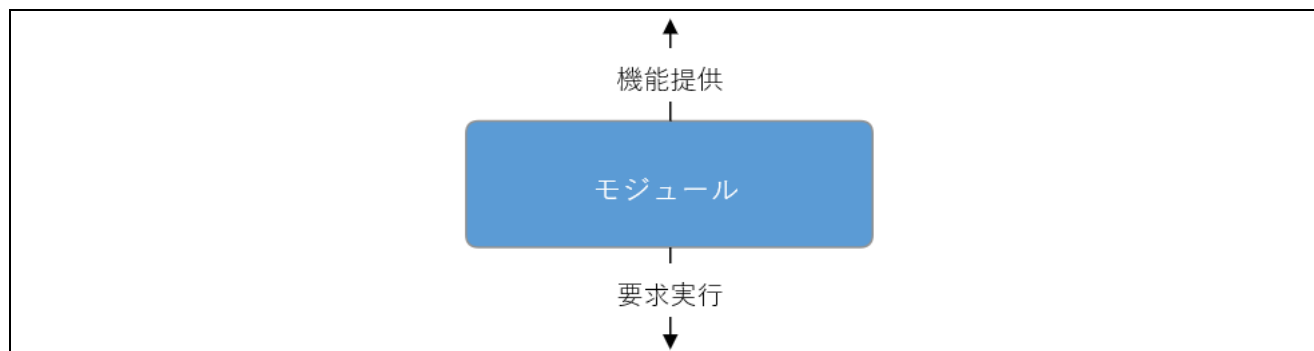


図 2-2 FSP モジュール

モジュールによって提供される機能の量は、機能的なユースケースに基づいて決定されます。

BSP は FSP モジュールの基盤であり、使用されている MCU を決定する機能や、クロック、割り込み、ピンを構成する機能を提供します。

FSP モジュールはルネサスが提供してきた既存のソフトウェアである FIT モジュール、RX スマート・コンフィグレータでコード生成されたソースコードとは機能に互換性がありません。それぞれの API も仕様が異なります。また、FSP を使用した C/C++ プロジェクト(以下 FSP プロジェクト)で使用するレジスタ定義ファイルは、FIT モジュールおよび RX スマート・コンフィグレータのコード生成を使ったプロジェクトで使用するレジスタ定義ファイル (iodef.h) と構成が異なります。FSP ではレジスタ定義ファイルを周辺機能ごとに分割した複数のファイルで構成される「renesas.h」として提供します。各モジュールの仕様の差異については 4 ソフトウェアの仕様比較をご参照ください。

2.2 FSP 開発環境と FSP スタック

FSP は、ルネサスの RX ファミリの MCU デバイスを対象とするプロジェクトを開発するための効率向上ツールを多数提供しています。IDE の e² studio は、プロジェクトの作成、モジュールの選択と設定、コード開発、コード生成、デバッグなどの主要なステップをすべて管理できる、使い慣れた開発環境を提供します。

FSP の一つの特長として FSP スタックがあります。FSP スタックとは、ある機能またはユースケースを実現するために必要なモジュールをグループ化したものです。FSP では、モジュールを組み合わせると FSP スタックが形成されます。FSP スタックは選択したモジュールが必要するものと別モジュールが提供するものを一致させることによって形成されます。たとえば、SPI 通信のユースケースでは割り込みを使用する方法と DTC/DMAC を使用する方法がありますが、DTC を使用する場合、SPI モジュールには、データを送受信するための転送インタフェースを提供するモジュールが必要です。転送インタフェースの要件は、DTC ドライバモジュールで満たすことができます。SPI 通信の他に UART 通信や SD/MMC 通信でも DTC を使用する場合、FSP スタックを形成することにより複数のモジュールで DTC モジュールを同時に共有できます。

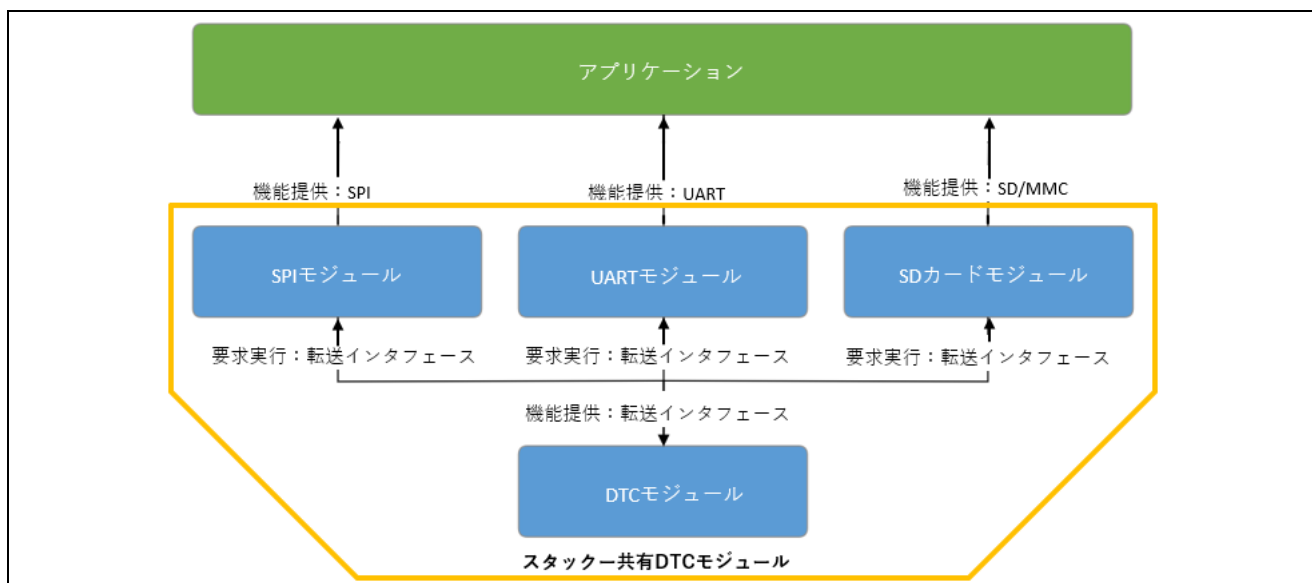


図 2-3 FSP スタックの概要

モジュールをスタックする機能により、アーキテクチャ全体の柔軟性が確保されます。複数のモジュールに同じ機能が含まれている場合、アプリケーション機能が異なるユーザ設計間で動作する必要があるときに問題が発生します。モジュールを再利用可能にするには、依存モジュールを同じ機能を提供する他のモジュールと交換する必要があります。FSP アーキテクチャは、FSP インタフェースを使用してモジュールをスワップインおよびスワップアウトすることにより、この柔軟性を提供します。

2.3 FSP のプロジェクト作成手順

FSP を使用した C/C++ プロジェクトの作成手順を以下に説明します。

(1) e² studio の起動

e² studio を起動し、ワークスペースを指定します。起動後、メニューの[ファイル] → [新規] → [Renesas C/C++ Project] → [Renesas RX]の順に選択してプロジェクト作成ウィザードを開きます。

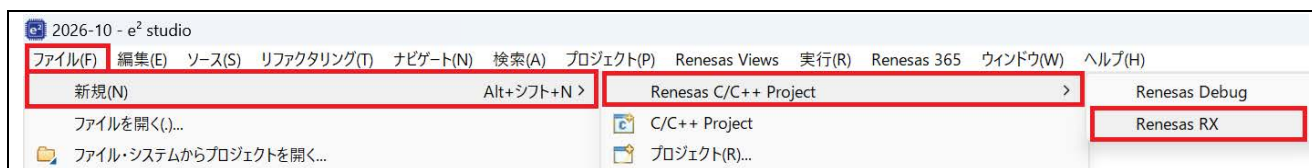


図 2-4 新規プロジェクトの作成

(2) FSP プロジェクトの選択

プロジェクト作成ウィザードで、[Renesas RX C/C++ FSP Project]を選択し、[次へ]ボタンをクリックします。

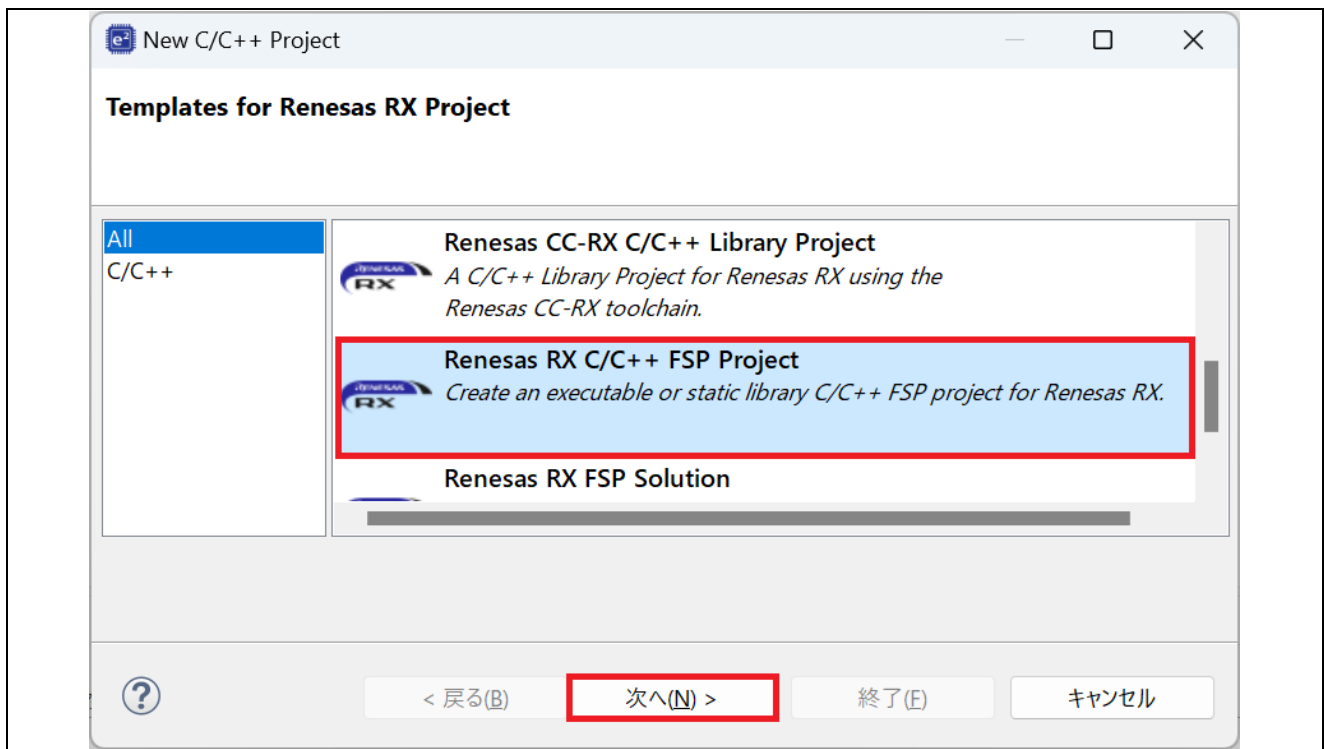


図 2-5 プロジェクトテンプレートの選択

(3) プロジェクト名入力

プロジェクト名を入力し、[次へ]ボタンをクリックして次に進みます。

(例 : RX C/C++ FSP Project、プロジェクト名 : "FSP_example")



図 2-6 Renesas RX C/C++ Project の作成

(4) デバイス・ツールチェーン・デバッグ選択

[Device Selection]で[FSP Version]、[Board]、[Device]、[Language]を選択します。

(デバイスはボードを選択するとボードに搭載されたデバイスが選択されます。)

[IDE Project Type]でプロジェクトのタイプを選択します。[Toolchains]で使用するツールチェーンとバージョンを選択します。[Debugger]で使用するデバッガを選択します。[次へ]ボタンをクリックします。

Renesas RX C/C++ FSP Project

Renesas RX C/C++ FSP Project

Device and Tools Selection

Device Selection

FSP Version: 1.0.0

Board: EK-RX74M

Device: R5K574MCBxAE

Core: RXv3e

Language: ☒ C ☐ C++

Board Description

Evaluation kit for RX74M MCU Group

Visit <https://renesas.com/ek-rx74m> to get kit user's manual, quick start guide, errata, design package, example projects, etc.

Device Details

Protect Zone	No
Pins	320
Processor	RXv3e

IDE Project Type

e² studio managed build

Debugger

E2 Lite (RX)

Toolchains

GCC for Renesas RX

Renesas CC-RX

v3.08.00 [Manage Toolchains...](#)

< 戻る(B) **次へ(N) >** 終了(F) キャンセル

図 2-7 デバイス、IDE プロジェクトタイプ、ツールチェーン、デバッグ設定の選択

(5) プロジェクトタイプ選択

[Project Type Selection]でプロジェクトタイプを選択します。

ここでは「Flat (Non-Protect Zone) Project」のまま [次へ]ボタンをクリックします。

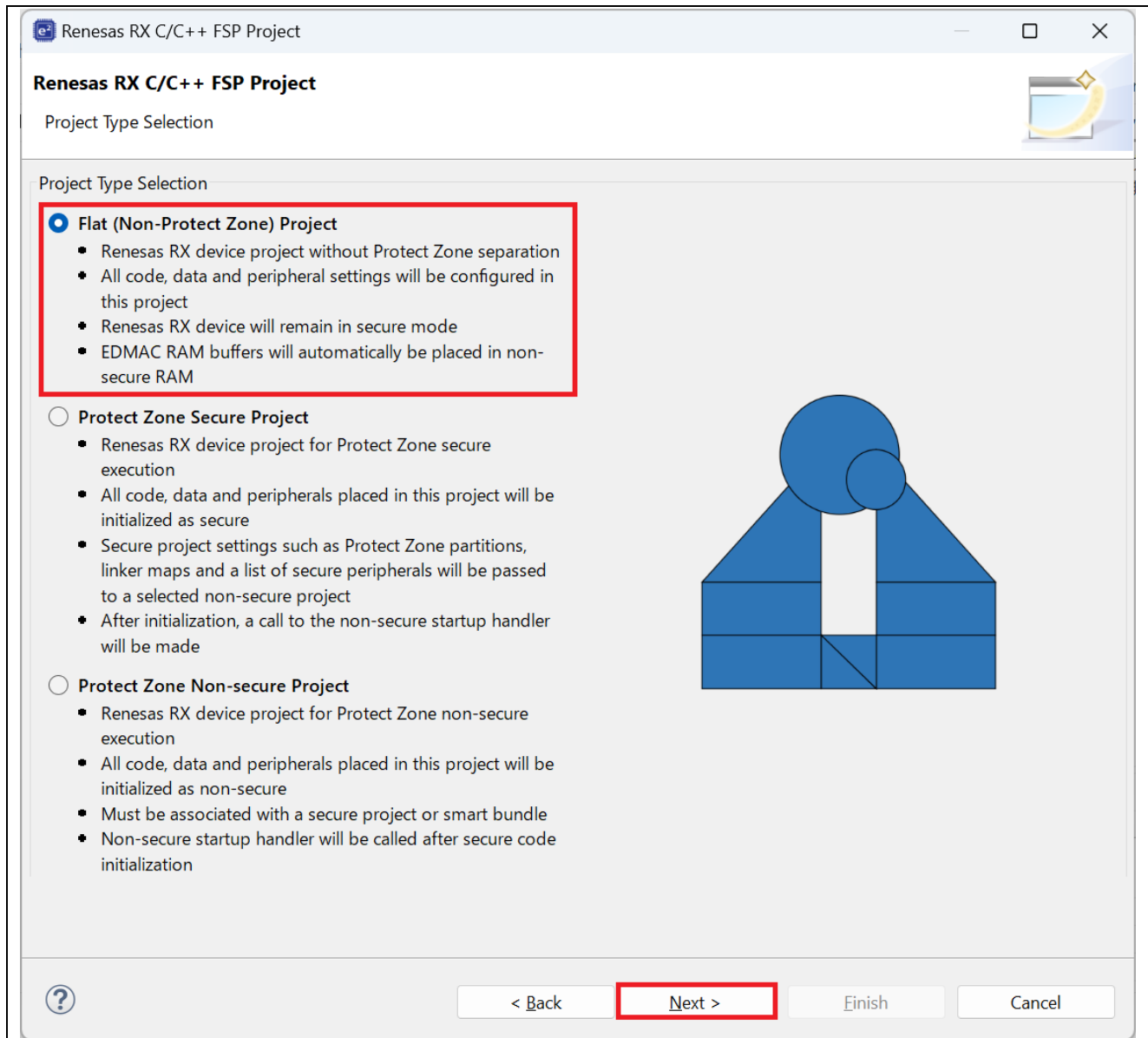


図 2-8 Project Type の選択

(6) Preceding Project または Smart Bundle Selection の選択

「None」のまま、[次へ]ボタンをクリックします。

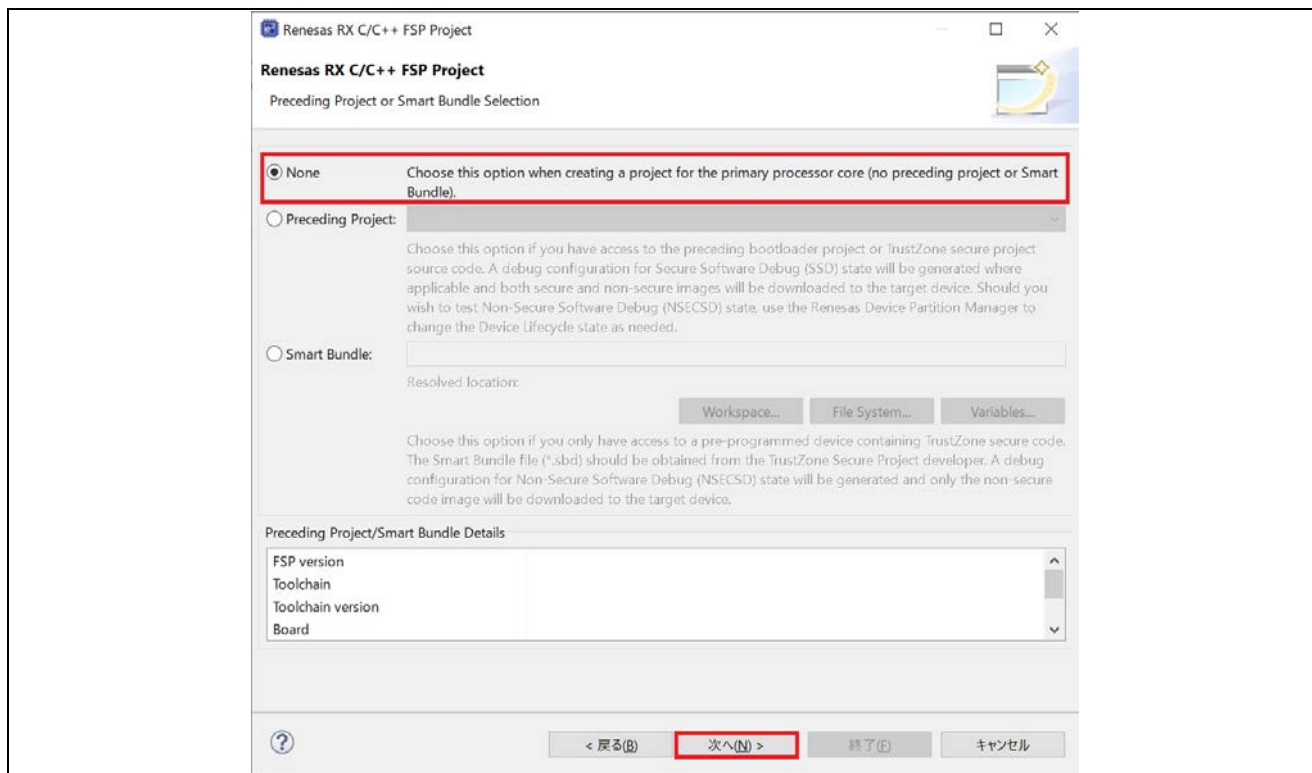


図 2-9 Preceding Project または Smart Bundle Selection の選択

(7) プロジェクトの種類・RTOS の選択

[Build Artifact Selection]で作成するプロジェクトの種類を選択します。

[RTOS Selection]で RTOS を使用するかどうかなを選択します。ここでは「No RTOS」のまま

[次へ]ボタンをクリックします。

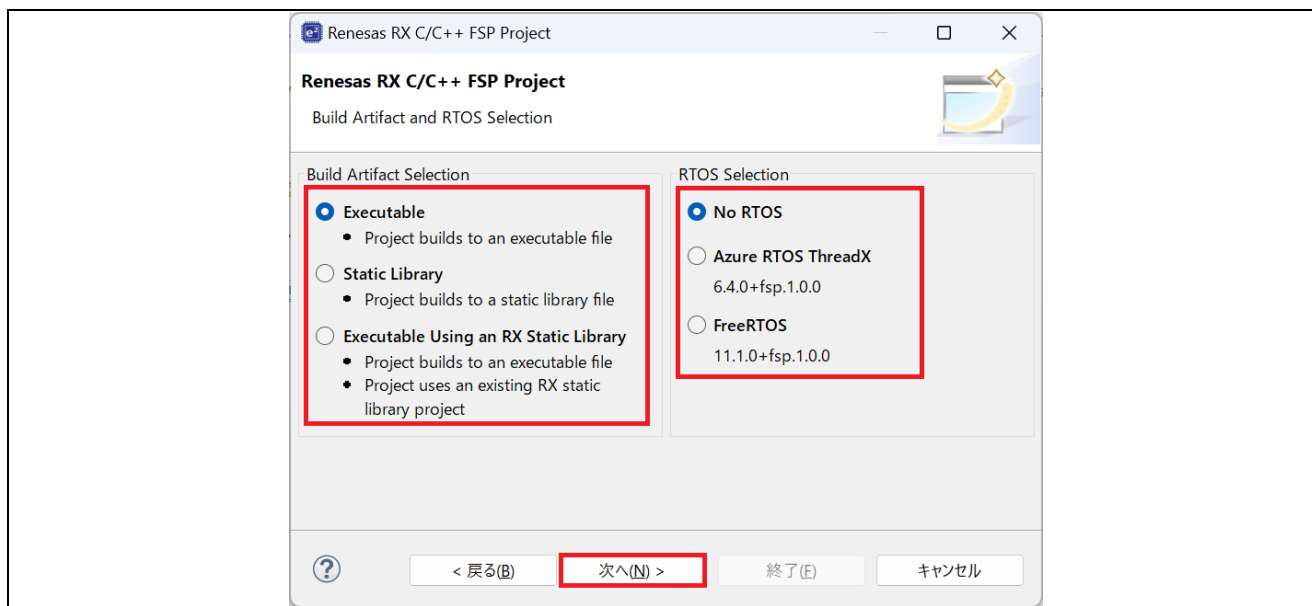


図 2-10 プロジェクトの種類、RTOS の選択

(8) プロジェクトテンプレートの選択

[Project Templates Selection]でテンプレートを選択します。[終了] ボタンをクリックします。

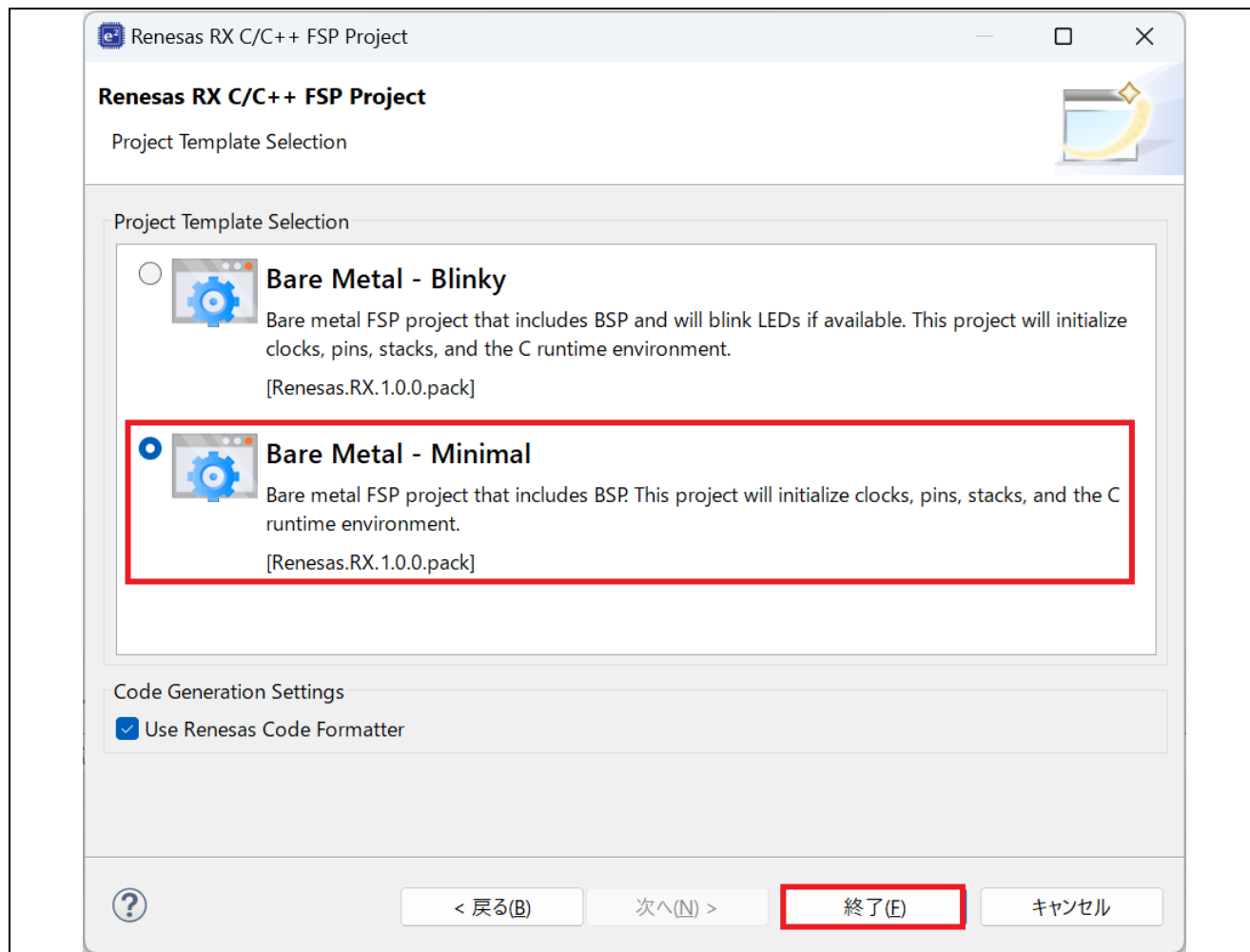


図 2-11 プロジェクトテンプレートの選択

(9) FSP Configuration パースペクティブの選択

FSP Configuration のパースペクティブを開くかどうかを選択します。ここでは[パースペクティブを開く]ボタンをクリックします。

(Eclipse で「パースペクティブ」とは、あらかじめ複数のビューを組み合わせるレイアウトを定義したものを指します。)

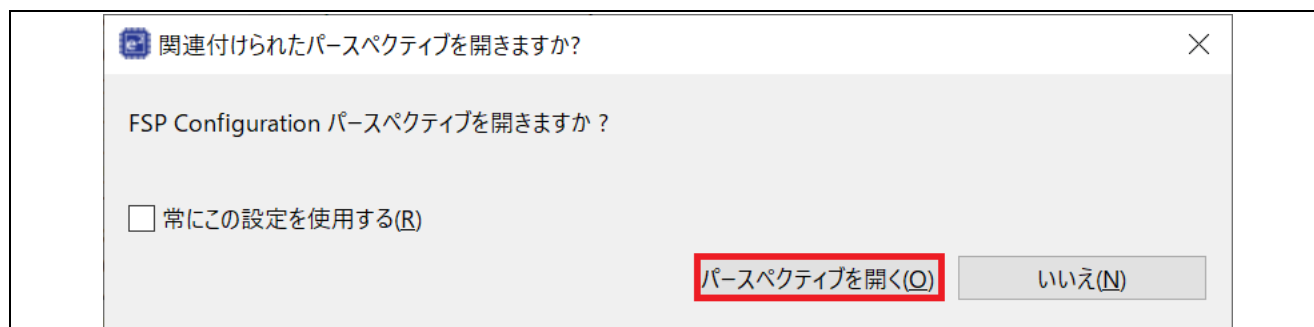


図 2-12 パースペクティブを開くかどうかの選択

(10) 新規プロジェクト作成完了

新規 C/C++ プロジェクトの作成が成功すると、作成したプロジェクトが FSP Configuration のパースペクティブ上で開きます。

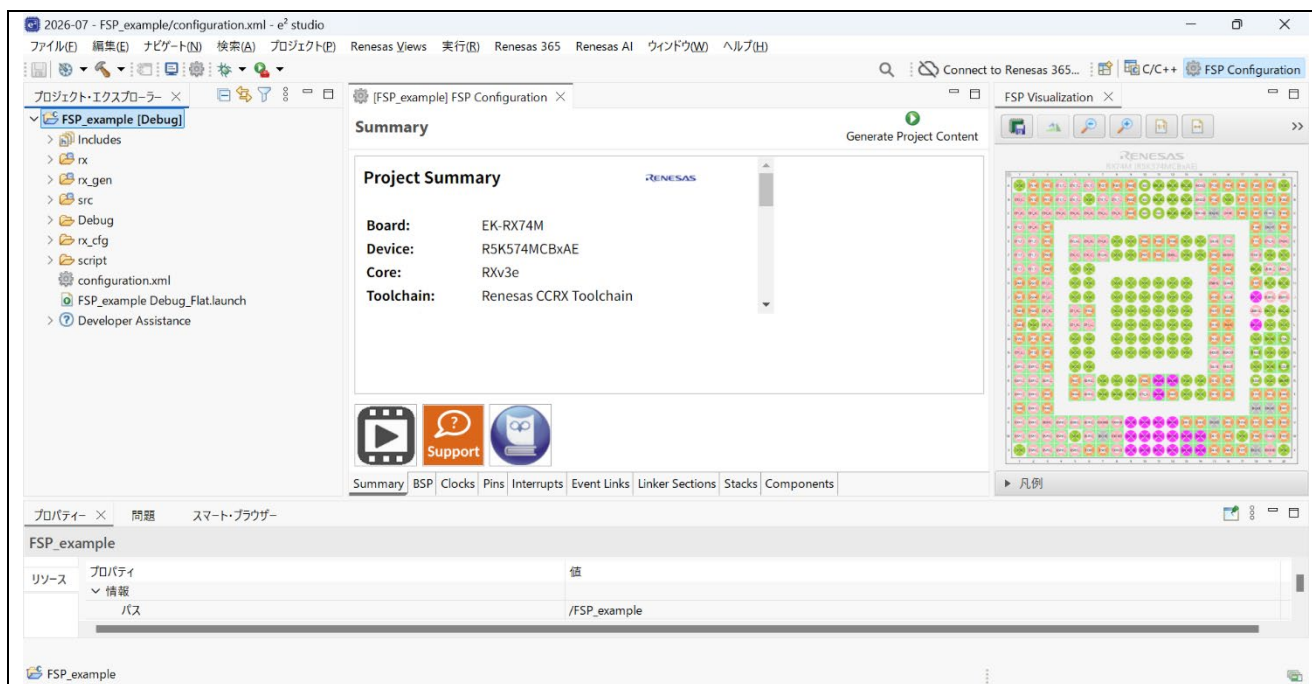


図 2-13 [FSP Configuration]パースペクティブ

2.4 FSP プロジェクトのウィンドウ

[FSP Configuration]パースペクティブの構成を下图に示します。

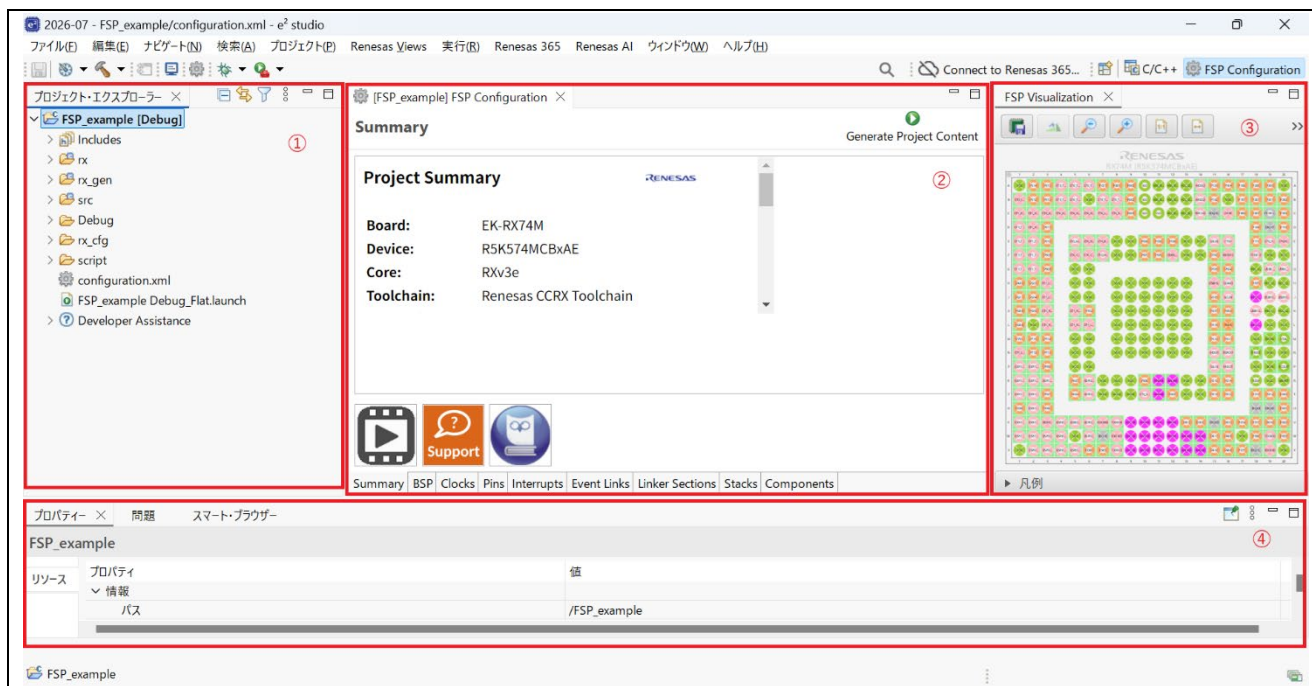


図 2-14 [FSP Configuration]パースペクティブの構成

- ① プロジェクト・エクスプローラー
- ② RX プロジェクトコンフィグレーションエディタビュー
- ③ Visualization ビュー
- ④ プロパティビュー

2.4.1 プロジェクト・エクスプローラー

プロジェクトのフォルダ・ファイル構成をツリーで表示します。

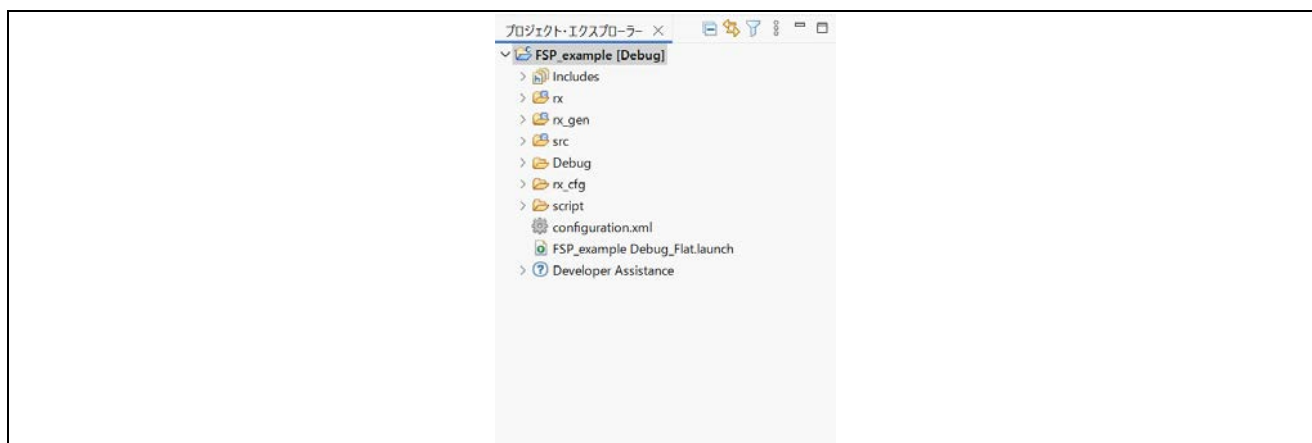


図 2-15 プロジェクト・エクスプローラー

ビューが開いていない場合は、e² studio メニュー上の[ウィンドウ] → [ビューの表示] → [プロジェクト・エクスプローラー]を選択してください。

2.4.2 RX プロジェクトコンフィグレーションエディタビュー

概要 (Summary) ページ、BSP 構成 (Board Support Package Configuration) ページ (以降 BSP ページと呼称します。)、クロック構成 (Clocks Configuration) ページ、端子構成 (Pin Configuration) ページ、割り込み構成 (Interrupts Configuration) ページ、イベントリンク構成 (Event Links Configuration) ページ、リンカセクションのマッピング構成 (Linker Section Mapping Configuration) ページ、スタック構成 (Stacks Configuration) ページ、コンポーネント構成 (Components Configuration) ページの 9 ページから構成されます。タブをクリックして、ページを選択すると選択したタブに応じて内容が切り替わります。

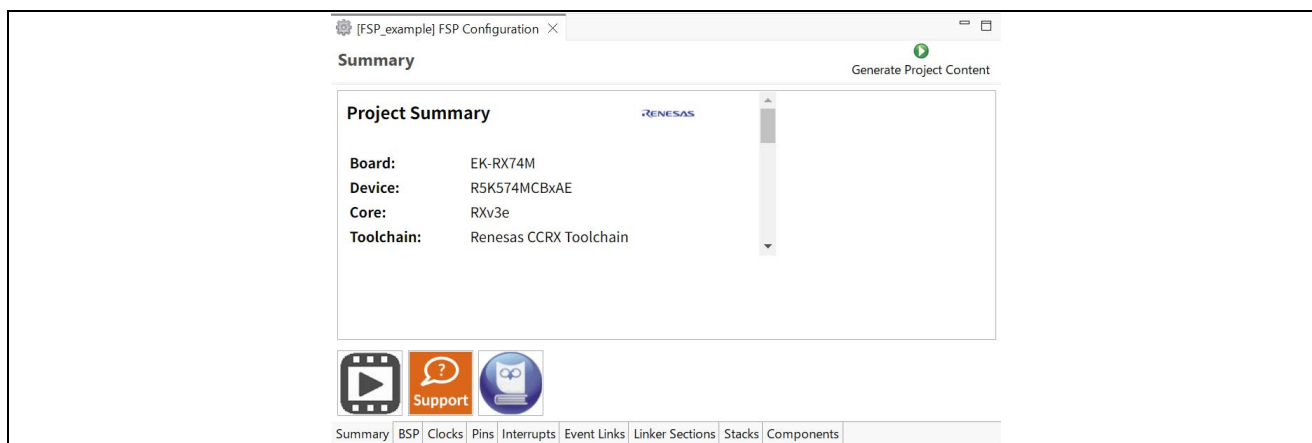


図 2-16 RX プロジェクトコンフィグレーションエディタビュー

ビューが開いていない場合は、[プロジェクト・エクスプローラー] からコンフィギュレーション・ファイル (configuration.xml) を右クリックし、コンテキストメニューから [開く] を選択してください。

2.4.3 Visualization ビュー

MCU/MPU パッケージ図上に端子状態を表示します。端子設定を変更することもできます。

画面右上の[>>]ボタンで初期状態では[Pin Function]と[Module Name]となっていますが、リストから選択を切り替えることにより表示する端子情報の切り替えが行えます。



図 2-17 Visualization ビュー

ビューが開いていない場合は、e² studio メニュー上の[Renesas Views] → [C/C++] → [FSP Visualization]を選択してください。

2.4.4 プロパティビュー

プロパティビューを下図に示します。BSP ページや[Stacks Configuration]ページでモジュールを選択した際にプロパティ情報を持つ場合、詳細が表示されます。プロパティから設定を変更することが可能です。

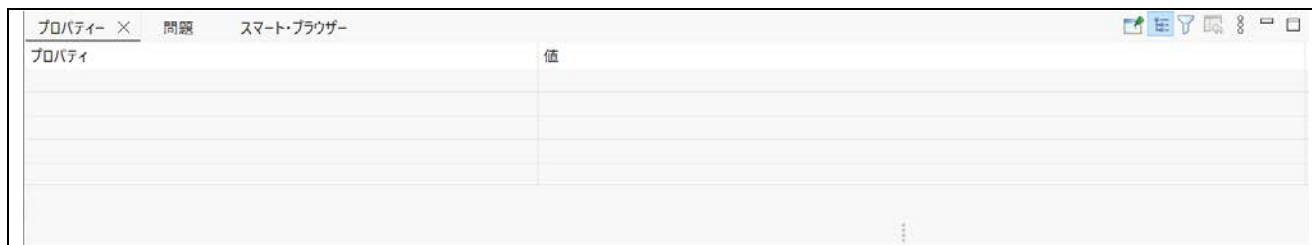


図 2-18 プロパティビュー（初期状態）

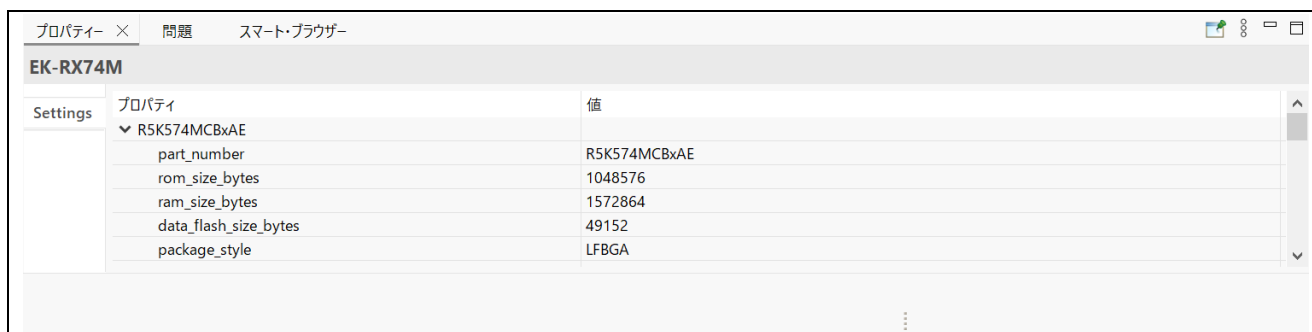


図 2-19 プロパティビュー（BSP ページ表示時）

ビューが開いていない場合は、e² studio メニュー上の[ウィンドウ] → [ビューの表示] → [プロパティ]を選択してください。

2.5 FSP プロジェクトにおける設定箇所

FSP プロジェクトにおける設定のほとんどは RX プロジェクトコンフィグレーションエディタビューとプロパティビューで行います。

2.5.1 基本設定

プロジェクト作成後も BSP ページで FSP バージョンやボード、デバイスの変更が可能です。

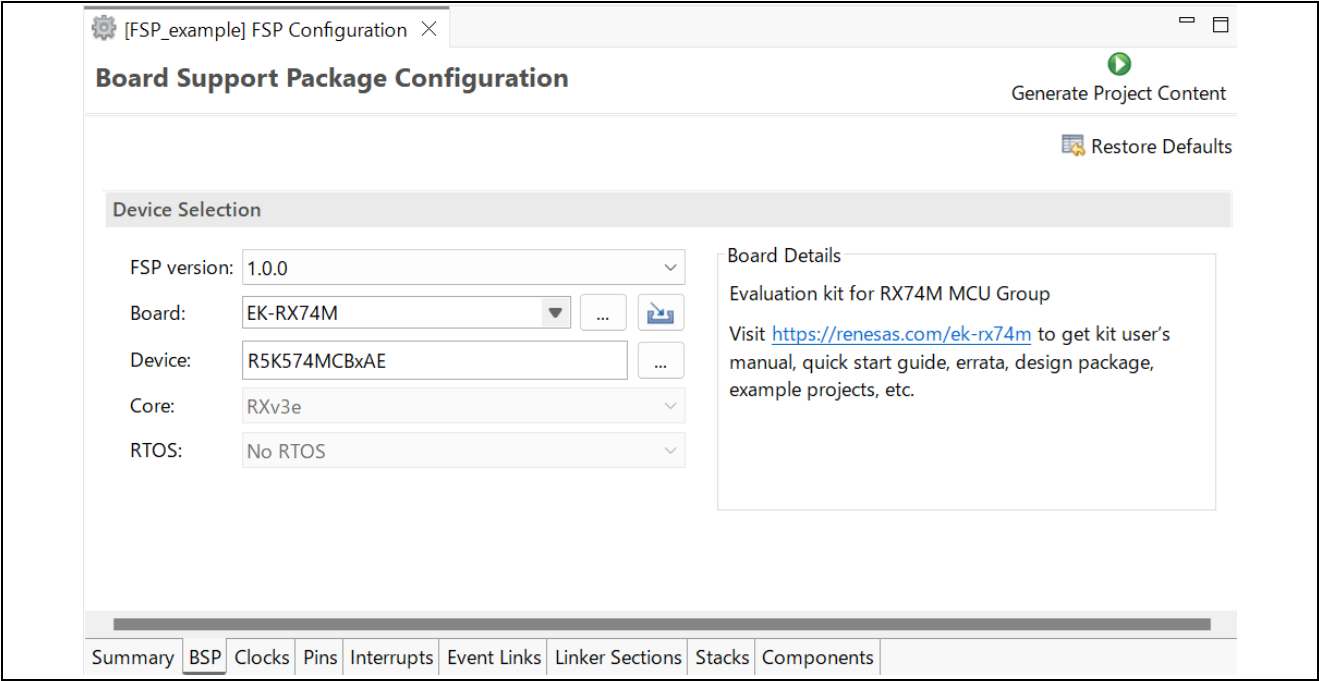


図 2-20 [BSP]ページ

BSP ページ選択時のプロパティビューでは Security や Clocks 等のデバイス毎の設定項目と RX Common の基本的な設定項目があり、必要に応じて設定が必要です。

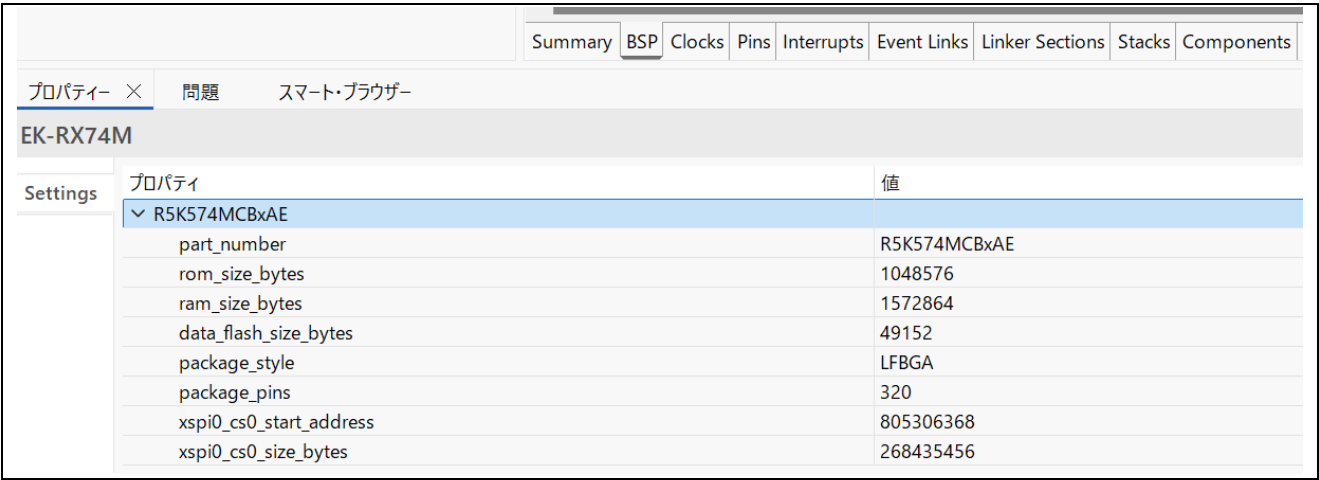


図 2-21 BSP のプロパティ

RX Common の設定項目について下表に示す。

表 2.1 基本設定項目

No	設定項目表示	設定項目の概要	初期値
1	Main stack size (bytes)	システムで使用するスタックサイズ	0x400
2	Heap size (bytes)	システムで使用するヒープサイズ	0
3	Protect zone root stack size (bytes)	プロテクトゾーンの Root ワールドで使用するスタックサイズ	0x800
4	MCU Vcc (mV)	MCU に供給される電圧	3300
5	Parameter checking	API のパラメータチェックを有効にするかどうかの設定	Disabled (無効)
6	Assert Failures	FSP_ASSERT()マクロの動作の設定	Return FSP_ERR_ASSERTION (FSP_ERR_ASSERTION を返す)
7	Clock Registers not Reset Values during Startup	起動時にクロック関連レジスタをリセット値にするかどうかの設定	Disabled (無効)
8	Main Oscillator Populated	メインクロック発振器を発振させるかどうかの設定	Populated (発振)
9	PFS Protect	PFS レジスタへのロックを行うかどうかの設定	Enabled (ロックを保持する)
10	C Runtime Initialization	C ランタイムの初期化を行うかどうかの設定	Enabled (初期化を行う)
11	Early BSP Initialization	C ランタイムの初期化の前に BSP の関数を使うための初期化を行うかどうかの設定	Disabled (初期化を行わない)
12	Main Oscillator Clock Source	メインクロック発振器の発振源の設定	Crystal or Resonator
13	Subclock Populated	サブクロック発振器を発振させるかどうかの設定	Not Populated (発振停止)
14	Subclock Drive (Drive capacitance availability varies by MCU)	サブクロック発振器のドライブ能力の設定	Standard/Normal mode
15	Subclock Stabilization Time (ms)	サブクロック発振器の発振安定待ち時間の設定	1000

例えば、システムで使用するスタックサイズを 256 バイト増やしたい時は Main stack size (bytes)を 0x500 に変更することによってシステムで使用するスタックサイズを増やすことができます。

なお、基本設定の項目は選択されたデバイスや FSP バージョンにより変わることがあります。

2.5.2 クロック設定

[Clocks Configuration]ページではクロックについての設定項目があり、ご使用の環境に合わせて設定が必要です。

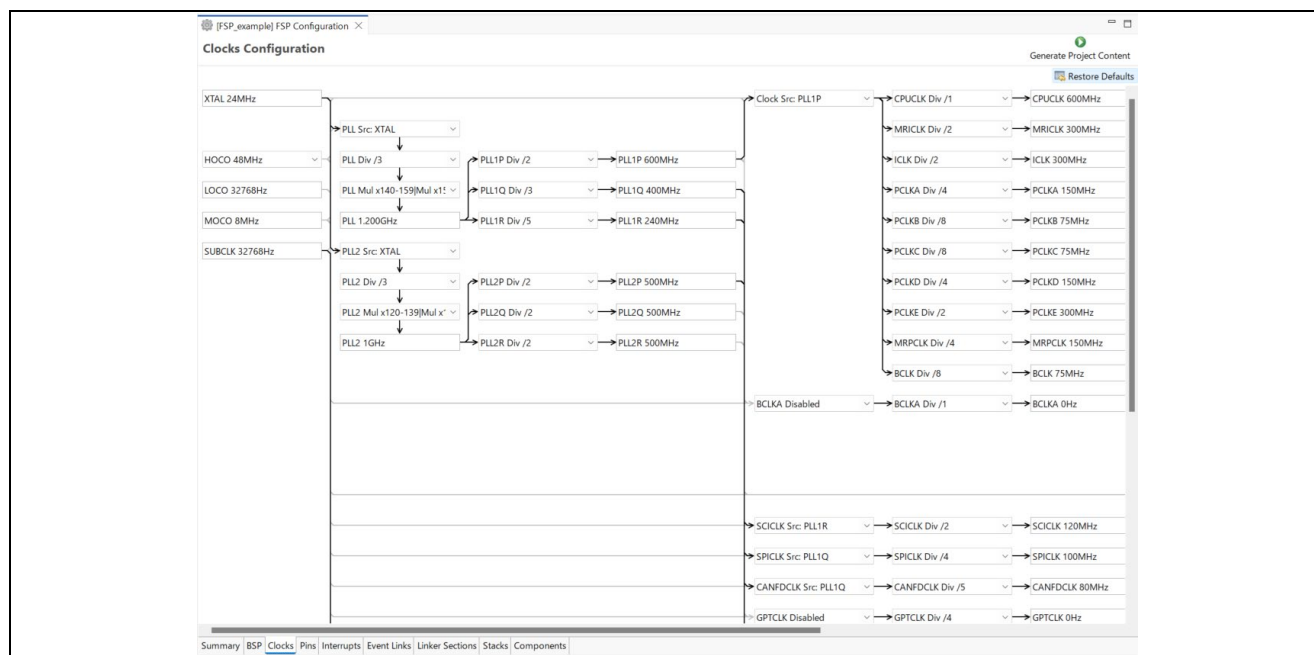


図 2-22 [Clocks Configuration]ページの一例

例えば、システムクロック (ICLK) の分周比を 4 分周に変更したい時は ICLK Div /2 を ICLK Div /4 に変更することによってシステムクロック (ICLK) の分周比を 4 分周にすることができます。

なお、クロック設定の項目は選択されたデバイスや FSP バージョンにより変わることがあります。

2.5.3 端子設定

[Pin Configuration] ページでは端子機能の割り当てについての設定項目があり、必要に応じて設定が必要です。端子機能の割り当ては[Visualization]ビューからも設定可能です。

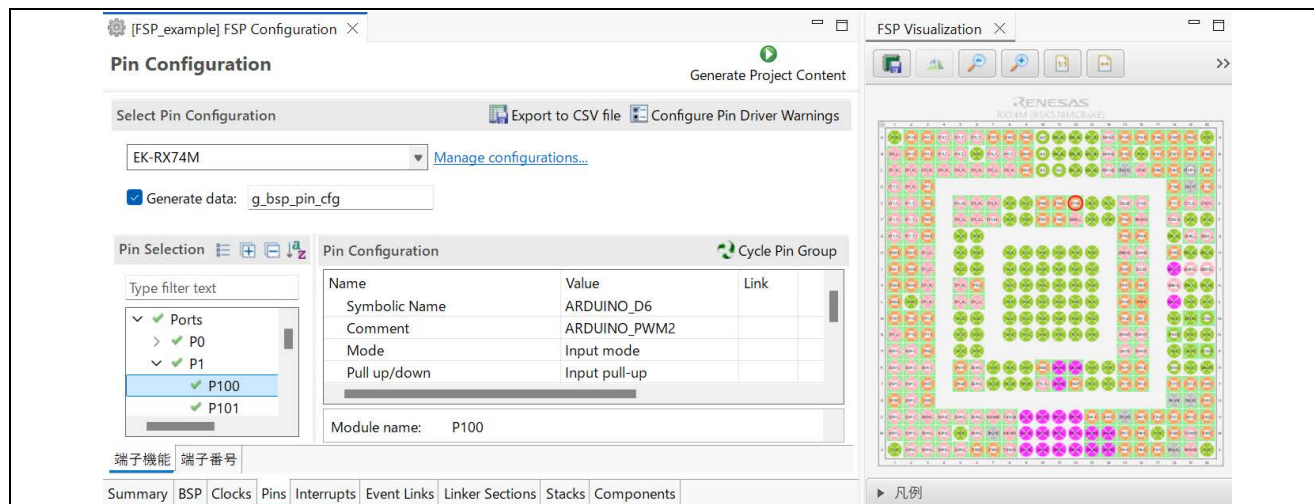


図 2-23 [Pin Configuration]ページと[Visualization]ビュー

例えば、P300 を入力ポートに変更したい時は Mode を Input mode に変更することによって入力ポートに設定することができます。

なお、ポート設定の項目は選択されたデバイスや FSP バージョンにより変わることがあります。また、初期値はポートの端子機能の設定により変わることがあります。

2.5.4 割り込み設定

[Interrupts Configuration]ページでは割り込みの設定が行えます。割り込みを使用する場合は設定が必要です。後述のコンポーネント設定でコンポーネントを追加した際に割り込みを使用する設定に変更すると追加したモジュールが使用する割り込み設定が自動的に反映されます。

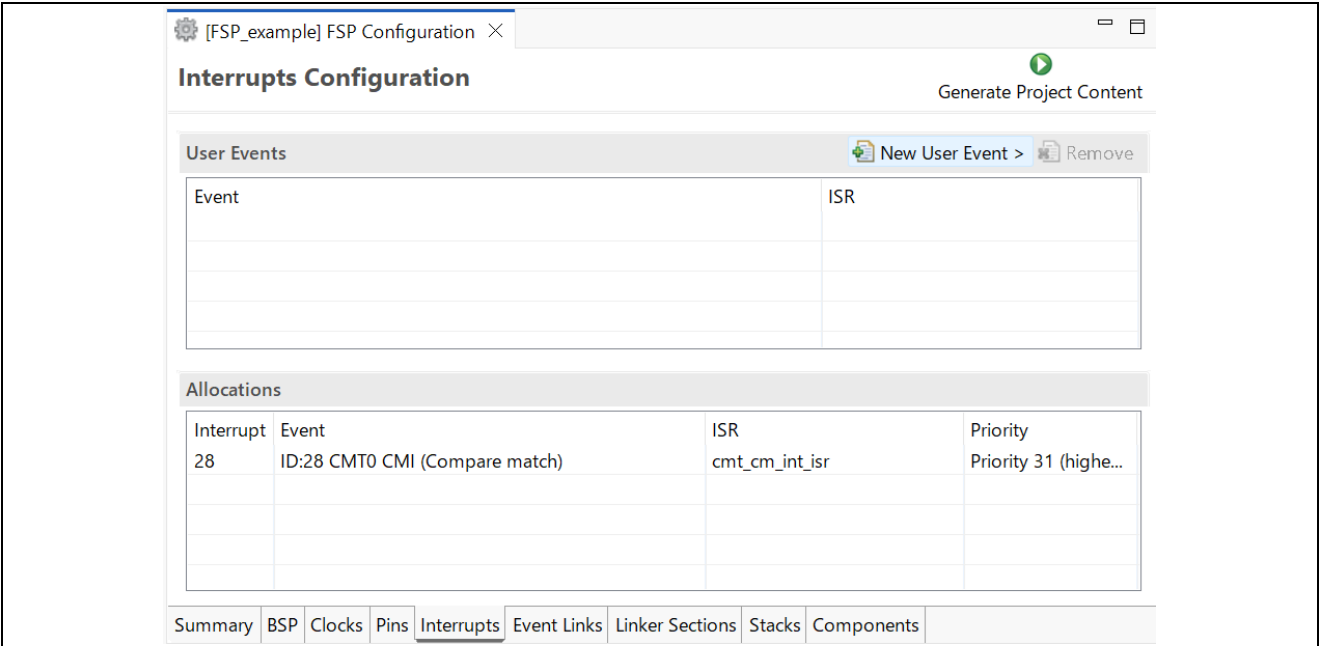


図 2-24 [Interrupts Configuration]ページ

2.5.5 イベントリンク設定

[Event Links Configuration]ページではイベントリンクの設定が行えます。イベントリンク機能を使用する場合は設定が必要です。

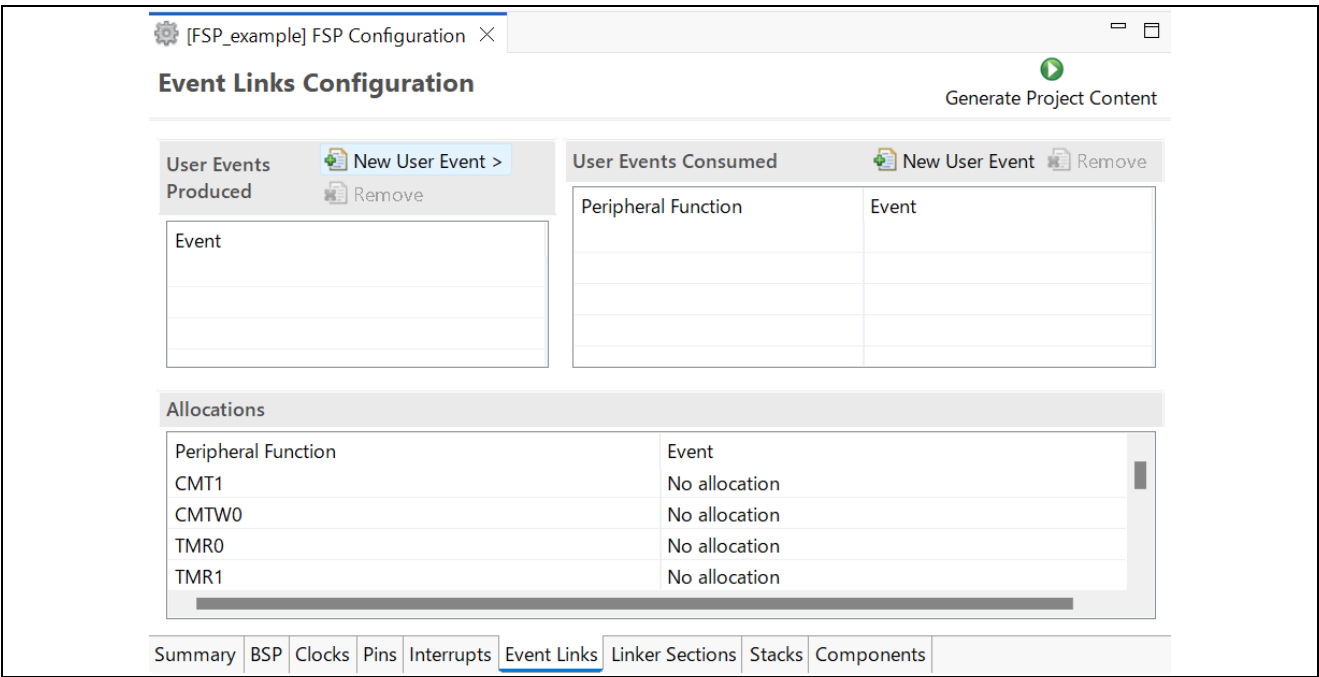


図 2-25 [Event Links Configuration]ページ

2.5.6 リンカセクションのマッピング設定

[Linker Section Mapping Configuration]ページではリンカセクションの設定が行えます。

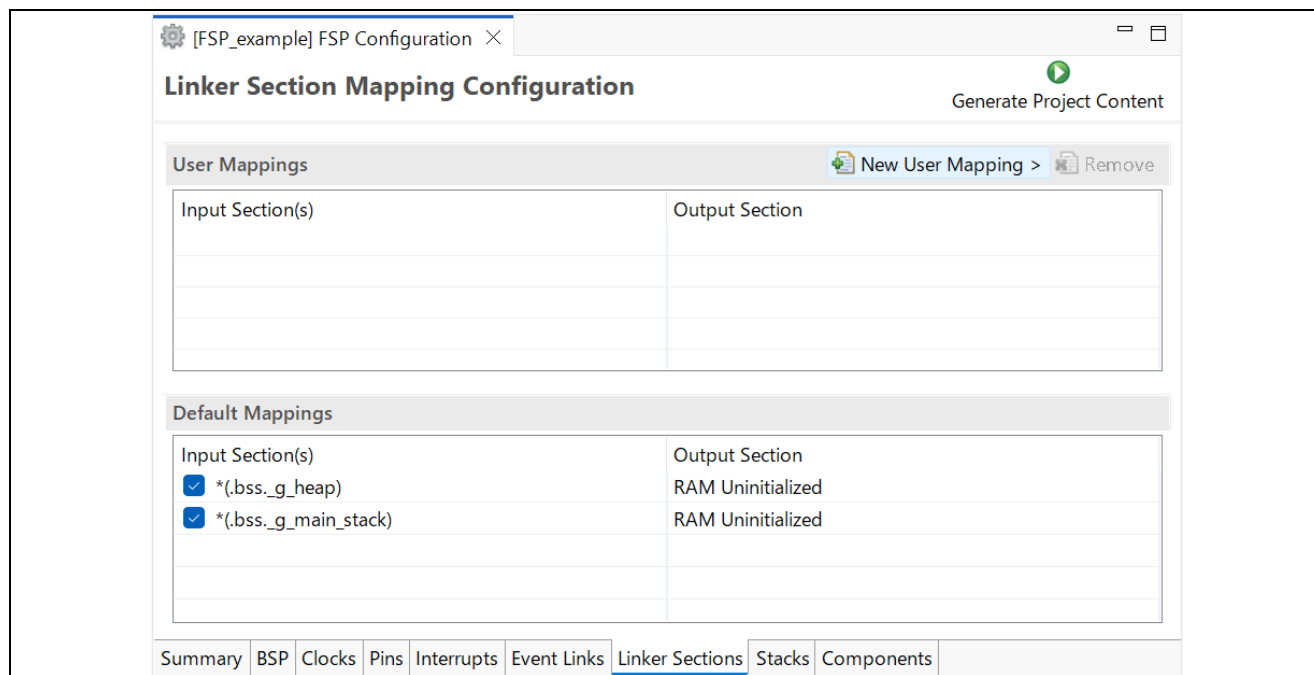


図 2-26 [Linker Section Mapping Configuration]ページ

2.5.7 コンポーネント(モジュール)設定とスタック設定

[Stacks Configuration]ページではコンポーネント(モジュール)の追加や詳細設定が行えます。詳細設定はモジュール選択時のプロパティビューから行います。設定された情報に基づいて FSP スタックが生成されます。

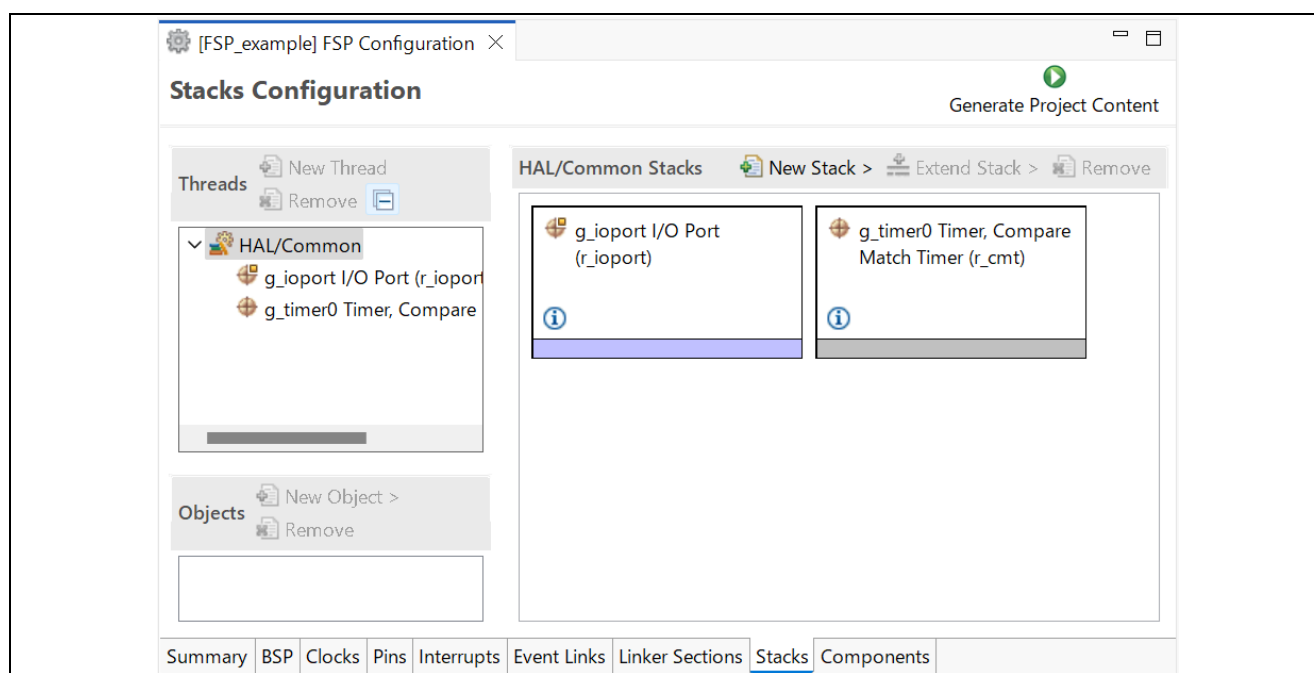
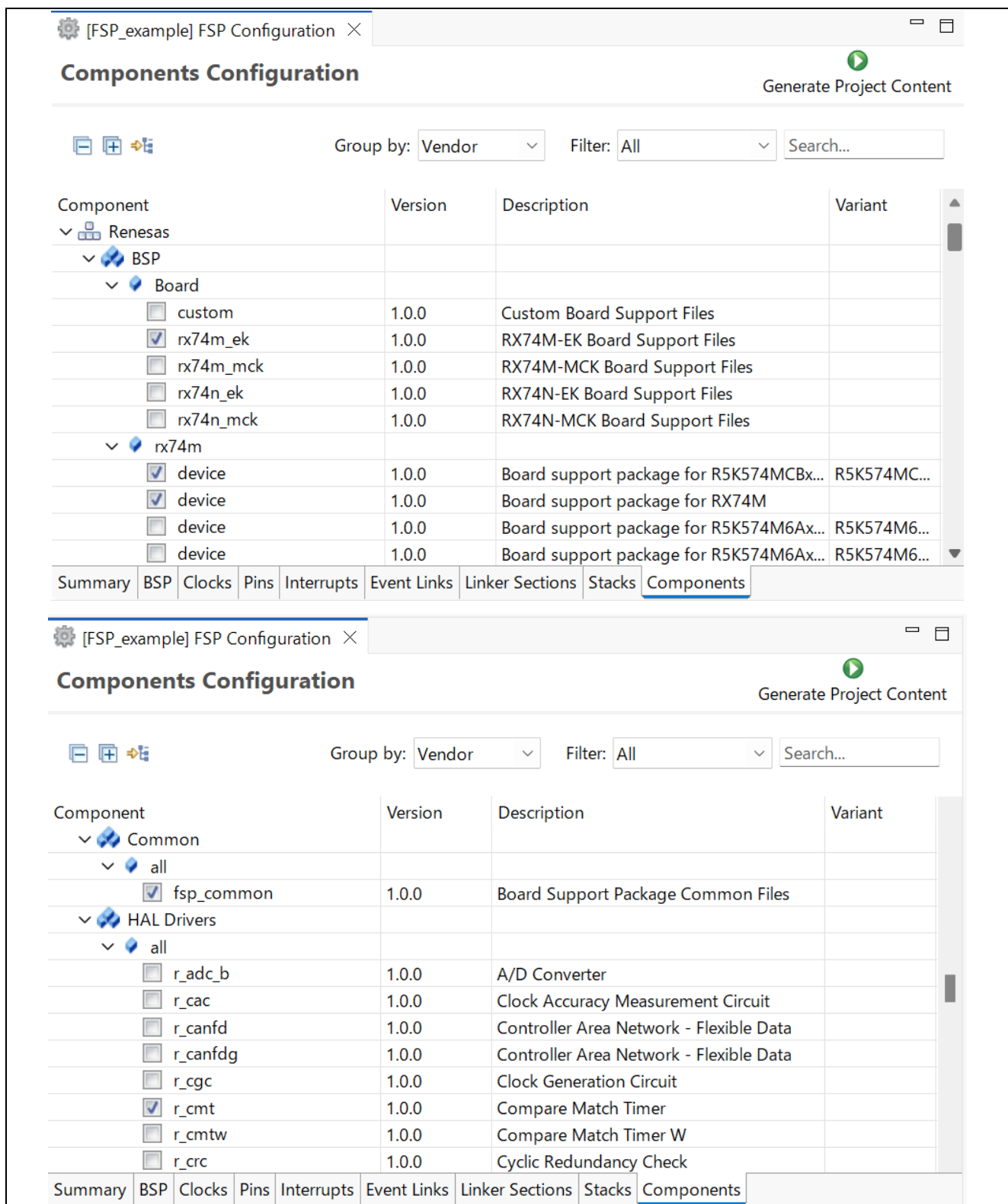


図 2-27 [Stacks Configuration]ページ

[Components Configuration]ページからもモジュールの追加や削除は行えますが、[Stacks Configuration]ページには変更したモジュールの情報が反映されません。[Components Configuration]ページからモジュールの追加や削除を行った場合でもその後に[Stacks Configuration]ページでの設定が必要になるため、モジュールの追加等の設定は[Stacks Configuration]ページから行うことを推奨します。



Components Configuration Generate Project Content

Group by: Vendor Filter: All Search...

Component	Version	Description	Variant
▼ Renesas			
▼ BSP			
▼ Board			
custom	1.0.0	Custom Board Support Files	
☒ rx74m_ek	1.0.0	RX74M-EK Board Support Files	
☐ rx74m_mck	1.0.0	RX74M-MCK Board Support Files	
☐ rx74n_ek	1.0.0	RX74N-EK Board Support Files	
☐ rx74n_mck	1.0.0	RX74N-MCK Board Support Files	
▼ rx74m			
☒ device	1.0.0	Board support package for R5K574MCBx...	R5K574MC...
☒ device	1.0.0	Board support package for RX74M	
☐ device	1.0.0	Board support package for R5K574M6Ax...	R5K574M6...
☐ device	1.0.0	Board support package for R5K574M6Ax...	R5K574M6...

Summary BSP Clocks Pins Interrupts Event Links Linker Sections Stacks **Components**

図 2-28 [Components Configuration]ページ

2.6 FSP プロジェクトのファイル構成

FSP プロジェクトのファイル構成を以下に示します。

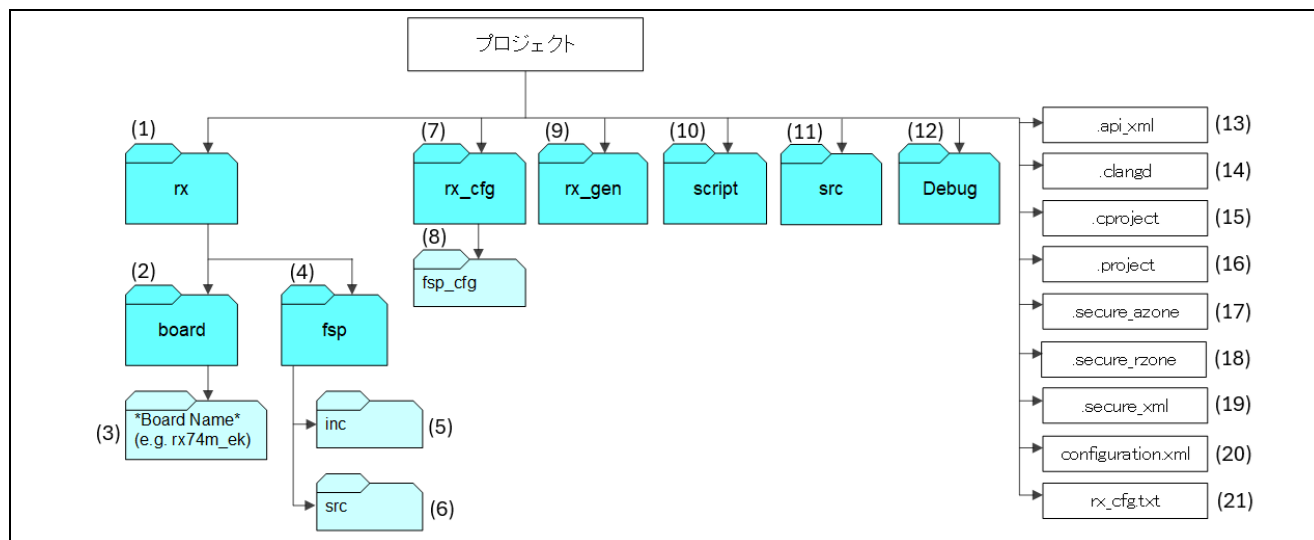


図 2-29 FSP プロジェクトファイル構成

表 2.2 プロジェクトの各フォルダの説明

No	フォルダ名	説明
1	rx	ベースとなるフォルダ
2	board	プロジェクト作成時に選択されたボードに関するファイルを出力するベースとなるフォルダ
3	*Board Name* (e.g. rx74m_ek)	ボードに関する定義を含むフォルダ。プロジェクト作成時に選択されたボードに応じた名称のフォルダが出力される。 例) 選択したボードが RX74M-EK の場合のフォルダ名は rx74m_ek。
4	fsp	FSP のベースとなるフォルダ
5	inc	コンポーネント設定で選択した FSP モジュールの定義をインクルードするためのヘッダファイルが出力されるフォルダ
6	src	コンポーネント設定で選択した FSP モジュールの定義を含むソースファイルが出力されるフォルダ
7	rx_cfg	RX プロジェクトコンフィグレーションエディタでの設定を反映した定義が含まれるヘッダファイルを出力するベースとなるフォルダ
8	fsp_cfg	RX プロジェクトコンフィグレーションエディタでの設定を反映した定義が含まれるヘッダファイルが出力されるフォルダ
9	rx_gen	RX プロジェクトコンフィグレーションエディタで生成されたソースコードが出力されるフォルダ
10	script	リンクスクリプトファイルが出力されるフォルダ
11	src	ユーザアプリケーションの main 関数相当の関数が含まれる hal_entry.c が出力されるフォルダ
12	Debug	リンクに必要なヘッダファイルやリンクスクリプトが出力されるフォルダ またビルド後はオブジェクト等ビルド結果が出力される。

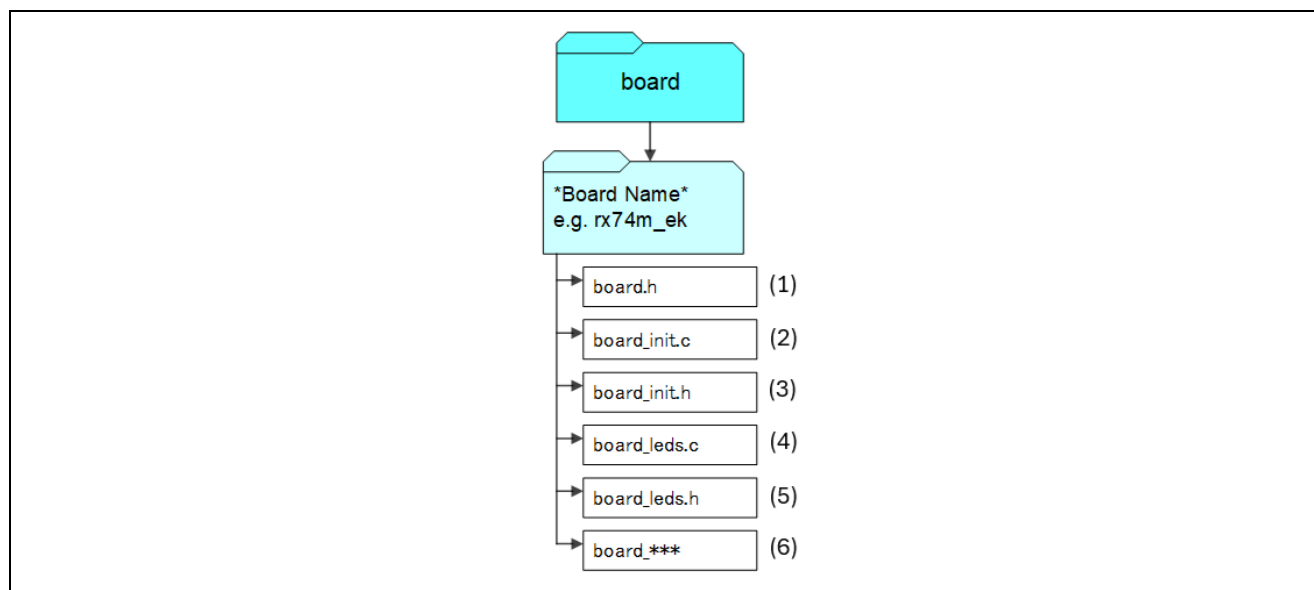


図 2-30 board フォルダのファイル構成

表 2.3 board フォルダの各ファイルの説明

No	ファイル名	説明
1	board.h	ボードに関するヘッダファイルをインクルードしたベースとなるヘッダファイル
2	board_init.c	ボード固有の初期化処理を定義したソースファイル
3	board_init.h	ボード固有の初期化処理に関する定義を含むヘッダファイル
4	board_leds.c	ボードの LED に関する定義を含むソースファイル
5	board_leds.h	ボードの LED に関する定義を含むヘッダファイル
6	board_*** (e.g. board_ethernet_phy.h)	選択したボードにより上記の他に定義がある場合出力されるソースファイルまたはヘッダファイル

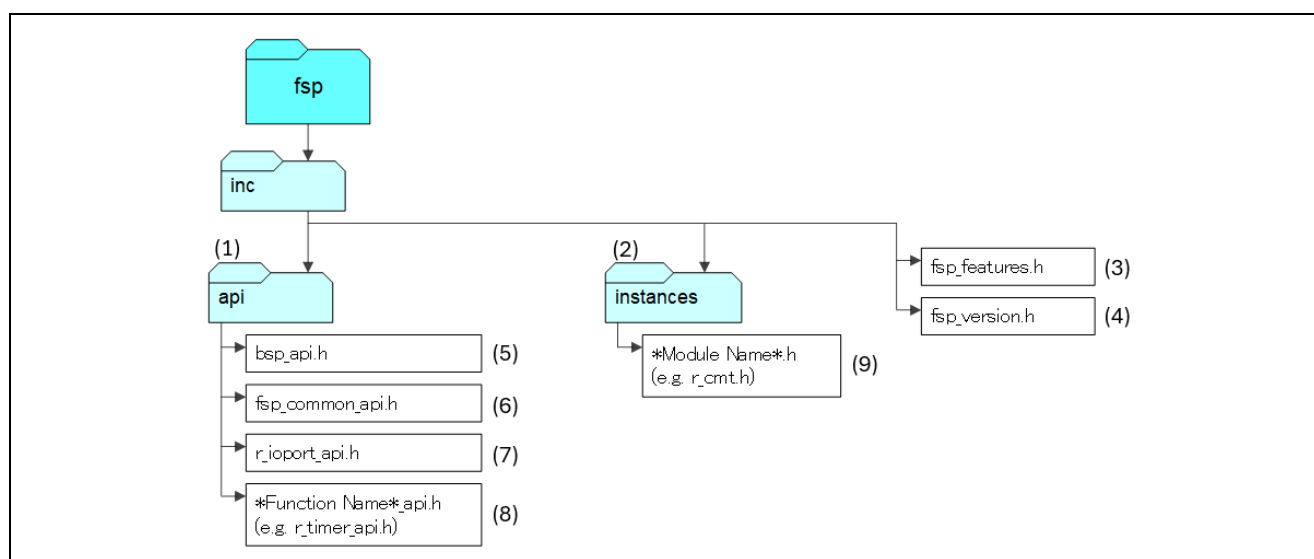


図 2-31 fsp inc フォルダのファイル構成

表 2.4 inc フォルダの各フォルダの説明

No	フォルダ名	説明
1	api	RX プロジェクトコンフィグレーションエディタでの設定したモジュールや機能の API の構造体定義を含むヘッダファイルを出力するフォルダ
2	instances	RX プロジェクトコンフィグレーションエディタでの設定したモジュールの API 関数の定義を含むヘッダファイルを出力するフォルダ

表 2.5 inc フォルダの各ファイルの説明

No	ファイル名	説明
3	fsp_features.h	FSP でサポートする機能に関する定義を含むヘッダファイル
4	fsp_version.h	FSP のバージョンの定義を含むヘッダファイル

表 2.6 api フォルダの各ファイルの説明

No	ファイル名	説明
5	bsp_api.h	BSP の API に関連する定義を含むヘッダファイル
6	fsp_common_api.h	FSP 共通の定義を含むヘッダファイル
7	r_ioport_api.h	IOPORT の API に関連する定義を含むヘッダファイル
8	*Function Name*_api.h (e.g. r_timer_api.h)	コンポーネント設定で選択した機能の API に関連する定義を含むヘッダファイル。選択した機能に応じた名称のファイルが出力される。 例) タイマの機能を選択した場合のファイル名は r_timer_api.h。

表 2.7 instances フォルダの各ファイルの説明

No	ファイル名	説明
9	*Module Name*.h (e.g. r_cmt.h)	コンポーネント設定で選択したモジュールの API に関連する定義を含むヘッダファイル。選択したモジュールに応じた名称のファイルが出力される。 例) CMT モジュール(r_cmt)を選択した場合のファイル名は r_cmt.h。

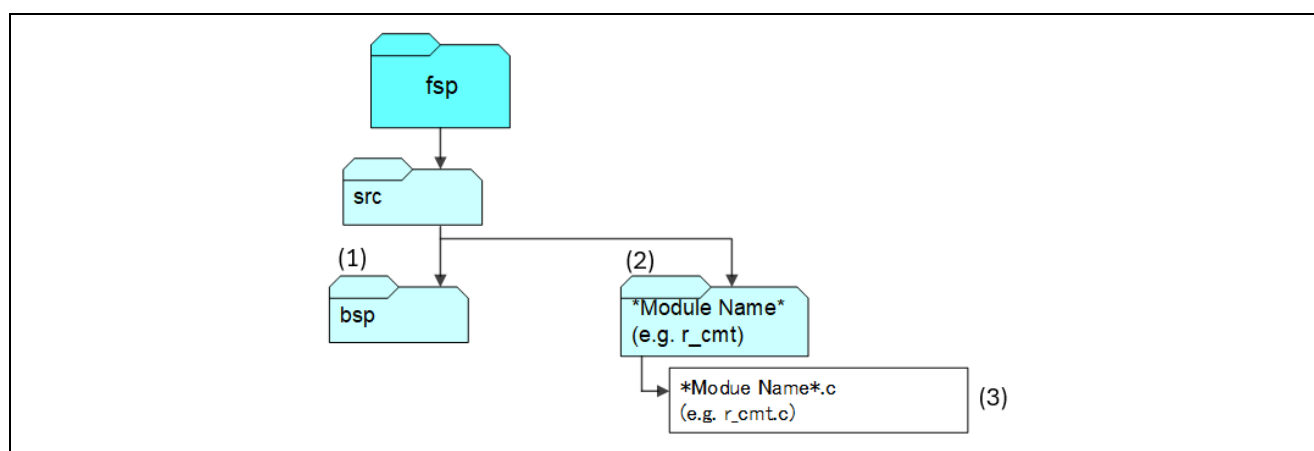


図 2-32 fsp src フォルダのファイル構成

表 2.8 src フォルダの各フォルダの説明

No	フォルダ名	説明
1	bsp	BSP のファイルを出力するベースとなるフォルダ
2	*Module Name* (e.g. r_cmt)	RX プロジェクトコンフィグレーションエディタでの設定したモジュールのファイルを出力するフォルダ。選択したモジュールに応じた名称のフォルダが出力される。 例) CMT モジュール(r_cmt)を選択した場合のフォルダ名は r_cmt。

表 2.9 *Module Name*フォルダの各ファイルの説明

No	ファイル名	説明
3	*Module Name*.c (e.g. r_cmt.c)	コンポーネント設定で選択したモジュールの API に関連する定義を含むソースファイル。選択したモジュールに応じた名称のファイルが出力される。 例) CMT モジュール(r_cmt)を選択した場合のファイル名は r_cmt.c。

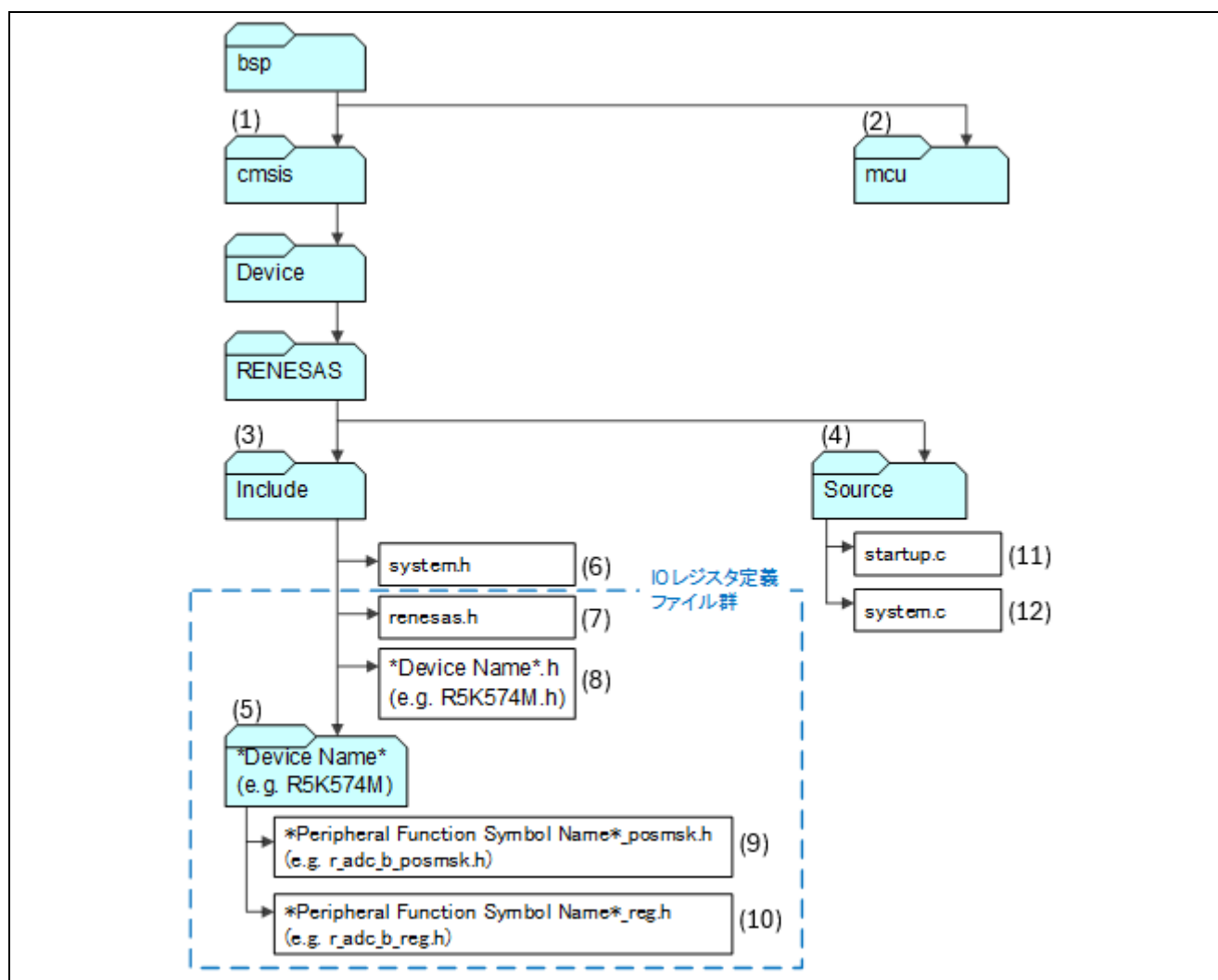


図 2-33 bsp フォルダのファイル構成

表 2.10 bsp フォルダの各フォルダの説明

No	フォルダ名	説明
1	cmsis	IO レジスタの定義やスタートアップのソースファイルを出力するベースとなるフォルダ
2	mcu	BSP の定義を含むファイルを出力するフォルダ
3	Include	IO レジスタの定義を含むヘッダファイルを出力するベースとなるフォルダ
4	Source	スタートアップに関するソースファイルを出力するフォルダ
5	*Device Name* (e.g. R5K574M)	プロジェクト作成時に選択したデバイスの各周辺機能の IO レジスタ定義を含むヘッダファイルを出力するフォルダ。選択したデバイスに応じた名称のフォルダが出力される。 例) RX74M を選択した場合のフォルダ名は R5K574M。

表 2.11 Include フォルダの各ファイルの説明

No	ファイル名	説明
6	system.h	システムの初期化に関する定義や処理を含むヘッダファイル
7	renesas.h	FSP での IO レジスタ定義ファイル群の起点となるヘッダファイル。*Device Name*.h をインクルードしている。
8	*Device Name*.h (e.g. R5K74M.h)	該当デバイスの IO レジスタの定義ファイル群のベースとなるヘッダファイル。*Device Name* フォルダ以下のすべての周辺機能のヘッダファイルをインクルードしている。選択したデバイス名に応じた名称のファイルが出力される。 例) RX74M を選択した場合のファイル名は R5K574M.h。
9	*Peripheral Function Symbol Name*_posmsk.h (e.g. r_adc_b_posmsk.h)	各周辺機能のレジスタのビットフィールドの位置とマスクの定義を含むヘッダファイル。周辺機能のシンボル名に応じた名称のファイルが出力される。 例) AD の場合のファイル名は r_adc_b_posmsk.h。
10	*Peripheral Function Symbol Name*_reg.h (e.g. r_adc_b_reg.h)	各周辺機能のレジスタ構造の定義を含むヘッダファイル。周辺機能のシンボル名に応じた名称のファイルが出力される。 例) AD の場合のファイル名は r_adc_b_reg.h。

表 2.12 Source フォルダの各ファイルの説明

No	ファイル名	説明
11	startup.c	リセットベクタおよび例外ベクタの定義やスタートアップ処理を含むソースファイル
12	system.c	システムの初期化に関する定義や処理を含むソースファイル

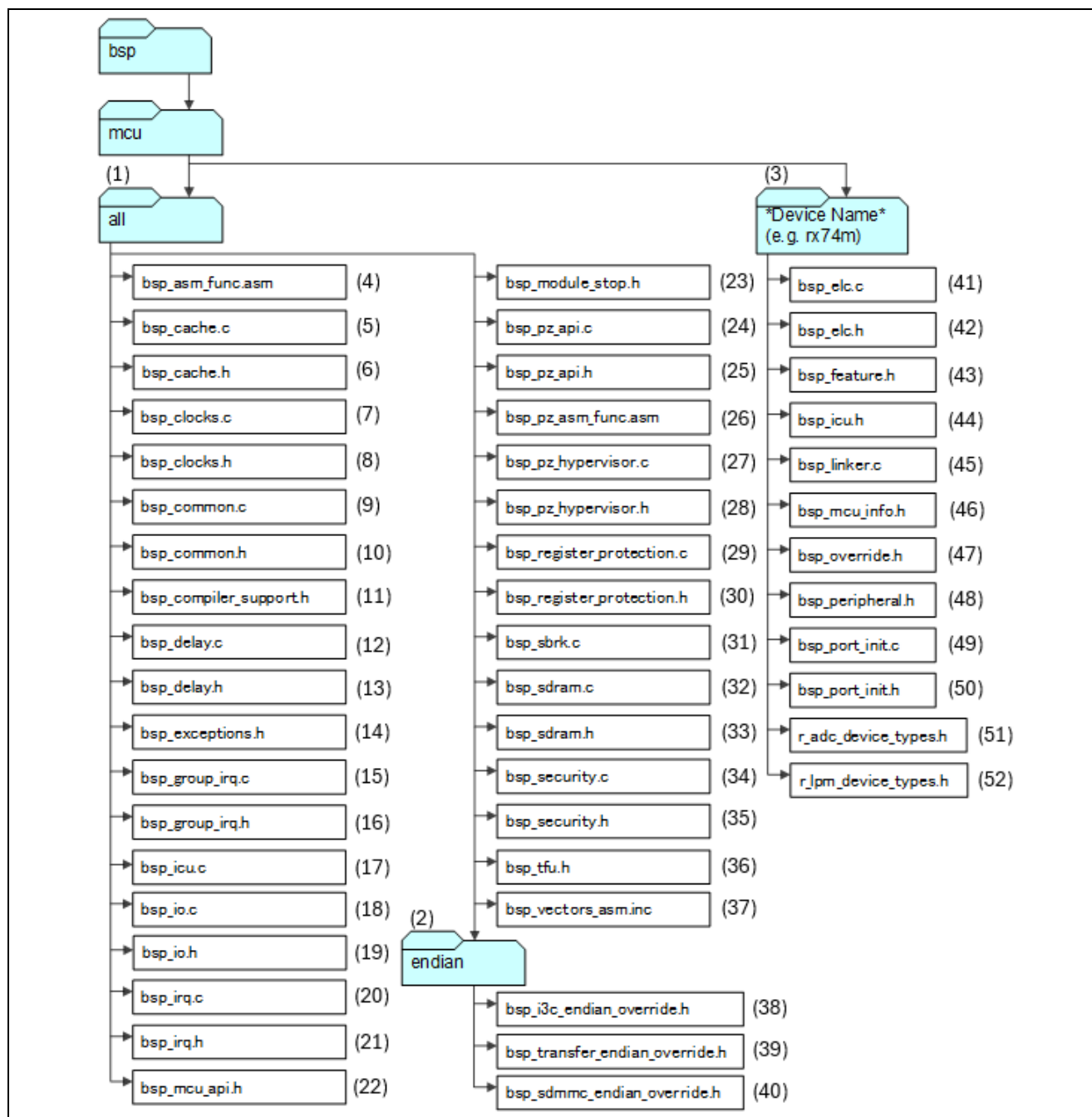


図 2-34 mcu フォルダのファイル構成

表 2.13 mcu フォルダの各フォルダの説明

No	フォルダ名	説明
1	all	BSP のデバイス共通の定義を含むファイルを出力するフォルダ
2	endian	BSP のエンディアンが関連する定義を含むファイルを出力するフォルダ
3	*Device Name* (e.g. rx74m)	BSP のデバイス固有の定義を含むファイルを出力するフォルダ。選択したデバイスに応じた名称のフォルダが出力される。 例) RX74M を選択した場合のフォルダ名は rx74m。

表 2.14 all フォルダの各ファイルの説明(1)

No	ファイル名	説明
4	bsp_asm_func.asm	アセンブラ関数の定義を含むソースファイル
5	bsp_cache.c	キャッシュ制御に関する定義を含むソースファイル
6	bsp_cache.h	キャッシュ制御に関する定義を含むヘッダファイル
7	bsp_clocks.c	クロック発生回路の設定に関連する定義を含むソースファイル
8	bsp_clocks.h	クロック発生回路の設定に関連する定義を含むヘッダファイル
9	bsp_common.c	BSP 共通の定義を含むソースファイル
10	bsp_common.h	BSP 共通の定義を含むヘッダファイル
11	bsp_compiler_support.h	コンパイラに関連する定義を含むヘッダファイル
12	bsp_delay.c	ソフトウェアによる遅延処理に関する定義を含むソースファイル
13	bsp_delay.h	ソフトウェアによる遅延処理に関する定義を含むヘッダファイル
14	bsp_exceptions.h	例外処理に関する定義を含むヘッダファイル
15	bsp_group_irq.c	グループ割り込みに関する定義を含むソースファイル
16	bsp_group_irq.h	グループ割り込みに関する定義を含むヘッダファイル
17	bsp_icu.c	割り込み設定に関する定義を含むソースファイル
18	bsp_io.c	IO ポートに関する定義を含むソースファイル
19	bsp_io.h	IO ポートに関する定義を含むヘッダファイル
20	bsp_irq.c	割り込み処理に関する定義を含むソースファイル
21	bsp_irq.h	割り込み処理に関する定義を含むヘッダファイル
22	bsp_mcu_api.h	BSP のデバイス共通の API 関数に関する定義を含むヘッダファイル
23	bsp_module_stop.h	消費電力低減機能のモジュールストップコントロールレジスタに関する定義を含むヘッダファイル
24	bsp_pz_api.c	Protect Zone のワールド遷移に関する API 関数の定義を含むソースファイル
25	bsp_pz_api.h	Protect Zone のワールド遷移に関する API 関数の定義を含むヘッダファイル
26	bsp_pz_asm_func.asm	Protect Zone の制御に関するアセンブラ関数の定義を含むソースファイル
27	bsp_pz_hypervisor.c	Protect Zone の制御に関する定義を含むソースファイル
28	bsp_pz_hypervisor.h	Protect Zone の制御に関する定義を含むヘッダファイル
24	bsp_register_protection.c	レジスタライトプロテクション機能に関する定義を含むソースファイル
25	bsp_register_protection.h	レジスタライトプロテクション機能に関する定義を含むヘッダファイル
26	bsp_sbrk.c	低水準インタフェース関数の sbrk 関数の定義を含むソースファイル

表 2.15 all フォルダの各ファイルの説明(2)

No	ファイル名	説明
27	bsp_sdram.c	SDRAM に関する定義を含むソースファイル
28	bsp_sdram.h	SDRAM に関する定義を含むヘッダファイル
29	bsp_security.c	セキュリティ機能の初期化に関する定義を含むソースファイル
30	bsp_security.h	セキュリティ機能の初期化に関する定義を含むヘッダファイル
31	bsp_tfu.h	三角関数演算器に関する定義を含むヘッダファイル
32	bsp_vectors_asm.inc	ベクタに関する定義を含むインクルードファイル

表 2.16 endian フォルダの各ファイルの説明

No	ファイル名	説明
38	bsp_i3c_endian_override.h	I3C のエンディアンに関する定義を含むヘッダファイル
39	bsp_sdmmc_endian_override.h	SDMMC のエンディアンに関する定義を含むヘッダファイル
40	bsp_transfer_endian_override.h	転送のエンディアンに関する定義を含むヘッダファイル

表 2.17 *Device Name*フォルダの各ファイルの説明

No	ファイル名	説明
41	bsp_elc.c	イベントリンクコントローラに関する定義を含むソースファイル
42	bsp_elc.h	イベントリンクコントローラに関する定義を含むヘッダファイル
43	bsp_feature.h	デバイスがサポートする機能に関連する定義を含むヘッダファイル
44	bsp_icu.h	割り込み設定に関する定義を含むヘッダファイル
45	bsp_linker.c	オプション設定に関する定義を含むソースファイル
46	bsp_mcu_info.h	デバイス情報に関する定義を含むヘッダファイル
47	bsp_override.h	クロック設定や割り込み設定の変更に関する定義を含むヘッダファイル
48	bsp_peripheral.h	周辺機能に関する定義を含むヘッダファイル
49	bsp_port_init.c	ポート方向レジスタ(PDR)の初期化に関する定義を含むソースファイル
50	bsp_port_init.h	ポート方向レジスタ(PDR)の初期化に関する定義を含むヘッダファイル
51	r_adc_device_types.h	A/D コンバータに関する定義を含むヘッダファイル
52	r_lpm_device_types.h	消費電力低減機能に関する定義を含むヘッダファイル

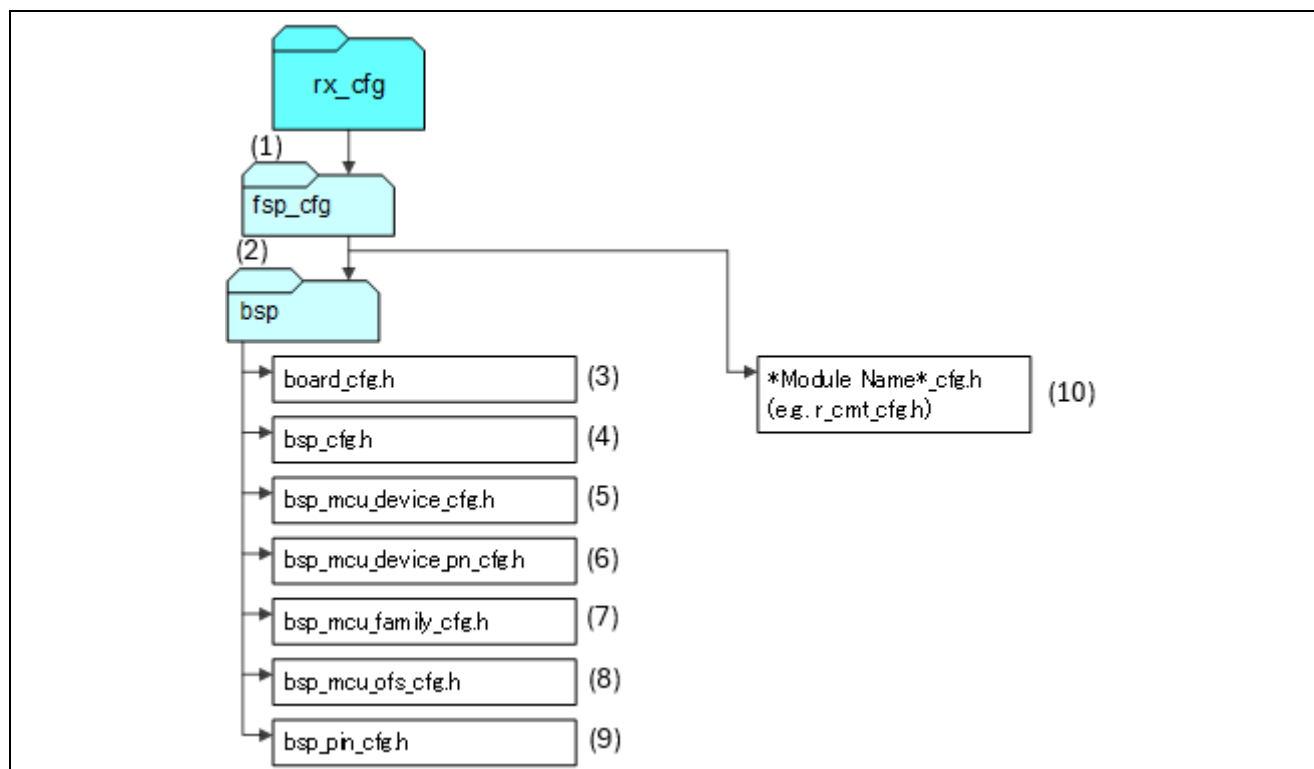


図 2-35 rx_cfg フォルダのファイル構成

表 2.18 rx_cfg フォルダの各フォルダの説明

No	フォルダ名	説明
1	fsp_cfg	RX プロジェクトコンフィグレーションエディタでの設定したモジュールの設定情報を反映した定義を含むファイルを出力するベースとなるフォルダ
2	bsp	BSP の設定情報を反映した定義を含むファイルを出力するフォルダ

表 2.19 bsp フォルダの各ファイルの説明

No	ファイル名	説明
3	board_cfg.h	board.h へのインクルードパスを含むヘッダファイル
4	bsp_cfg.h	BSP の設定情報を反映した定義を含むヘッダファイル
5	bsp_mcu_device_cfg.h	MCU デバイスに関する設定を反映した定義を含むヘッダファイル
6	bsp_mcu_device_pn_cfg.h	MCU デバイスのパッケージに関する定義を含むヘッダファイル
7	bsp_mcu_family_cfg.h	MCU ファミリに関する設定を反映した定義を含むヘッダファイル
8	bsp_mcu_ofs_cfg.h	MCU デバイスのオプション設定メモリに関する定義を含むヘッダファイル
9	bsp_pin_cfg.h	端子設定に関する設定を反映した定義を含むヘッダファイル

表 2.20 fsp_cfg フォルダの各ファイルの説明

No	ファイル名	説明
10	*Module Name*_cfg.h (e.g. r_cmt_cfg.h)	コンポーネント設定で選択したモジュールの設定に関連する定義を含むヘッダファイル。選択したモジュールに応じた名称のファイルが出力される。 例) CMT モジュール(r_cmt)を選択した場合のファイル名は r_cmt_cfg.h。

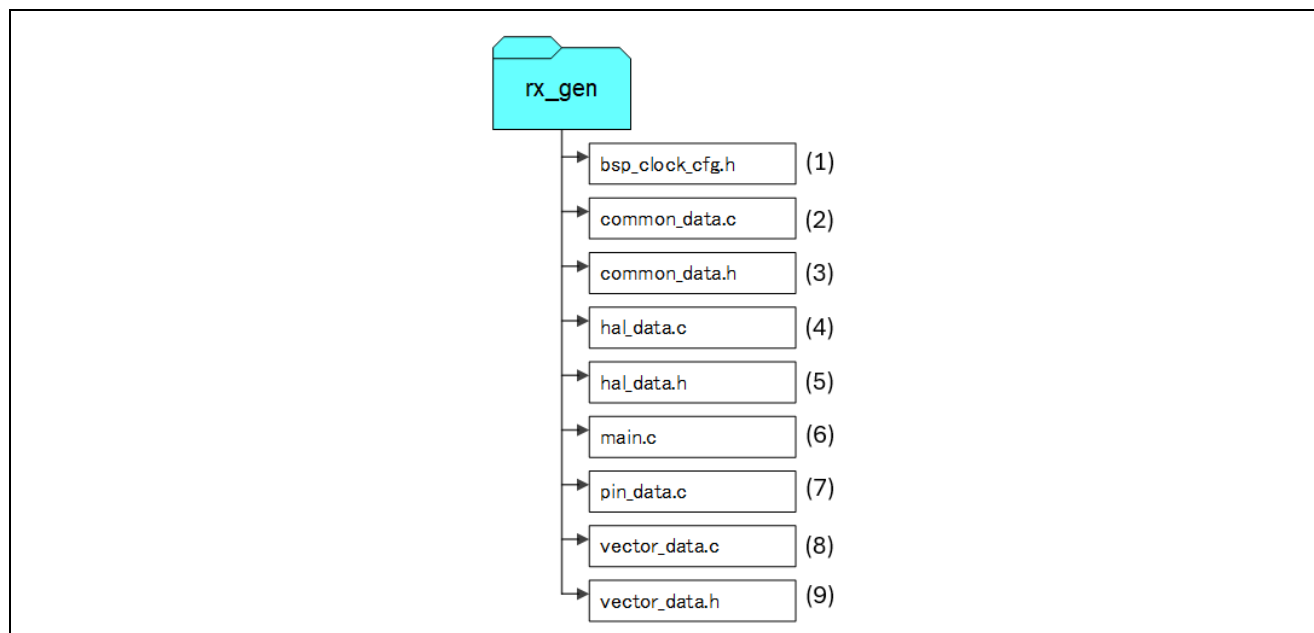


図 2-36 rx_gen フォルダのファイル構成

表 2.21 rx_gen フォルダの各ファイルの説明

No	ファイル名	説明
1	bsp_clock_cfg.h	クロック設定での設定を反映した定義を含むヘッダファイル
2	common_data.c	RX プロジェクトコンフィグレーションエディタでの設定を反映した FSP 共通の定義を含むソースファイル
3	common_data.h	RX プロジェクトコンフィグレーションエディタでの設定を反映した FSP 共通の定義を含むヘッダファイル
4	hal_data.c	コンポーネント設定で選択したモジュールの設定を反映した定義を含むソースファイル
5	hal_data.h	コンポーネント設定で選択したモジュールの設定を反映した定義を含むヘッダファイル
6	main.c	main 関数を定義したソースファイル
7	pin_data.c	端子設定での設定を反映した定義を含むソースファイル
8	vector_data.c	割り込み設定での設定を反映した割り込みベクタテーブルと割り込み処理の定義を含むソースファイル
9	vector_data.h	割り込み設定での設定を反映した割り込みベクタテーブルと割り込み処理の定義を含むヘッダファイル

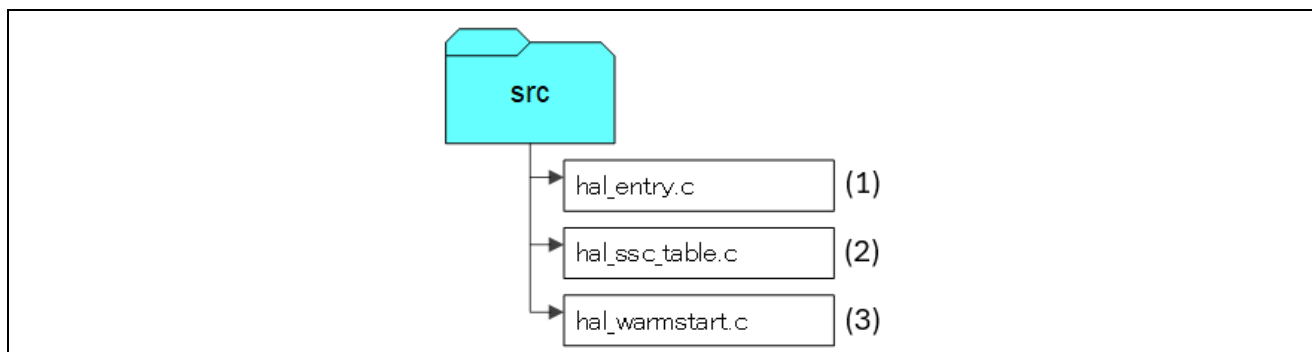


図 2-37 src フォルダのファイル構成

表 2.22 src フォルダの各ファイルの説明

No	ファイル名	説明
1	hal_entry.c	FSP プロジェクトにおける main 関数に相当する関数の定義を含むソースファイル
2	hal_ssc_table.c	セキュアワールドのハンドラの定義を含むソースファイル
3	hal_warmstart.c	ウォームスタートの関数の定義を含むソースファイル

2.7 FSP モジュールの API の情報の確認方法

FSP モジュールの API の情報の詳細は以下の方法で確認することができます。

[Stacks Configuration]ページで API の詳細を確認したいモジュールのインフォメーションアイコンをクリックします。（またはモジュールを右クリックし表示されたメニューから[Module Resources...]を選択してください。）

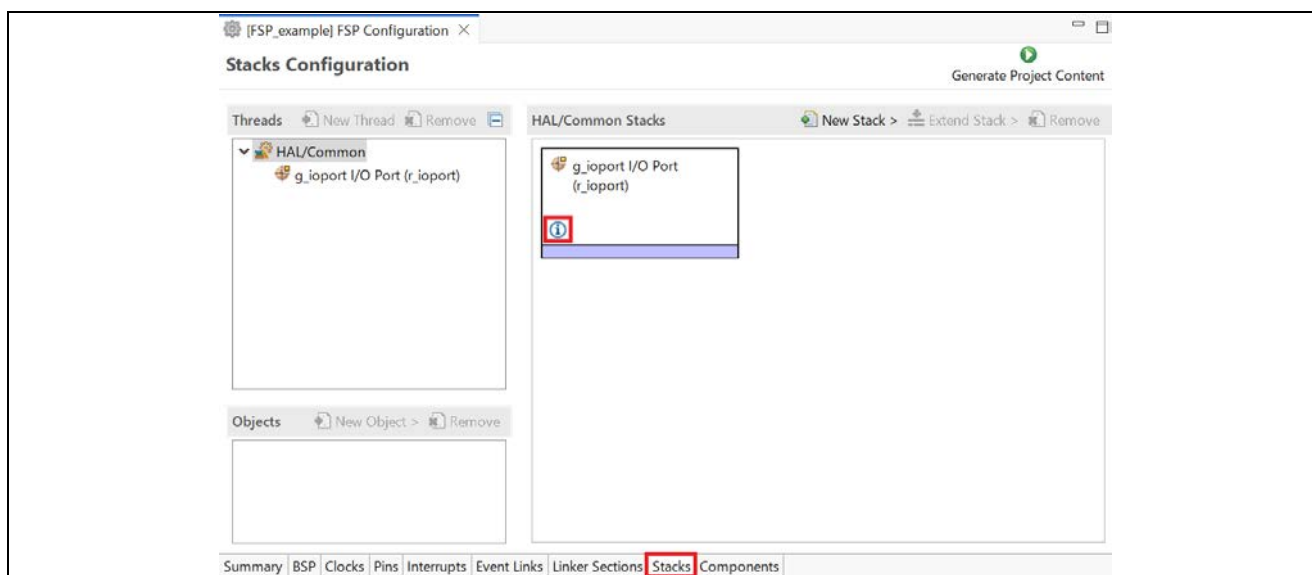


図 2-38 [Stacks Configuration]ページ

選択したモジュールの API マニュアルのドキュメントページが表示されます。

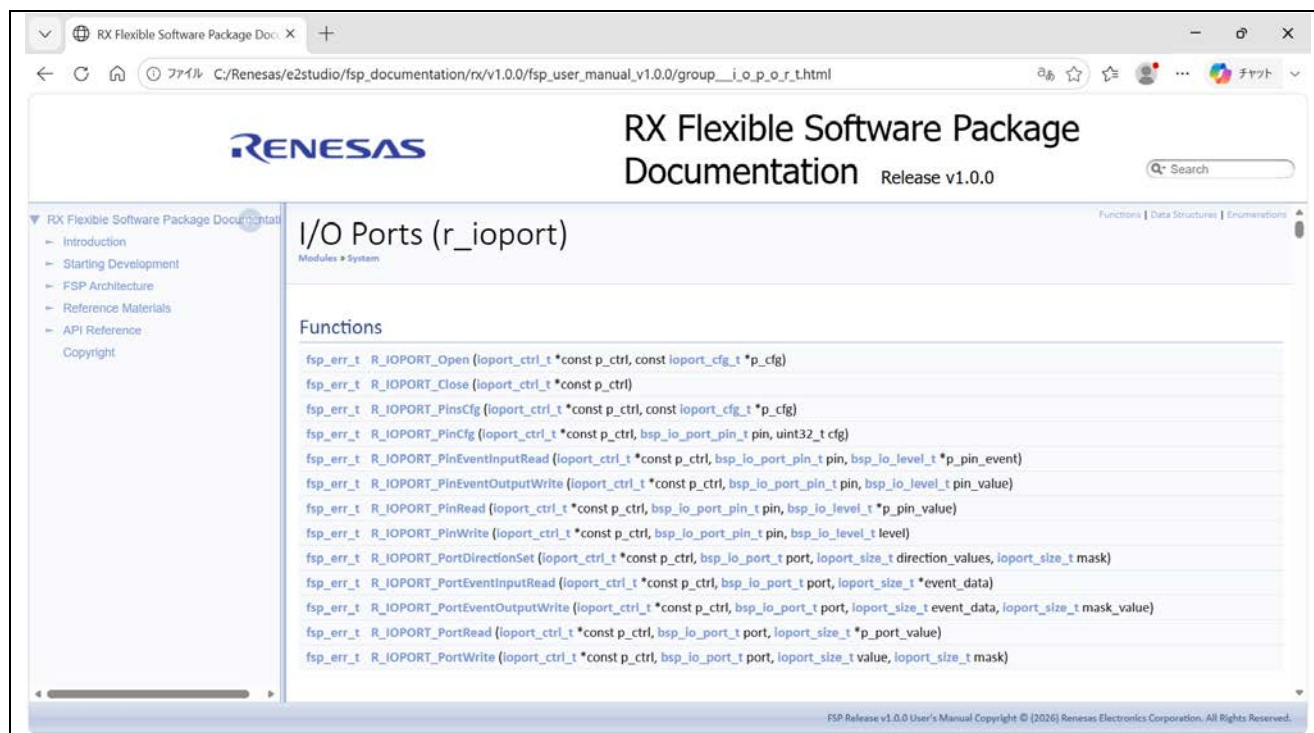


図 2-39 FSP ドキュメントのモジュールのヘルプ

3. FSP 開発環境への移行

本章では、既存の e² studio のスマート・コンフィグレータ環境から FSP プロジェクトの環境への移行について説明します。

3.1 スマート・コンフィグレータと FSP Configuration パースペクティブの操作の違い

スマート・コンフィグレータパースペクティブから[FSP Configuration]パースペクティブへの移行のため、操作面について変わった箇所を説明します。スマート・コンフィグレータビューが RX プロジェクトコンフィグレーションエディタビューに(赤枠)、MCU/MPU パッケージビューが[Visualization]ビューに(青枠)対応しています。

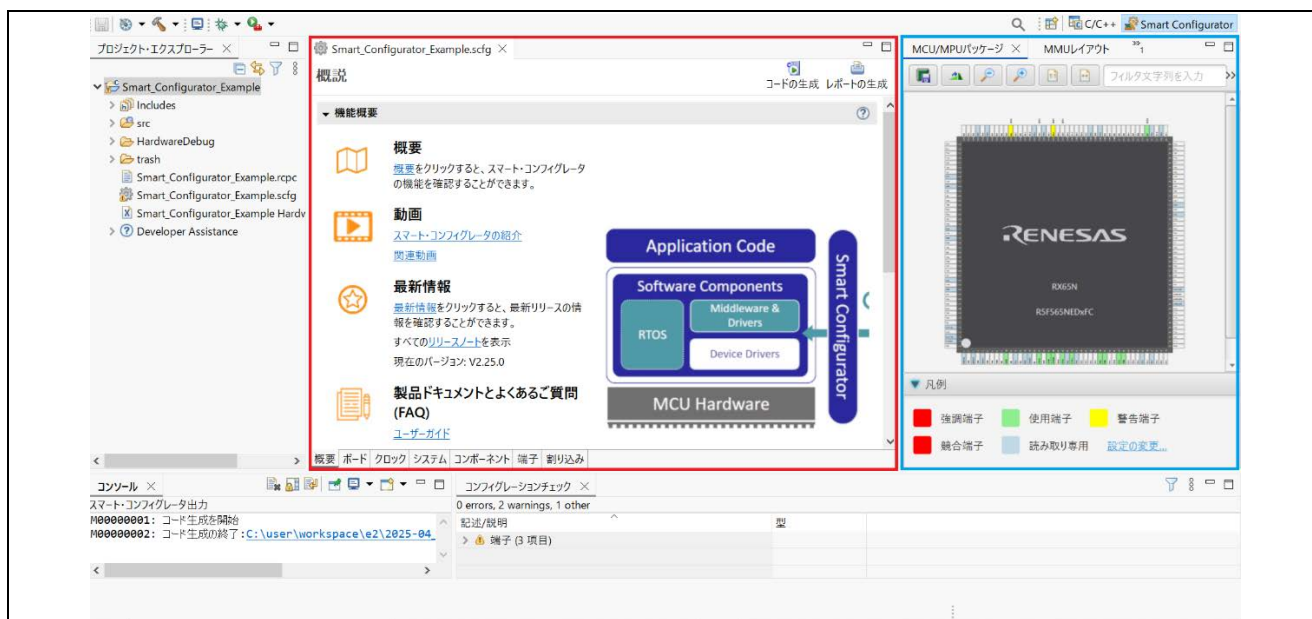


図 3-1 [スマート・コンフィグレータ]パースペクティブ

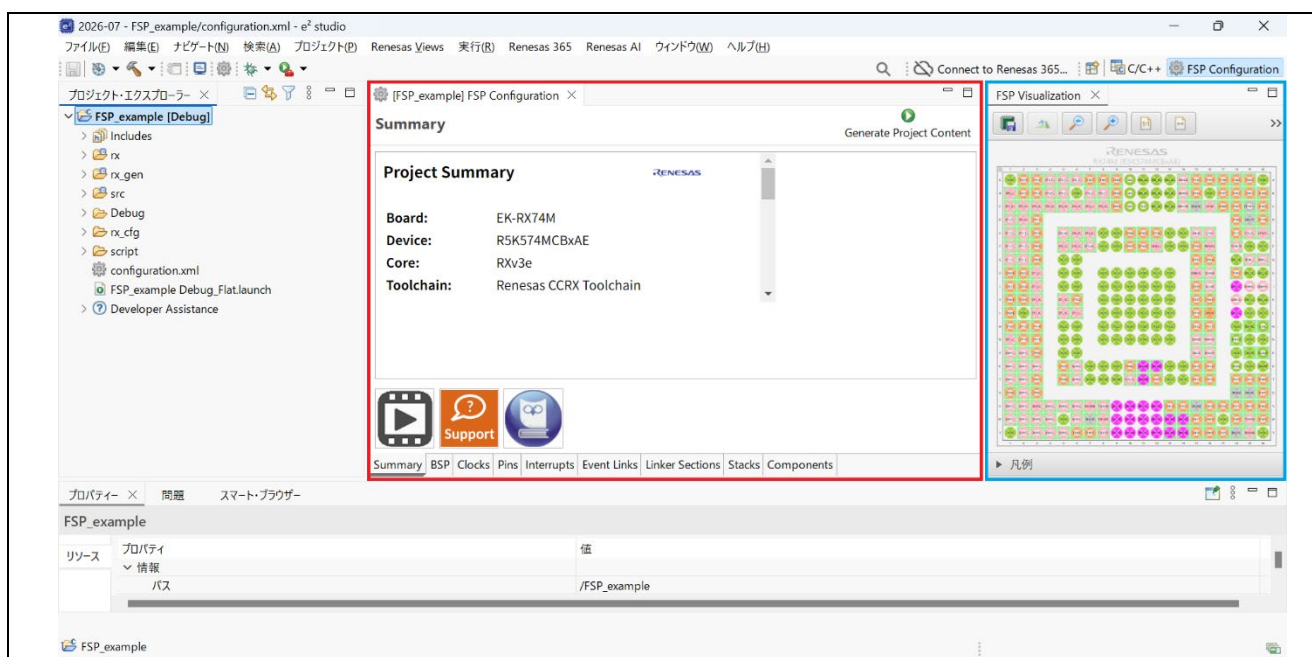


図 3-2 [FSP Configuration]パースペクティブ

3.1.1 概要とコンポーネントの状態の確認

スマート・コンフィグレータビューの概要ページでは、スマート・コンフィグレータの機能の概要や紹介動画、最新情報やFAQ、プロジェクトで使用しているコンポーネントの状態や各種モジュールのバージョンなどを確認することができます。

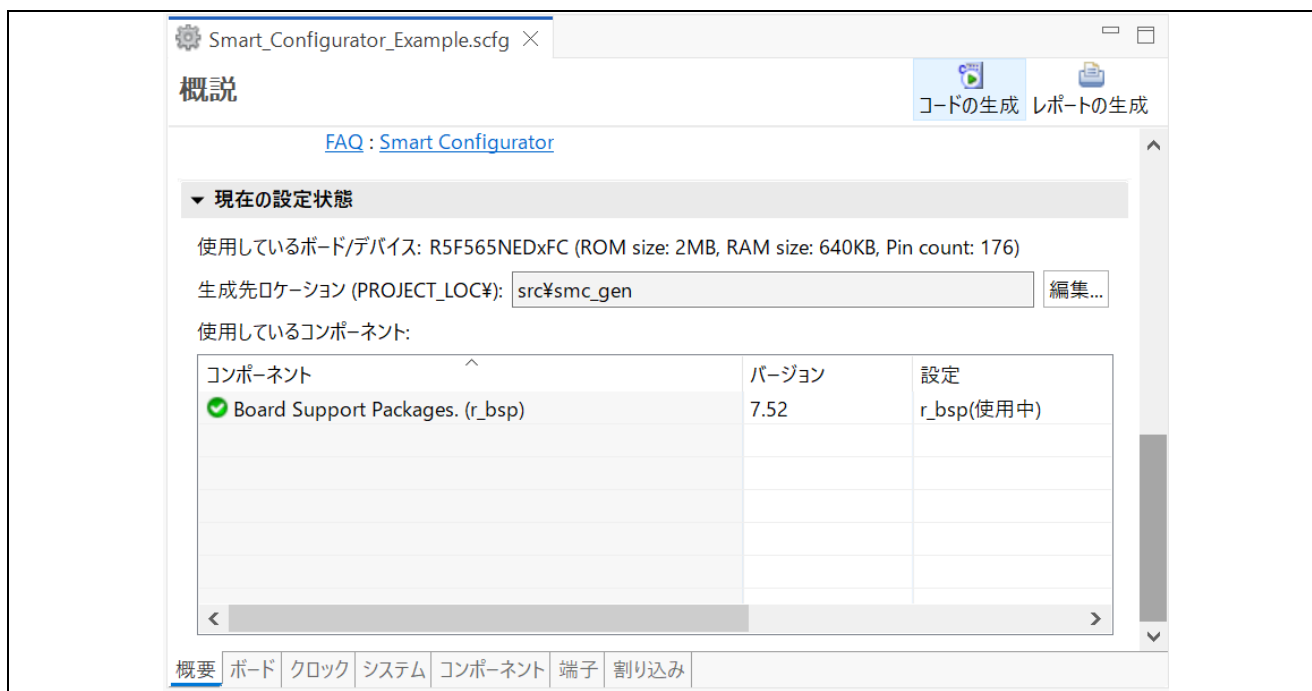


図 3-3 [スマート・コンフィグレータビュー]概要ページ

一方、RX プロジェクトコンフィグレーションエディタビューでは概要 (Summary) ページにてプロジェクトで使用しているコンポーネントの状態やバージョンの他にボード、デバイス、コンパイラのバージョンなども確認することができます。

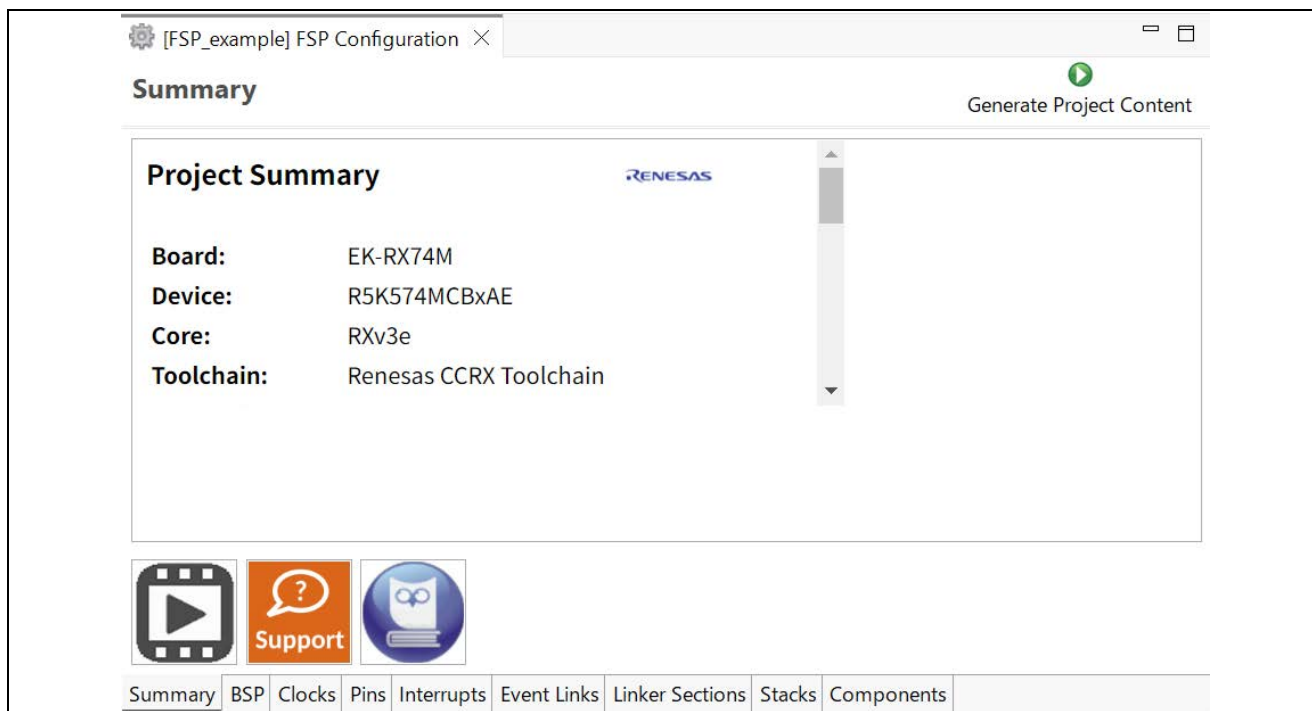


図 3-4 [Summary]ページ

3.1.2 ボード、デバイス選択

スマート・コンフィグレータビューではボードやデバイスの選択はボードページから行うことができます。



図 3-5 [スマート・コンフィグレータビュー]ボードページ

一方、RX プロジェクトコンフィグレーションエディタビューでは BSP ページで FSP のバージョン、ボードやデバイスの選択が行えます。

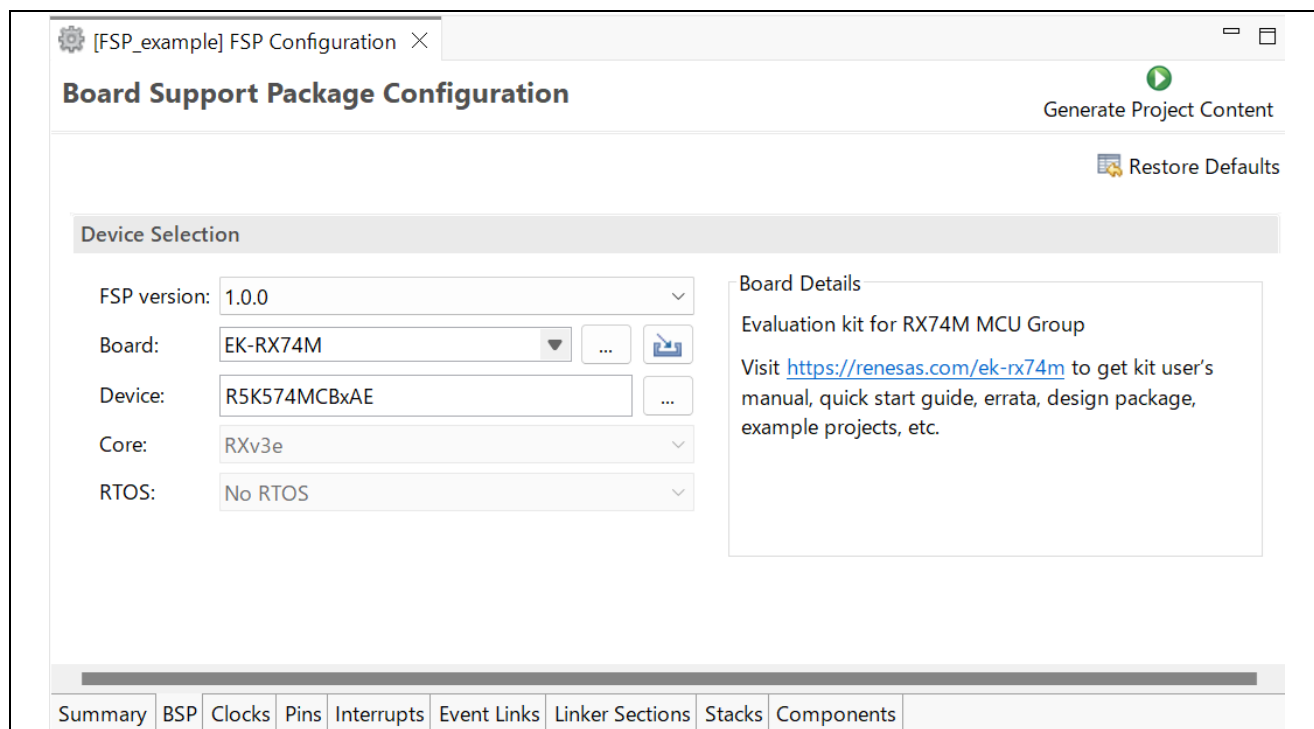


図 3-6 [BSP]ページ

3.1.3 クロック設定

スマート・コンフィグレータビューではクロックの設定はクロックページから行うことができます。

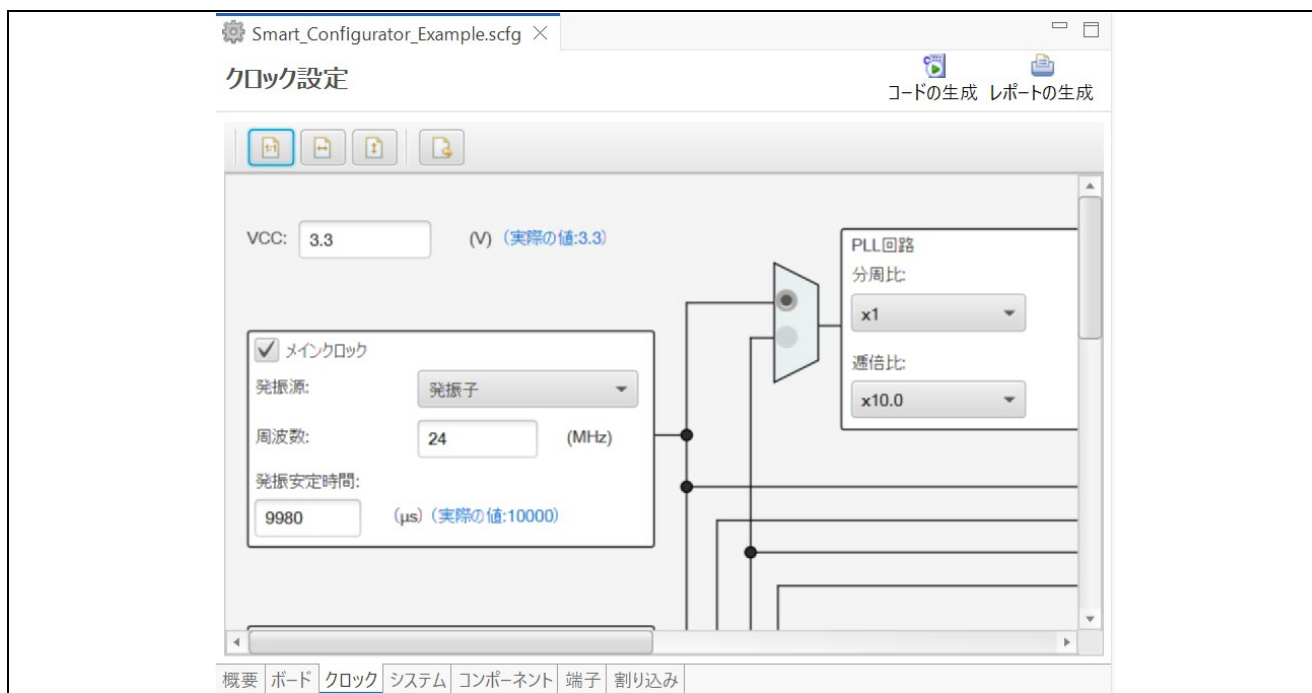


図 3-7 [スマート・コンフィグレータビュー]クロックページ

一方、RX プロジェクトコンフィグレーションエディタビューでは[Clocks Configuration]ページでクロックの設定が行えます。ただし、スマート・コンフィグレータビューではクロックソースとなるすべてのクロックの発振停止を設定することができましたが、[Clocks Configuration]ページではクロックの発振停止を設定することはできません。

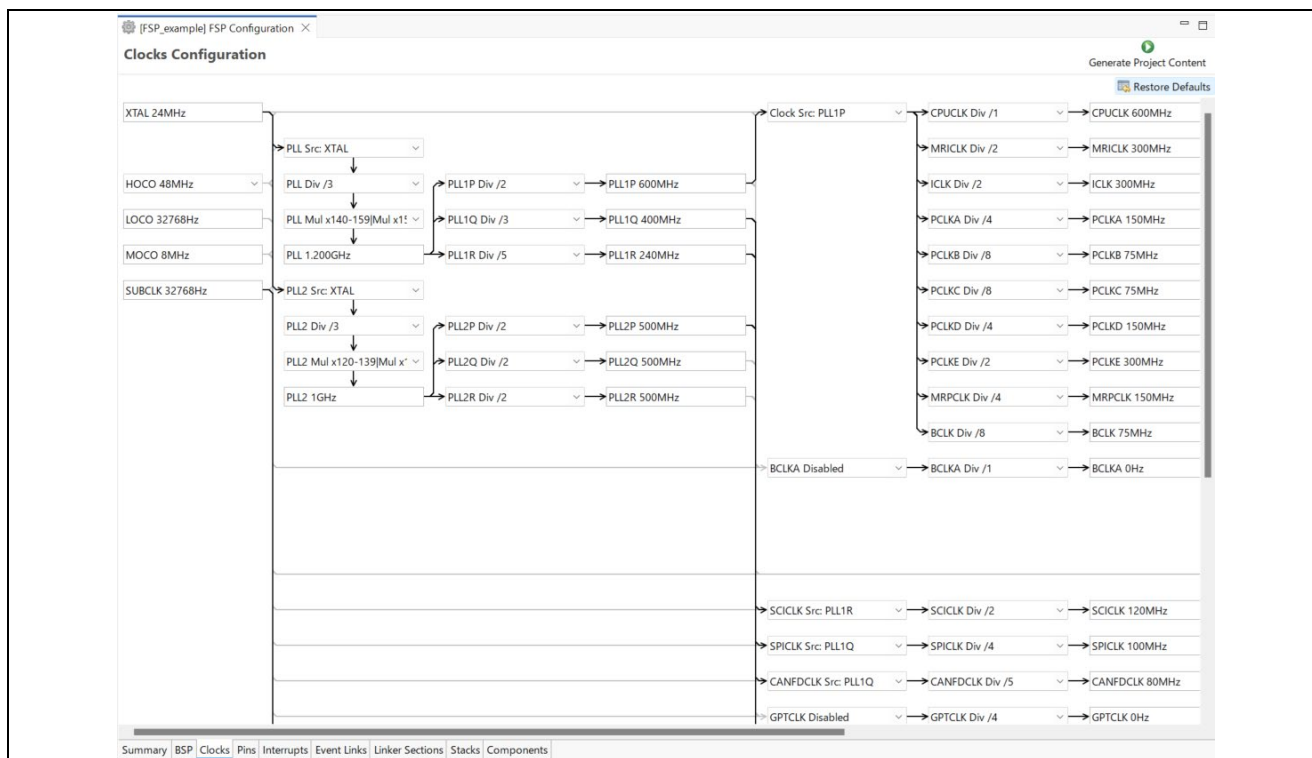


図 3-8 [Clocks Configuration]ページ

クロックの発振・発振停止は BSP ページ選択時のプロパティビューで選択可能なクロックソースについてのみ設定することができます。

Summary	BSP	Clocks	Pins	Interrupts	Event Links	Linker Sections	Stacks	Components
プロパティ	×	問題	スマート・ブラウザー					
EK-RX74M								
Settings	プロパティ							値
	Clock Registers not Reset Values during Startup							Disabled
	Main Oscillator Populated							Populated
	PFS Protect							Enabled
	C Runtime Initialization							Enabled
	Early BSP Initialization							Disabled
	Main Oscillator Clock Source							Crystal or Resonator
	Subclock Populated							Populated
	Subclock Drive (Drive capacitance availability varies by MCU)							Standard/Normal mode
	Subclock Stabilization Time (ms)							1000

図 3-9 [プロパティビュー]BSP ページ選択時の発振/停止設定項目

オンチップオシレータ（HOCO、LOCO など）を含むプロパティビューで選択できないクロックソースは発振停止に設定にすることはできません。振停止したい場合は以下実装箇所一例などにお客様自身で発振停止処理を実装してください。

表 3.1 発振停止処理の実装箇所一例

No	ファイル名	実装箇所
1	bsp_clocks.c	bsp_clock_init 関数（クロック初期設定関数）内 ^(注)
2	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_POST_CLOCK（クロック初期設定直後）もしくは BSP_WARM_START_POST_C の条件分岐内
3	hal_entry.c	hal_entry 関数内（スタートアップ完了後）

【注】 実装後に FSP バージョンを変更し、Generate Project Content を実行した場合、実装内容は保持されません。その場合は、再度実装してください。

3.1.4 デバッグインタフェース設定

スマート・コンフィグレータビューではシステムページでデバッグインタフェースの設定を行うことができます。



図 3-10 [スマート・コンフィグレータビュー]システムページ

一方、RX プロジェクトコンフィグレーションエディタビューにはデバッグインタフェースを設定するページはありません。デバッグインタフェースに使用する端子は[Visualization]ビューもしくは[Pin Configuration]ページから設定します。(3.1.6 端子設定を参照) デバッグインタフェースの種類はデバッグ構成の Debugger 設定で設定します。

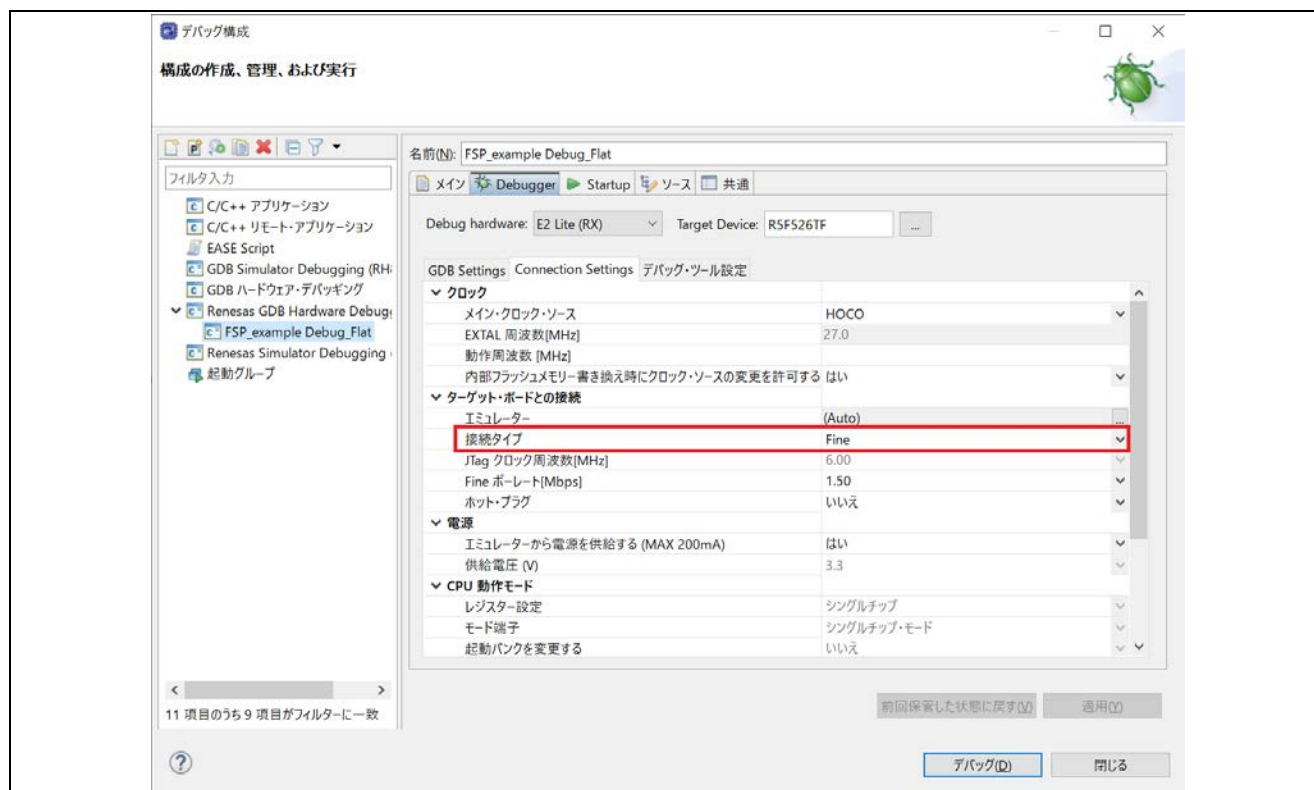


図 3-11 デバッグ構成

3.1.5 コンポーネント設定

スマート・コンフィグレータビューではボードページまたはコンポーネントページでコンポーネントの追加を行うことができます。コンポーネントは[Firmware Integration Technology] (FIT モジュール) または [コード生成]の 2 種類のコンポーネントから選択することができ、コンポーネントの詳細設定はコンポーネントページで各コンポーネントのモジュールを選択することでプロパティから設定可能です。

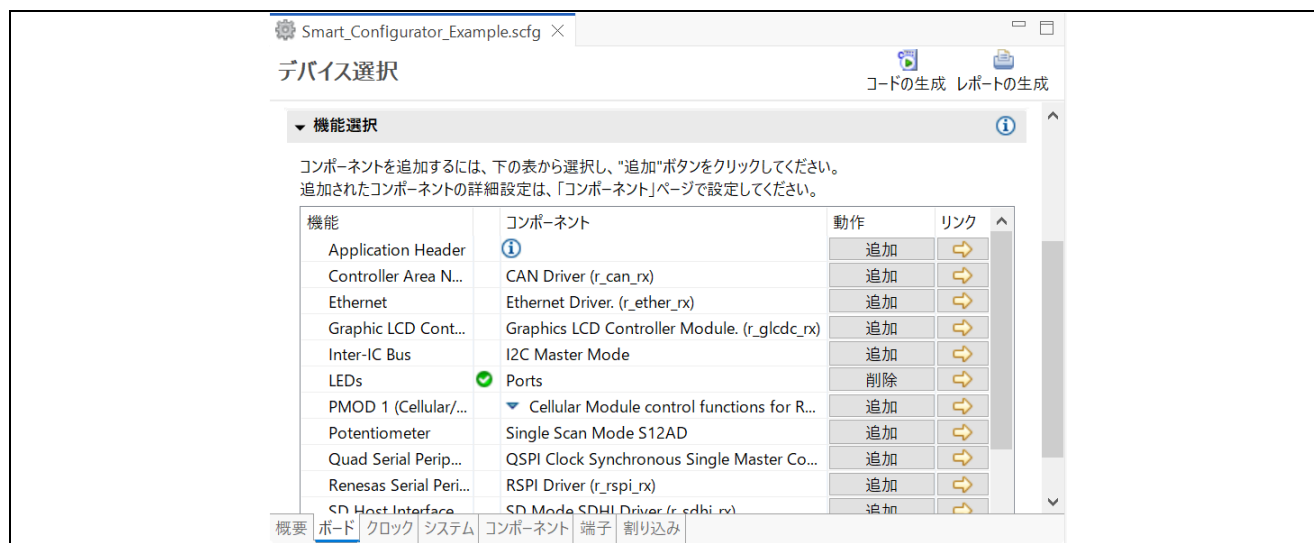


図 3-12 [スマート・コンフィグレータビュー]ボードページ機能選択

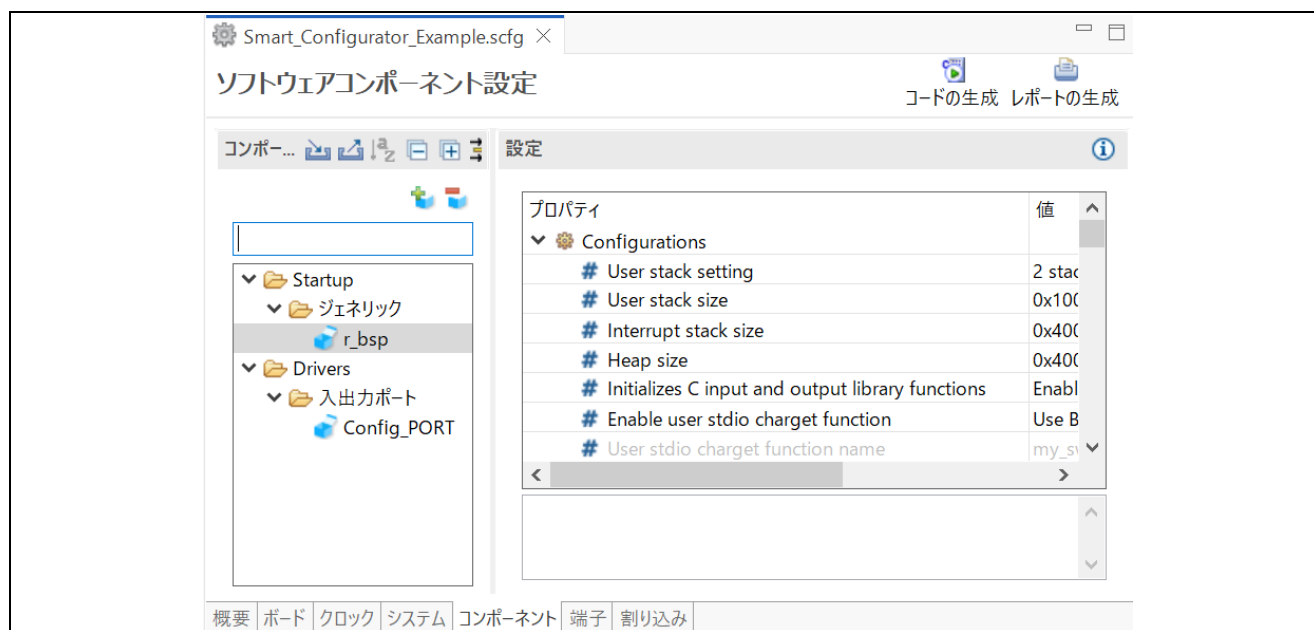


図 3-13 [スマート・コンフィグレータビュー]コンポーネントページ BSP プロパティ

一方、RX プロジェクトコンフィグレーションエディタビューでは[Stacks Configuration]ページまたは[Components Configuration]ページからコンポーネント(モジュール)を追加できます。Components ページからコンポーネントを追加した場合でも[Stacks Configuration]ページの設定は必要です。従って通常は[Stacks Configuration]ページからコンポーネントを追加してください。コンポーネントの詳細設定はプロパティビューで設定することができます。BSP の詳細設定を行う場合には BSP ページを選択し、プロパティビューで詳細項目を設定してください。BSP 以外のコンポーネントは[Stacks Configuration]ページで設定したいコンポーネントのモジュールを選択し、プロパティビューで詳細項目を設定してください。

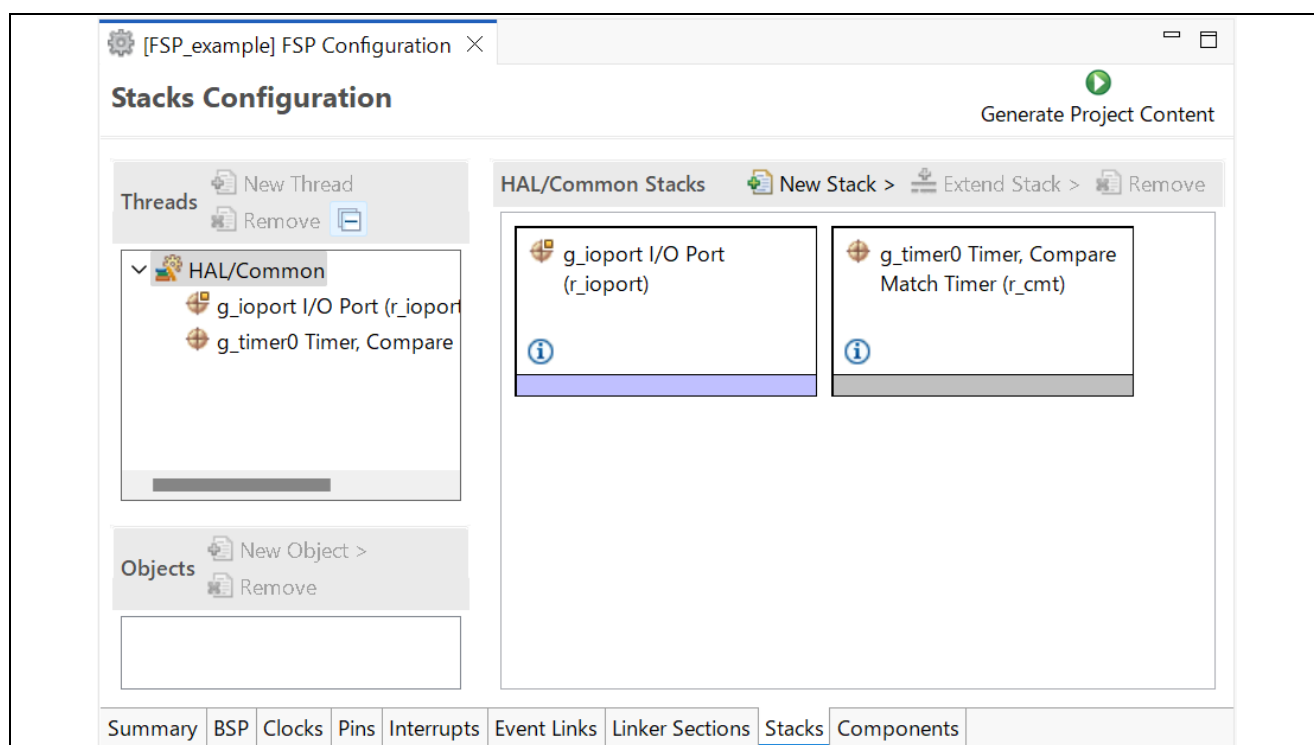


図 3-14 [Stacks Configuration]ページ

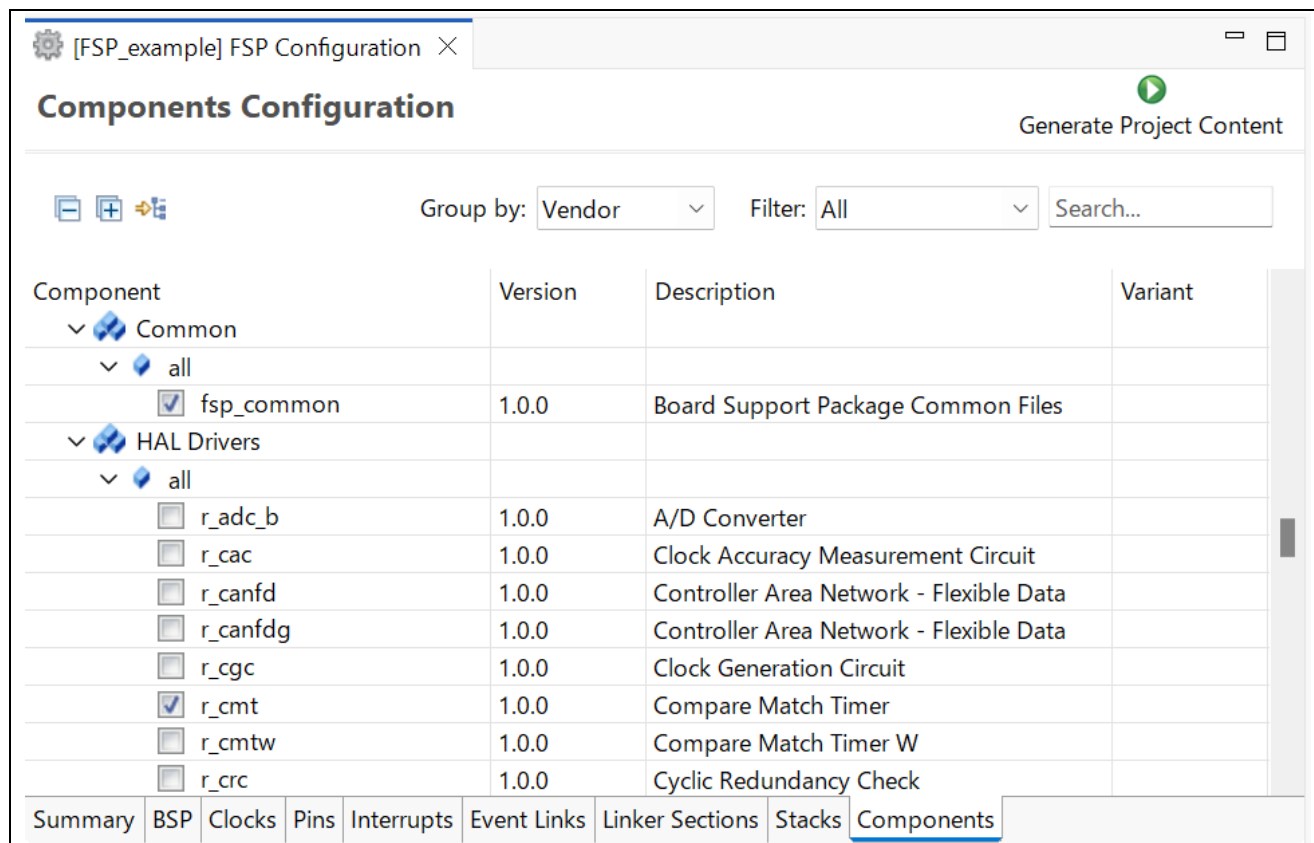


図 3-15 [Components Configuration]ページ

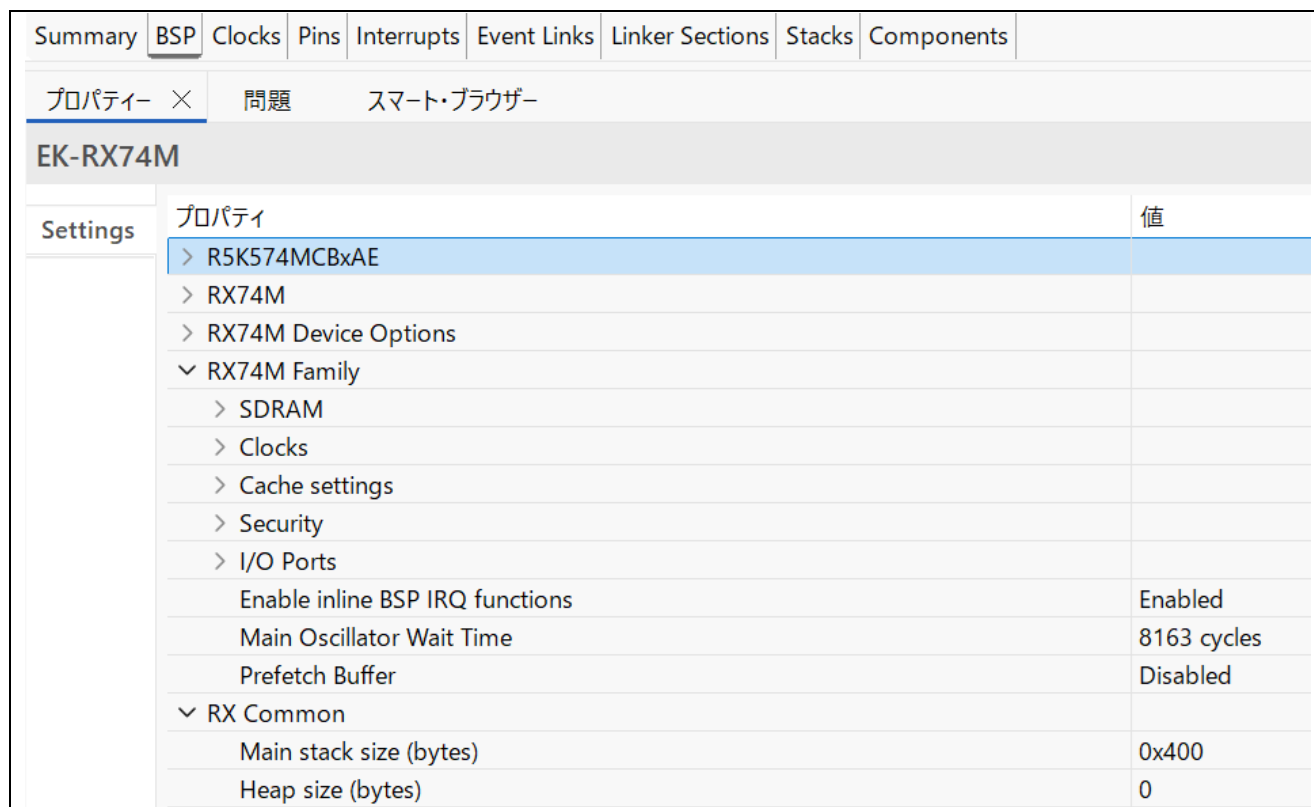


図 3-16 [プロパティビュー]BSP ページ選択時

3.1.6 端子設定

スマート・コンフィグレータビューでは端子ページもしくは MCU/MPU パッケージビューで端子機能の割り当てについて設定することができます。

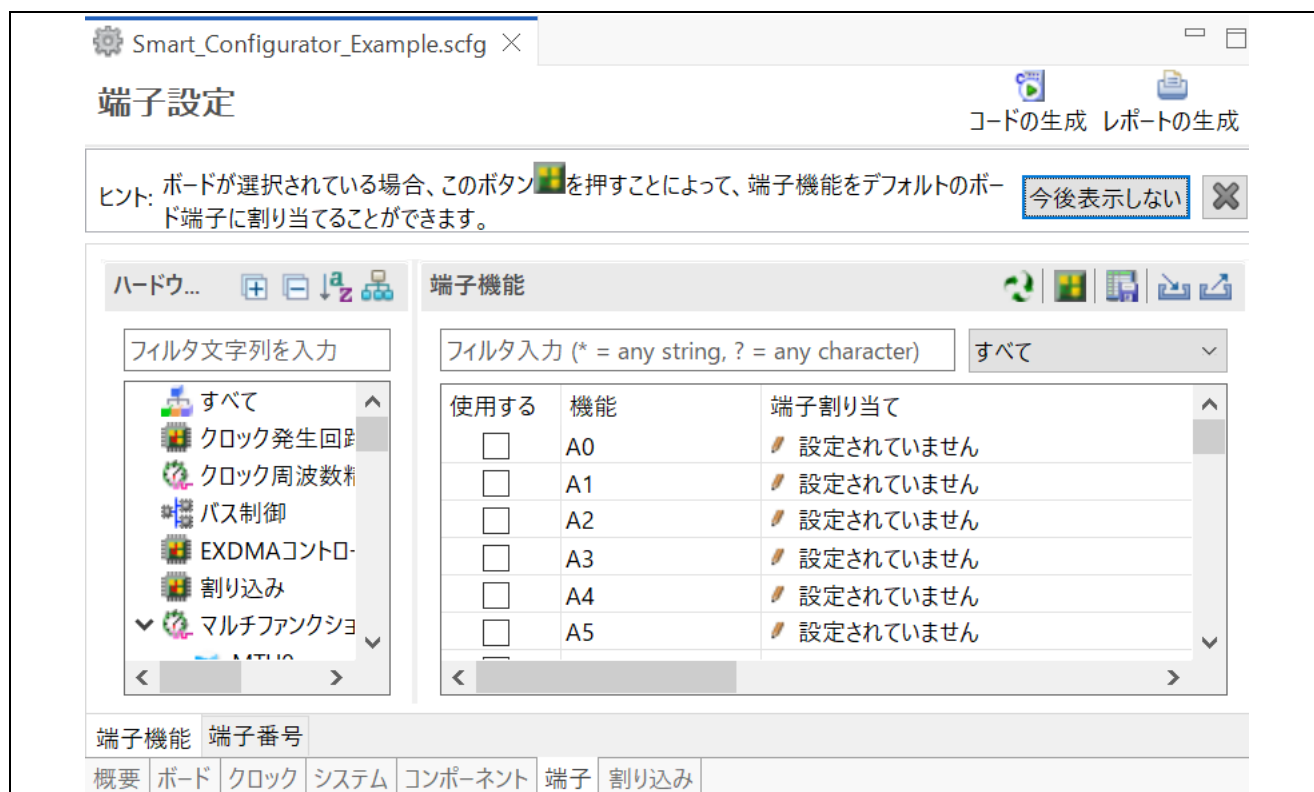


図 3-17 [スマート・コンフィグレータビュー]端子ページ

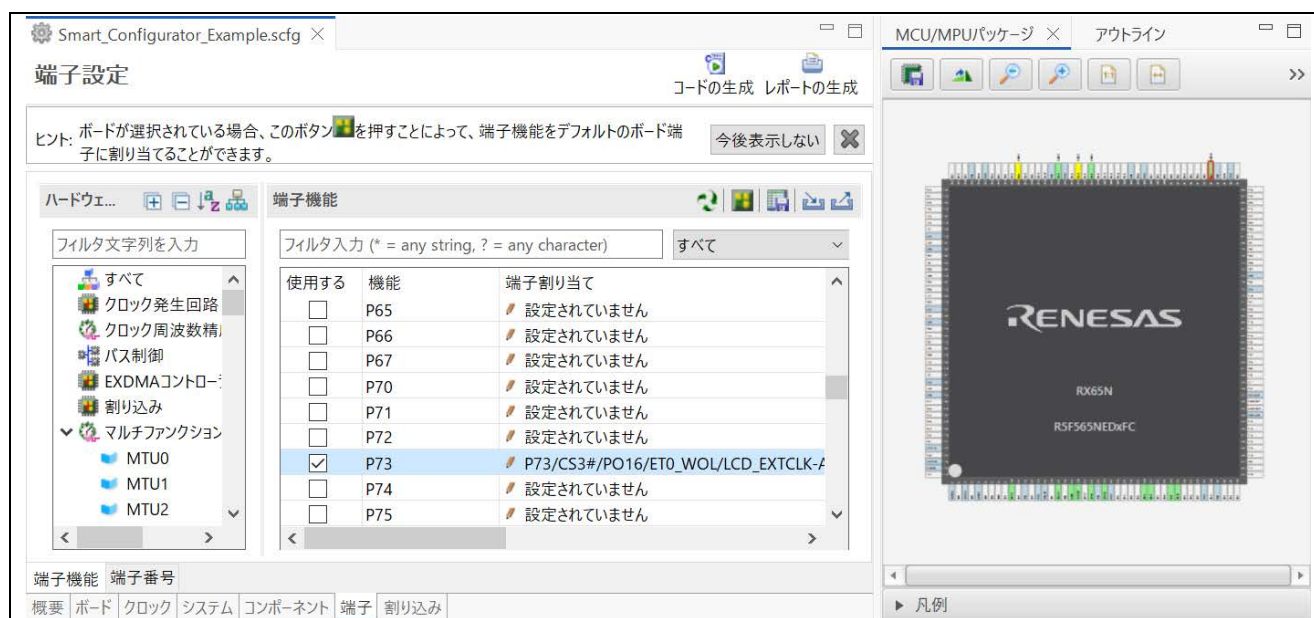


図 3-18 [スマート・コンフィグレータビュー]端子ページと[MCU/MPU パッケージビュー]

一方、RX プロジェクトコンフィグレーションエディタビューでは[Pin Configuration]ページもしくは Visualization ビューから端子機能の割り当ての設定が行えます。

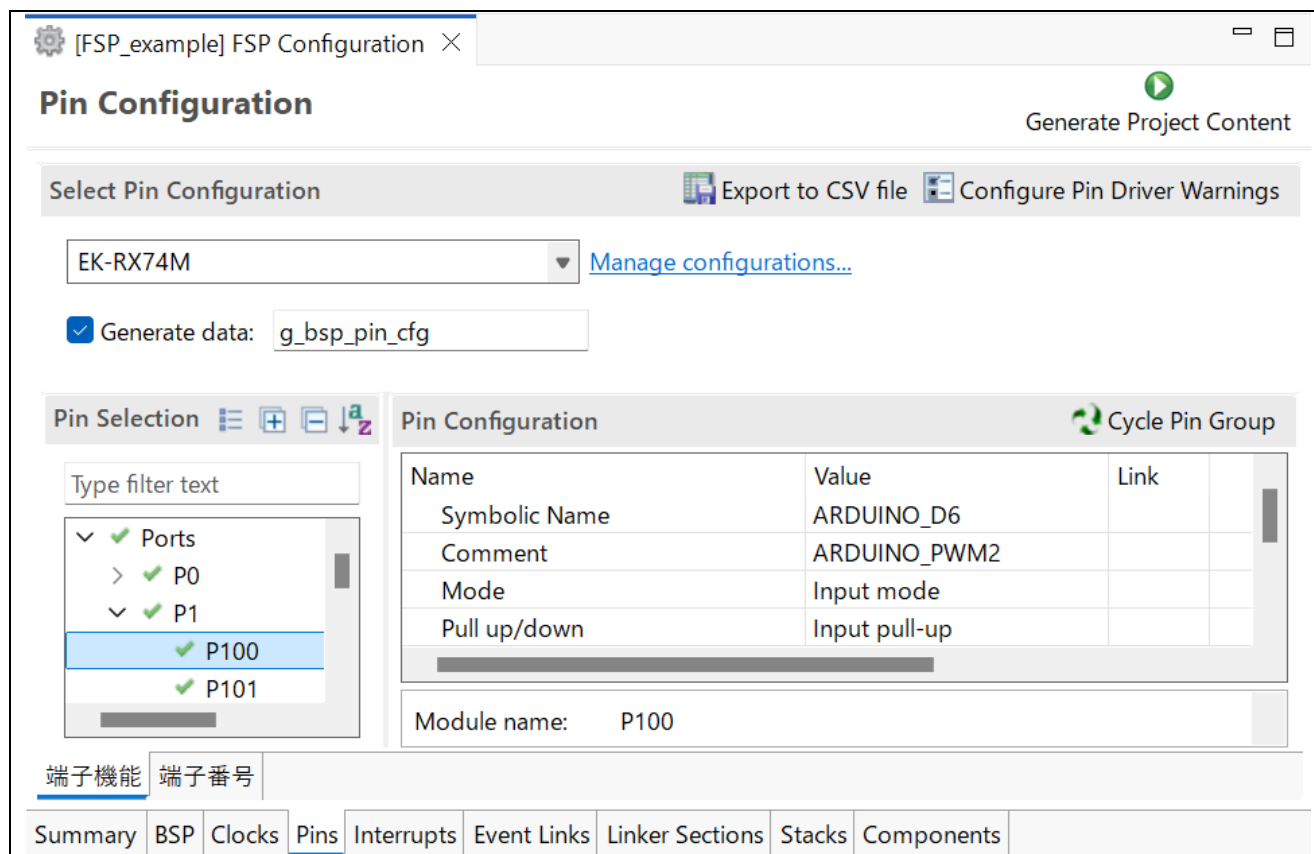


図 3-19 [Pin Configuration]ページ

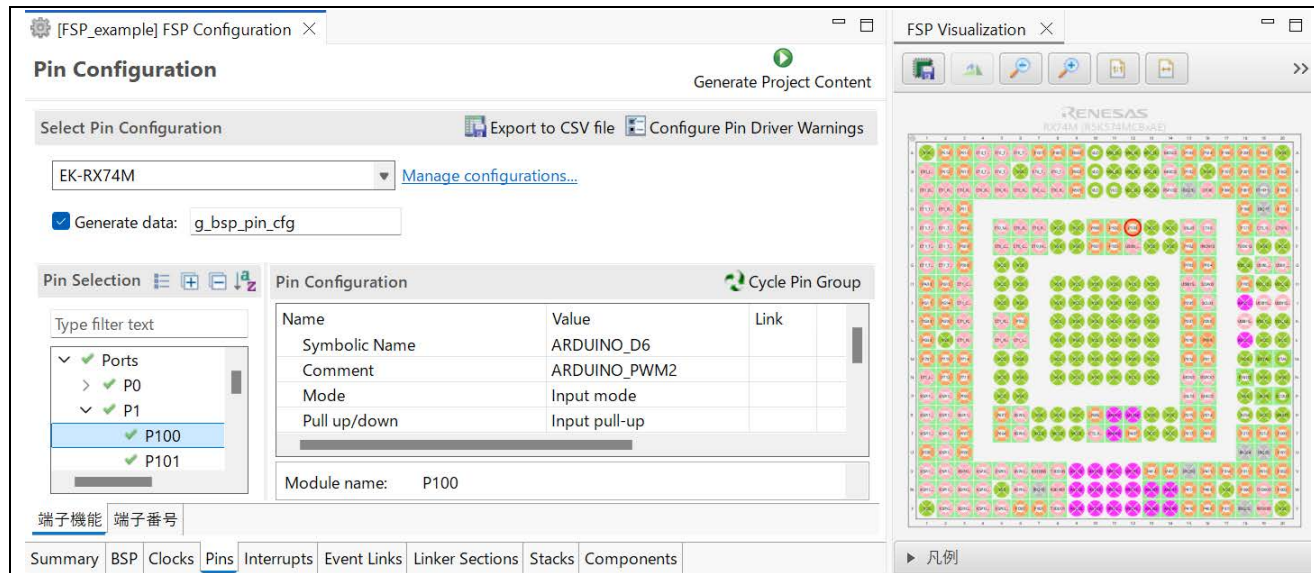


図 3-20 [Pin Configuration]ページと[Visualization]ビュー

3.1.7 割り込み設定

スマート・コンフィグレータビューでは割り込みページで割り込みについて設定することができます。



図 3-21 [スマート・コンフィグレータビュー]割り込みページ

一方、RX プロジェクトコンフィグレーションエディタビューでは[Interrupts Configuration]ページから割り込みの設定が行えます。

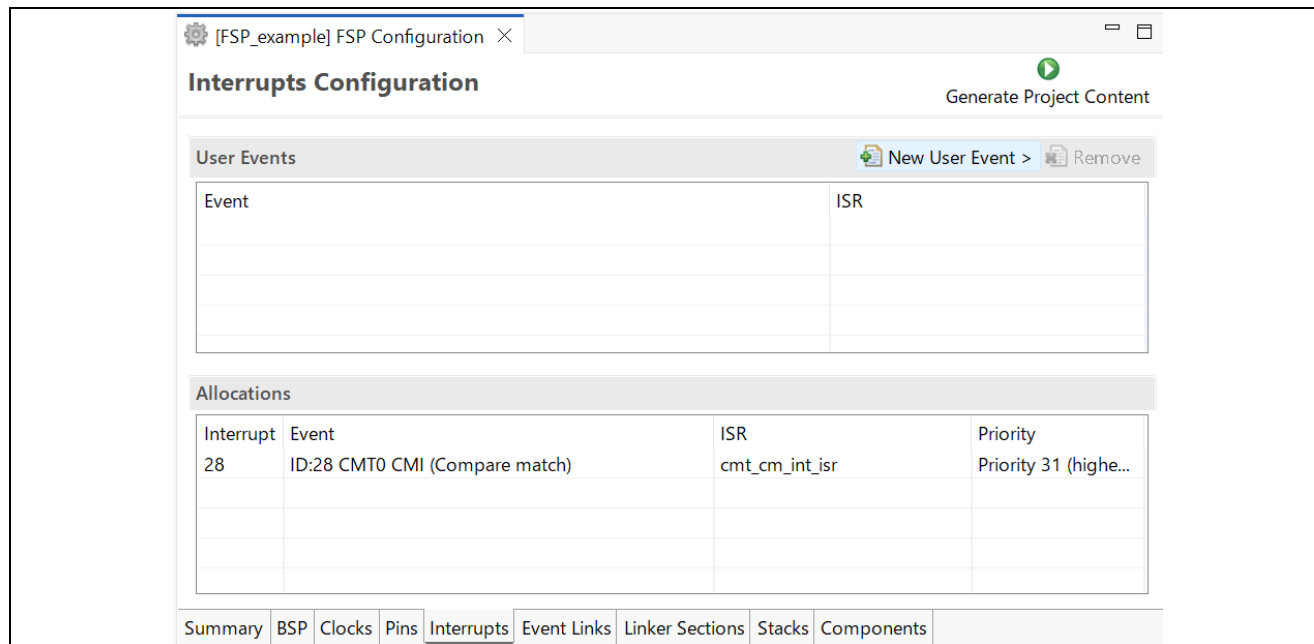


図 3-22 [Interrupts Configuration]ページ

3.1.8 イベントリンク設定

スマート・コンフィグレータビューではイベントリンク機能の設定はコンポーネントページでコード生成タイプのイベントリンクコントローラもしくは Firmware Integration Technology タイプの ELC モジュールを追加してプロパティから設定することができます。

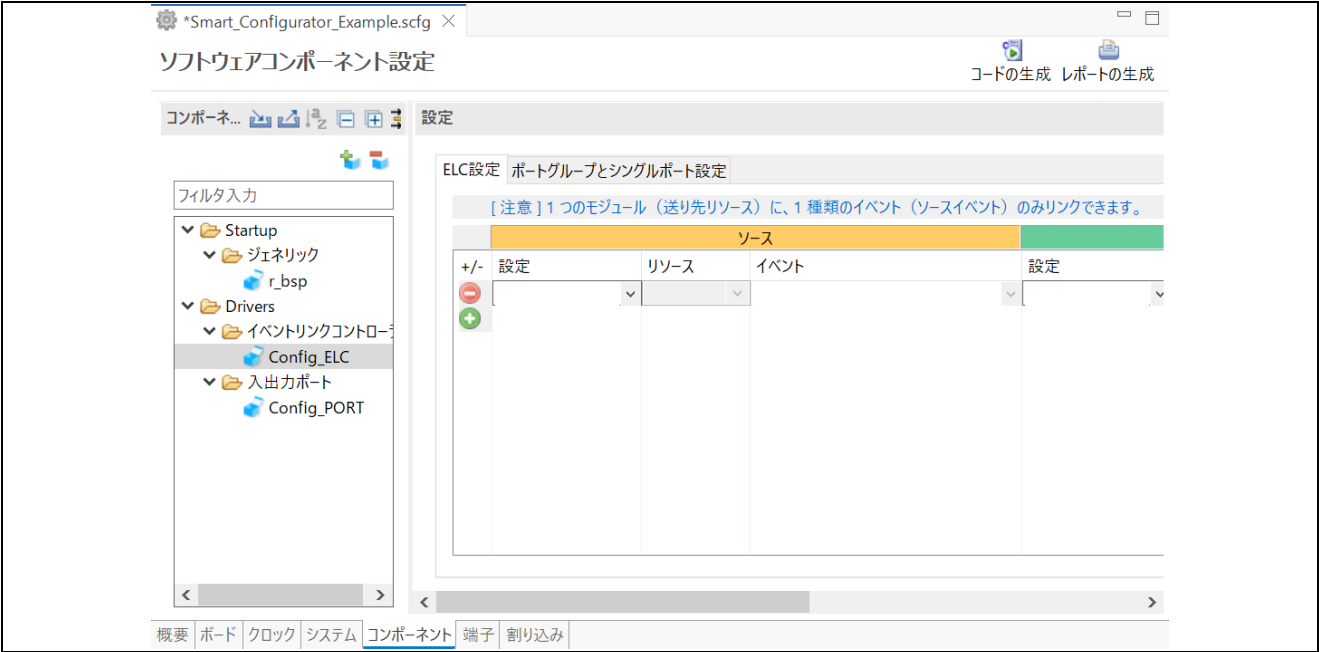


図 3-23 [スマート・コンフィグレータビュー]コンポーネントページ Config_ELC

一方、RX プロジェクトコンフィグレーションエディタビューでは[Event Links Configuration]ページからイベントリンクの設定が行えます。

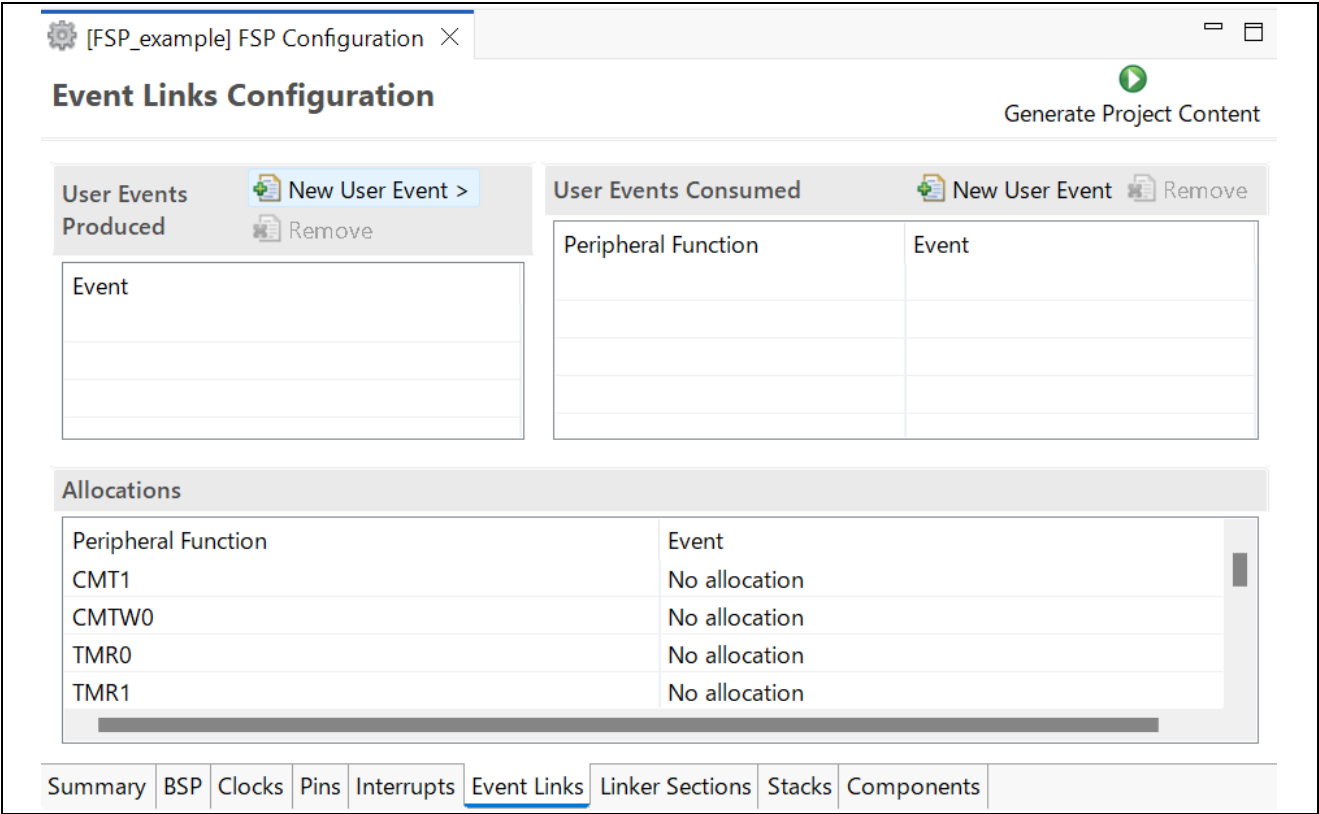


図 3-24 [Event Links Configuration]ページ

4. ソフトウェアの仕様比較

4.1 スマート・コンフィグレータによるコード生成/FIT モジュール/FSP モジュール対応機能比較

4.1.1 周辺機能の対応機能比較

周辺機能別のコード生成または FIT モジュールに対応する FSP モジュールの有無を以下に示します。

表 4.1 周辺機能別の対応比較一覧

機能名	コード生成	FIT モジュール	FSP モジュール
基本設定	—(r_bsp を使用)	r_bsp	bsp fsp_common
バッテリーバックアップ機能(VBATT)	—	r_vbatt_rx	(bsp に含む)
電圧検出回路(LVD)	←と同名	r_lvd_rx	r_pvd
消費電力低減機能(LPC)	←と同名	r_lpc_rx	r_lpm
割り込みコントローラ(IRQ)	←と同名	r_irq_rx	r_icu
データトランスファコントローラ(DTC)	←と同名	r_dtc_rx	r_dtc
DMA コントローラ(DMAC)	←と同名	r_dmaca_rx	r_dmac
I/O ポート	ポート	r_gpio_rx	r_ioport
マルチファンクションピンコントローラ(MPC)	—	r_mpc_rx	(Pins に含む)
コンペアマッチタイマ(CMT)	←と同名	r_cmt_rx	r_cmt
コンペアマッチタイマ W (CMTW)	コンペアマッチタイマ	r_cmtw_rx	r_cmtw
リアルタイムクロック(RTC)	←と同名	r_rtc_rx	r_rtc
ローパワータイマ(LPT)	←と同名	r_lpt_rx	—(注)
独立ウォッチドッグタイマ(IWDT)	ウォッチドッグタイマ	r_iwdt_rx	r_iwdt
ウォッチドッグタイマ(WDTA)	←と同名	r_wdt_rx	r_wdt
8 ビットタイマ(TMR)	←と同名	—	r_tmr
ポートアウトプットイネーブル(POE)	←と同名	—	—(注)
GPTW 用ポートアウトプットイネーブル(POEG)	ポートアウトプットイネーブル	—	r_poeg
汎用 PWM タイマ(GPTW)	←と同名	—	r_gpt r_gpt_three_phase
マルチファンクションタイマパルスユニット(MTU)	ノーマルモード	—	—(注)
	PWM モードタイマ	—	—(注)
	位相係数モードタイマ	—	—(注)
	相補 PWM モードタイマ	—	—(注)
	モータ	—	—(注)
	デッドタイム補償用カウンタ	—	—(注)
プログラマブルパルスジェネレータ (PPG)	←と同名	—	—(注)
バス	←と同名	—	(bsp に含む)
クロック周波数精度測定回路	←と同名	—	r_cac
コンパレータ	←と同名	—	—(注)
CRC 演算器(CRC)	←と同名	—	r_crc
データ演算回路	←と同名	—	r_doc

【注】 今後サポートを予定しています。

表 4.2 周辺機能別の対応比較一覧(2)

機能名	コード生成	FIT モジュール	FSP モジュール
シリアルコミュニケーションインタフェース(SCI)	SCI/SCIF 調歩同期式モード	r_sci_rx, r_scif_rx	—(注)
	SCI/SCIF クロック同期式モード		
	スマートカードインタフェース	—	—(注)
	I2C マスタモード	r_sci_iic_rx	—(注)
	SPI クロック同期式モード(3 線式)	—	—(注)
シリアルコミュニケーションインタフェース(RSCI)	SCI/SCIF 調歩同期式モード	r_rsci_rx	r_sci_b_uart
	SCI/SCIF クロック同期式モード		—(注)
	スマートカードインタフェース	—	—(注)
	I2C マスタモード	r_rsci_iic_rx	r_sci_b_i2c
	SPI クロック同期式モード(3 線式)	—	r_sci_b_spi
I ² C バスインタフェース(RIIC)	I2C マスタモード	r_riic_rx	—(注)
	I2C スレーブモード		—(注)
ハイスピード I ² C バスインタフェース(RIICHS)	—	r_riichs_rx	r_iic_b_master r_iic_b_slave
I3C バスインタフェース(RI3C)	—	r_i3c_rx	r_i3c
シリアルペリフェラルインタフェース(RSPI)	SPI クロック同期式モード(3 線式)	r_rspx_rx	—(注)
	SPI 動作モード(4 線式)		
シリアルペリフェラルインタフェース(RSPIA)	—	r_rspia_rx	r_spi_b
クワッドシリアルペリフェラルインタフェース(QSPI)	—	r_qspix_smstr_rx	—(注)
クワッド SPI メモリインタフェース(QSPIX)	—	r_qspix_rx	—(注)
USB2.0FS ホスト/ ファンクションモジュール(USB)	—	r_usb_basic	r_usb_basic
		r_usb_basic_mini	—(注)
		r_usb_hmsc	r_usb_hmsc
		r_usb_hmsc_mini	—(注)
		r_usb_hcdc	r_usb_hcdc
		r_usb_hcdc_mini	—(注)
		r_usb_hhid	r_usb_hhid
		r_usb_hhid_mini	—(注)
		r_usb_pmssc	r_usb_pmssc
		r_usb_pmssc_mini	—(注)
		r_usb_pcdc	r_usb_pcdc
		r_usb_pcdc_mini	—(注)
		r_usb_phid	r_usb_phid
		r_usb_phid_mini	—(注)

【注】 今後サポートを予定しています。

表 4.3 周辺機能別の対応比較一覧(3)

機能名	コード生成	FIT モジュール	FSP モジュール
イーサネットコントローラ用 PTP コントローラ(EPTPC)	—	r_ptp_rx	r_gptp
EPTPC Light モジュール (EPTPC Light:イーサネット機能拡張デバイスドライバ)	—	r_ptp_light_rx	—(注)
EtherCAT スレーブコントローラ(ESC)	—	r_ecat_rx	r_ethercat_phy
イーサネットコントローラ(ETHERC)	—	r_ether_rx	r_rmac r_rmac_phy
コントローラエリアネットワーク(CAN)	—	r_can_rx	—
コントローラエリアネットワーク(CANFD)	—	r_canfd_rx	r_canfd r_canfd
コントローラエリアネットワーク(RSCAN)	—	r_rscan_rx	—(注)
IrDA インタフェース	—	r_irda_sci_rx	—(注)
パラレルデータキャプチャユニット(PDC)	—	r_pdc_rx	r_pdc
SD ホストインタフェース(SDHI)	—	r_sdhi_rx	r_sdhi
SD スレーブインタフェース(SDSI)	—	r_sdsi_rx	—(注)
マルチメディアカードインタフェース(MMCIF)	—	r_mmcif_rx	—(注)
Δ - Σ モジュレータインタフェース(DSMIF)	←と同名	r_dsmif_rx	r_dsmif
リモコン信号受信機能	←と同名	—	—(注)
12 ビット A/D コンバータ(S12AD)	シングルスキャンモード S12AD	r_s12ad_rx	—(注)
	連続スキャンモード		
	グループスキャンモード S12AD		
12 ビット D/A コンバータ(R12DA)	D/A コンバータ	r_dac_rx	r_dac
アナログフロントエンド(AFE)	←と同名	—	—(注)
サンプリングレートコンバータ(SRC)	—	r_src_api_rx	—(注)
シリアルサウンドインタフェース(SSIE)	—	r_ssi_api_rx	r_ssi
フラッシュメモリ(FLASH)	—	r_flash_rx	—(注)
Trusted Secure IP(TSIP)	—	r_tsip_rx	—(注)
Renesas Secure IP(RSIP)	—	r_rsip_protected_rx	r_rsip_protected
		r_rsip_cm_rx	—(注)
LCD コントローラ/ドライバ(LCDC)	LCD コントローラ	r_lcdc_rx	—(注)
グラフィック LCD コントローラ(GLCDC)	—	r_glcdd_rx	—(注)
ユニーク ID リード	—	r_uid_rx	—(注)
Byte Queue Buffer(データ管理)	—	r_byteq	—(注)
Long Queue Buffer(データ管理)	—	r_longq	—(注)
静電容量式タッチセンサ(CTSU)	—	r_ctsu_qe	—(注)
イベントリンクコントローラ(ELC)	←と同名	r_elc_rx	r_elc
RX DRW2D ドライバ	—	r_drw2d_rx	—(注)

【注】 今後サポートを予定しています。

4.1.2 ミドルウェアの対応機能比較

ミドルウェアの FIT モジュール、FSP モジュールの有無を以下に示します。

表 4.4 ミドルウェアの対応比較一覧(1)

機能分類		FIT モジュール名称	FIT モジュール	FSP モジュール
通信	TCP/IP	lwIP (lightweight IP) Driver	r_lwip_driver_rx	rm_lwip_ether
		lwIP (lightweight IP)	r_lwip_rx	rm_lwip_sys_baremetal rm_lwip_sys_freertos
	Wi-Fi	US159-DA16XXXMEVZ Wi-Fi Control Module	r_wifi_da16xxx	—(注)
	BLE	BLE モジュール (BLE)	r_ble_rx23w	—(注)
		QE Utility モジュール (Profiles)	r_ble_qe_utility	—(注)
		メッシュモジュール (Mesh)	r_mesh_rx23w	—(注)
		US159-DA14531EVZ BLE Control Module	r_ble_da1453x_r x	—(注)
	Cellular	RYZ014A/GM01Q Cellular モジュール制御モジュール	r_cellular	—(注)
	NFC	PTX NFC 制御モジュール	r_nfc_ptx_rx	—(注)
ファイルシ ステム デバイスド ライバ	SD メモリカー ド	SD メモリカードドライバ モジュール	r_sdc_sdmem_r x	—(注)
		SPI モード SD メモリカー ドドライバモジュール	r_sdc_spi_rx	—(注)
	データフラッ シュドライバ	フラッシュメモリデータ 管理モジュール	r_datfrx_rx	—(注)
	シリアル NOR フラッシュ	シリアル NOR フラッシュ メモリアクセス クロック 同期制御モジュール	r_flash_spi	—(注)
	SPI シリアル EEPROM ドラ イバ	EEPROM アクセス ク ロック同期式制御モ ジュール	r_eeprom_spi	—(注)
	メモリアクセ スドライバ インタフェース	メモリアクセス用ドライ バインタフェースモ ジュール	r_memdrv_rx	—(注)
画像処理	JPEG	JPEG デコーダモジュール	r_jpegd_rx	—(注)
		JPEG エンコーダモジュール	r_jpege_rx	—(注)
グラフィッ クス		Aeropoint モジュール	r_aeropoint_rx	—(注)
		emWin モジュール	r_emwin_rx	—(注)
		QE for Display モジュール	r_qe_display_rx	—(注)

【注】 今後サポートを予定しています。

表 4.5 ミドルウェアの対応比較一覧(2)

機能分類		FIT モジュール名称	FIT モジュール	FSP モジュール
Reality AI	Reality AI UART 通信モ ジュール	Reality AI UART Communication Module	rm_comms_uart_rx	—(注)
	Reality AI デー タコレクター 制御モジュー ル	Reality AI Data Collector Control Module	rm_rai_data_collecto r_rx	—(注)
	Reality AI デー タシップパー制 御モジュール	Reality AI Data Shipper Control Module	rm_rai_data_shipper _rx	—(注)
	Reality AI USB PCDC 通信モ ジュール	Reality AI USB PCDC Communication Module	rm_comms_usb_pcd c_rx	rm_comms_usb_pcdc
ソフトウェア ライブラ リ	音声	音声録音・再生シス テム(独自 ADPCM コー デック) M3S-S2-Tiny モジュール	r_s2_rx	—(注)
	オープンソー ス FAT ファイ ルシステム	FAT ファイルシステム M3S-TFAT-Tiny モ ジュール	r_tfat_rx	rm_freertos_plus_fat
		M3S-TFAT-Tiny メモ リドライバインタ フェースモジュール	r_tfat_driver_rx	—(注)
ファーム ウェア更新		ファームウェアアップ デートモジュール	r_fwup	—(注)
MCUboot		MCUboot モジュール	rm_mcuboot	rm_mcuboot_port
ファーム ウェア更新 通信		ファームウェアアップ デート通信モジュール	r_fwupcomm	—(注)
センサー	タッチセン サーモジュー ル	QE Touch モジュール	rm_touch_qe	—(注)

【注】 今後サポートを予定しています。

4.2 初期設定の仕様比較

4.2.1 FIT モジュールと FSP モジュールの BSP の仕様比較

FIT BSP モジュールと FSP BSP モジュールの初期設定の差異について以下に示します。FIT の BSP の初期設定は[ボードサポートパッケージモジュール Firmware Integration Technology](#) の 2.2 初期設定をご参照ください。

表 4.6 初期設定の仕様比較(1)

No	FIT BSP モジュールの初期設定項目	FSP BSP モジュールにおける 当該処理と同処理の有無	FSP に無い場合 の対応方法
1	スタックアドレスを設定	有	
2	割り込みベクタテーブルの先頭アドレスを設定	有	
3	例外ベクタテーブルの先頭アドレスを設定 (【注】1)	有	
4	FPU の初期化 (【注】1) R_BSP_SET_FPSW()	無 (【注】3)	4.2.2.1 参照
5	倍精度浮動小数点コプロセッサの初期化 (【注】1) R_BSP_SET_DPSW()	無 (【注】3)	4.2.2.1 参照
6	三角関数演算器の初期化 (【注】1) R_BSP_INIT_TFU() / bsp_tfu_init()	無 (【注】3)	4.2.2.2 参照
7	VBATT 電圧安定待ち (【注】1) vbatt_voltage_stability_wait()	無 (【注】3)	4.2.2.3 参照
8	システムクロックを設定 mcu_clock_setup()	有	
	クロック分周比および逡倍率の設定	有	
	クロックソースとして使用しないクロックの停止	一部無 (HOCO および LOCO を停止する処理は未実装。)	4.2.2.4 参照
	クロックソースとして使用するクロックの発振安定待ち	有	
	システムクロックの切り替え	有	
9	ウォームスタート関数(PRE) (【注】2) User_Warm_start_func_pre()	有	
10	C ランタイム環境を初期化 _INIT_SCT()	有	
	未初期化データ領域のゼロ設定	有	
	初期化データ領域にデータをコピー	有	
11	BSP の変数を初期化 bsp_ram_initialize()	有	
12	ウォームスタート関数(POST) (【注】2) User_Warm_start_func_post()	有	
13	STDIO ライブラリを初期化 (【注】2) init_iolib()	無 (【注】3)	4.2.2.5 参照
14	割り込みコールバックを初期化 bsp_interrupt_open()	有	
	割り込みコールバックの初期化	有 (グループ割り込みのみ)	
	選択型割り込みの割り込み要因割り付け (【注】1)	有 ([Interrupts Configuration] ページで設定したもののみ)	

【注】1. MCU によって、この手順はスキップされます。

【注】2. r_bsp_config.h の設定によって、動作は異なります。

【注】3. v1.0.0 時点での仕様になります。

表 4.7 初期設定の仕様比較(2)

No	FIT BSP モジュールの初期設定項目	FSP BSP モジュールにおける当該処理と同処理の有無	FSP に無い場合の対応方法
15	レジスタ保護の初期化 bsp_register_protect_open()	有	
16	バス優先度の初期化 ^{〔注〕2)} bsp_bus_priority_initialize()	無 ^{〔注〕4)}	4.2.2.6 参照
17	ハードウェアと端子の初期化 hardware_setup()	有	
	周辺機能に使用する端子を許可	有 ^{〔[Pin Configuration]ページで設定したもののみ〕}	
	存在しないポートの初期化 bsp_non_existent_port_init()	有	
18	PSW の初期化 R_BSP_SET_PSW()	有	
	ユーザスタック使用の許可・禁止 ^{〔注〕2)}	無 ^{〔注〕4)}	4.2.2.7 参照
	CPU の割り込み受け付けを許可 ^{〔注〕3)}	有	
19	ユーザモードへ遷移 ^{〔注〕2)} R_BSP_CHG_PMUSR()	無 ^{〔注〕4)}	4.2.2.8 参照
20	main 関数へジャンプ	有	
21	STDIO ライブラリをクローズ ^{〔注〕2)} close_all()	無 ^{〔注〕4)}	4.2.2.9 参照

【注】2. r_bsp_config.h の設定によって、動作は異なります。

【注】3. CPU の割り込み受け付けのみ許可します。周辺割り込みは、個別に許可する必要があります。

【注】4. v1.0.0 時点での仕様になります。

4.2.2 FSP にない場合の対応方法

4.2.2.1 CPU 関連

当該処理が必要な場合は、ご使用のデバイスのユーザーズマニュアルハードウェア編の CPU 章、コンパイラマニュアル、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.8 CPU 関連処理の実装箇所一例

No	ファイル名	実装箇所
1	system.c	SystemInit 関数内 ^{〔注〕}
2	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_START_RESET（クロック初期設定直前）もしくは BSP_WARM_START_POST_CLOCK（クロック初期設定直後）もしくは BSP_WARM_START_POST_C の条件分岐内
3	hal_entry.c	hal_entry 関数内（スタートアップ完了後）

【注】実装後に FSP バージョンを変更し、Generate Project Content を実行した場合、実装内容は保持されません。その場合は、再度実装してください。

4.2.2.2 三角関数演算器

当該処理が必要な場合は、ご使用のデバイスのユーザーズマニュアルハードウェア編の三角関数演算器章、コンパイラマニュアル、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.9 三角関数演算器関連処理の実装箇所一例

No	ファイル名	実装箇所
1	system.c	SystemInit 関数内 ^(注)
2	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_START_RESET (クロック初期設定直前) もしくは BSP_WARM_START_POST_CLOCK (クロック初期設定直後) もしくは BSP_WARM_START_POST_C の条件分岐内
3	hal_entry.c	hal_entry 関数内 (スタートアップ完了後)

【注】 実装後に FSP バージョンを変更し、Generate Project Content を実行した場合、実装内容は保持されません。その場合は、再度実装してください。

4.2.2.3 VBATT 電圧安定待ち

当該処理が必要な場合は、ご使用のデバイスのユーザーズマニュアルハードウェア編のバッテリーバックアップ機能章、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.10 VBATT 電圧安定待ち処理の実装箇所一例

No	ファイル名	実装箇所
1	system.c	SystemInit 関数内の bsp_clock_init 関数 (クロック初期設定関数) 実行前 ^(注)
2	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_START_RESET (クロック初期設定直前) の条件分岐内

【注】 実装後に FSP バージョンを変更し、Generate Project Content を実行した場合、実装内容は保持されません。その場合は、再度実装してください。

4.2.2.4 クロックソースとして使用しないクロックの停止

当該処理が必要な場合は、ご使用のデバイスのユーザーズマニュアルハードウェア編のクロック発生回路章、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.11 発振停止処理の実装箇所一例

No	ファイル名	実装箇所
1	bsp_clocks.c	bsp_clock_init 関数 (クロック初期設定関数) 内 ^(注)
2	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_START_POST_CLOCK (クロック初期設定直後) もしくは BSP_WARM_START_POST_C の条件分岐内
3	hal_entry.c	hal_entry 関数内 (スタートアップ完了後)

【注】 実装後に FSP バージョンを変更し、Generate Project Content を実行した場合、実装内容は保持されません。その場合は、再度実装してください。

4.2.2.5 STDIO ライブラリの初期化

当該処理が必要な場合は、ご使用のコンパイラマニュアル、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.12 STDIO ライブラリ初期化処理の実装箇所一例

No	ファイル名	実装箇所
1	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_START_POST_C の条件分岐内
2	hal_entry.c	hal_entry 関数内（スタートアップ完了後）

4.2.2.6 バス優先度の初期化

当該処理が必要な場合は、ご使用のデバイスのユーザーズマニュアルハードウェア編のバス章、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.13 バス優先度初期化処理の実装箇所一例

No	ファイル名	実装箇所
1	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_START_POST_C の条件分岐内
2	hal_entry.c	hal_entry 関数内（スタートアップ完了後）

4.2.2.7 ユーザスタック使用の許可・禁止

当該処理が必要な場合は、ご使用のデバイスのユーザーズマニュアルハードウェア編の CPU 章、コンパイラマニュアル、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.14 ユーザスタック使用の許可・禁止処理の実装箇所一例

No	ファイル名	実装箇所
1	hal_warmstart.c	R_BSP_WarmStart 関数の BSP_WARM_START_POST_C の条件分岐内
2	hal_entry.c	hal_entry 関数内（スタートアップ完了後）

4.2.2.8 ユーザモードへの遷移

当該処理が必要な場合は、ご使用のデバイスのユーザーズマニュアルハードウェア編の CPU 章、コンパイラマニュアル、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.15 ユーザモードへの遷移処理の実装箇所一例

No	ファイル名	実装箇所
1	hal_entry.c	hal_entry 関数内（スタートアップ完了後）

4.2.2.9 STDIO ライブラリのクローズ

当該処理が必要な場合は、ご使用のコンパイラマニュアル、FIT の BSP モジュールのソースコードを参照し、以下実装箇所一例などにお客様自身で実装してください。

表 4.16 STDIO ライブラリのクローズ処理の実装箇所一例

No	ファイル名	実装箇所
1	hal_entry.c	hal_entry 関数内（スタートアップ完了後）

4.2.3 FSP BSP の初期設定

FSP の BSP では Reset_Handler 関数を MCU のリセットベクタとして設定します。Reset_Handler 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の初期設定として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システムクロック設定の動作をフローチャートで示します。下記のフローチャートは RX FSP v1.0.0 の内容に基づいて作成されています。

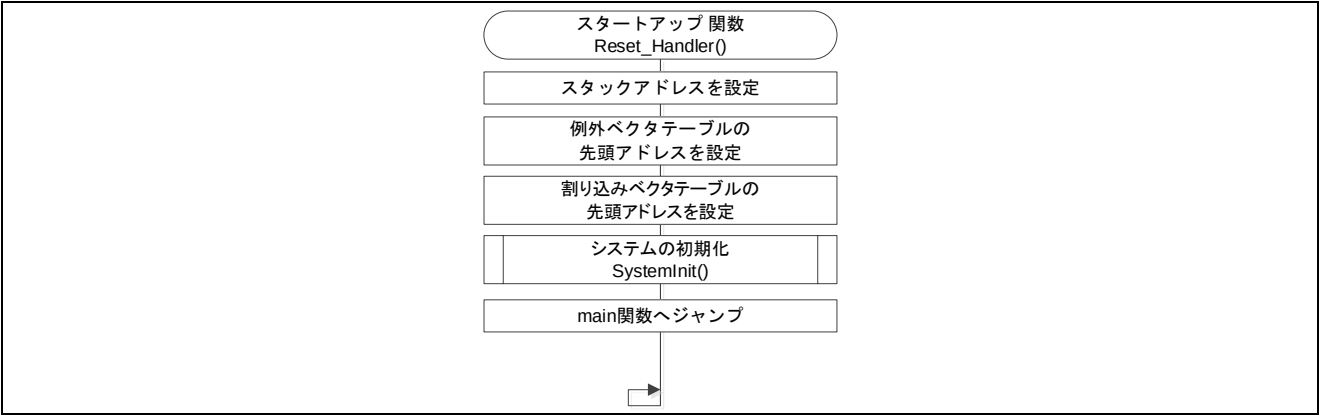


図 4-1 スタートアップ関数のフローチャート

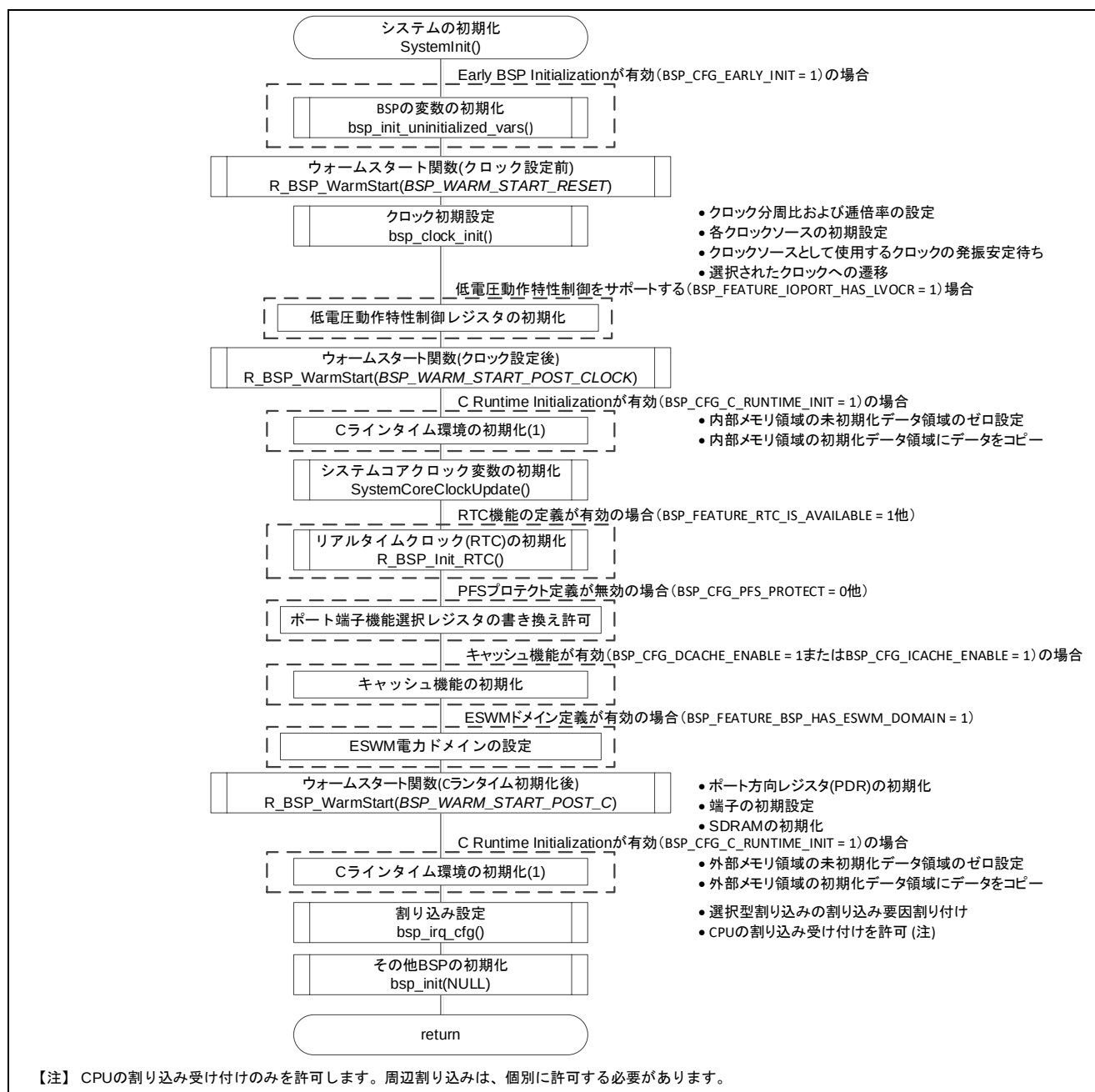


図 4-2 システムの初期化のフローチャート

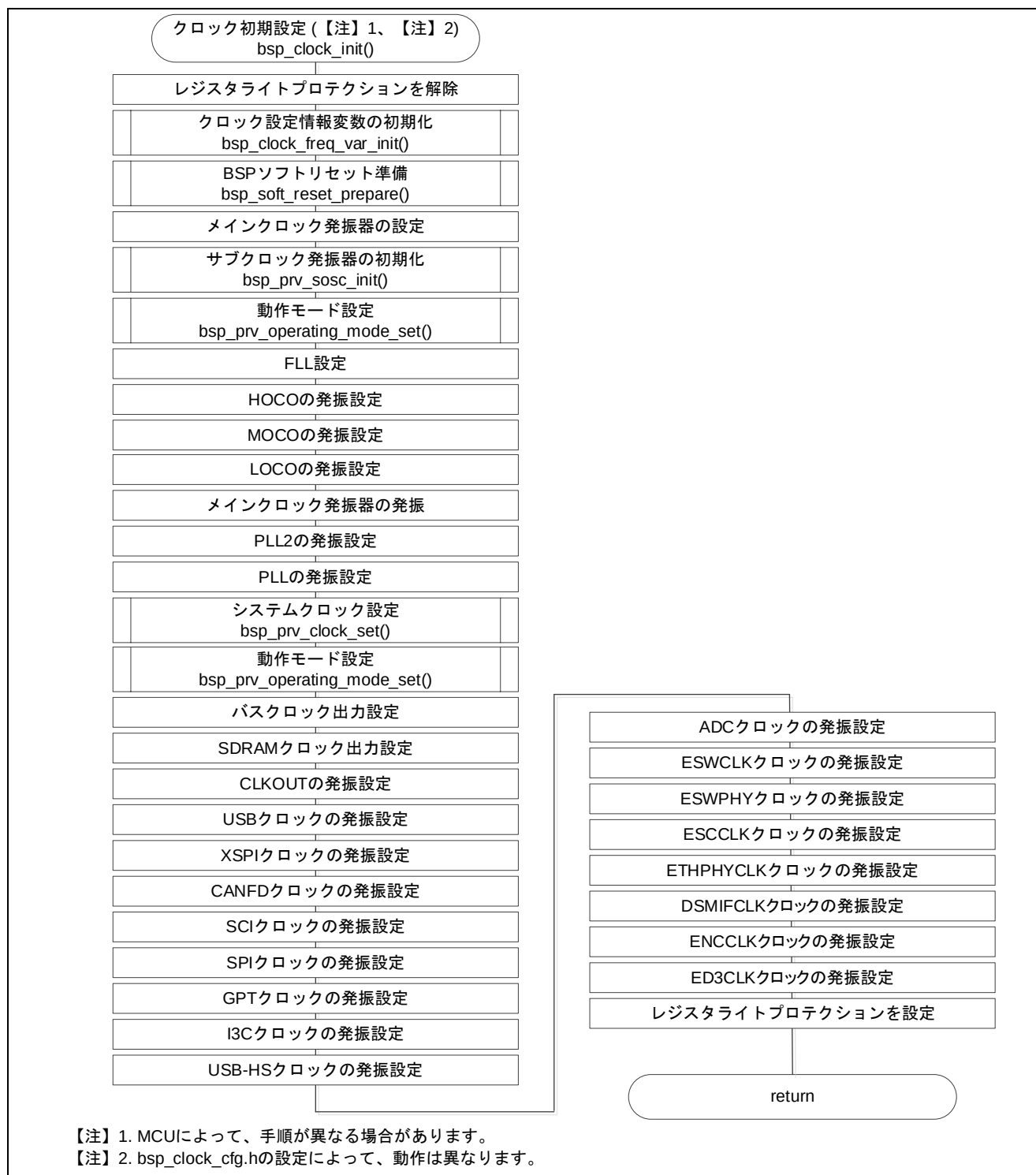


図 4-3 クロック初期設定のフローチャート

4.3 各モジュールの仕様比較

FIT モジュール、コード生成、FSP モジュールの API には互換性がありません。

以下にいくつかのモジュールの API 仕様について説明します。

4.3.1 コンペアマッチタイマ(CMT)

4.3.1.1 CMT FSP モジュール

表 4.17 CMT FSP モジュールの API

No	関数定義	機能概要
1	fsp_err_t R_CMT_Open(timer_ctrl_t * const p_ctrl, timer_cfg_t const * const p_cfg)	CMT タイマモジュールを初期化します。
		制御情報へのポインタ
		設定情報へのポインタ
2	fsp_err_t R_CMT_Stop(timer_ctrl_t * const p_ctrl)	タイマを停止します。
		制御情報へのポインタ
3	fsp_err_t R_CMT_Start(timer_ctrl_t * const p_ctrl)	タイマを開始します。
		制御情報へのポインタ
4	fsp_err_t R_CMT_Reset(timer_ctrl_t * const p_ctrl)	カウンタ値を 0 にリセットします。
		制御情報へのポインタ
5	fsp_err_t R_CMT_PeriodSet(timer_ctrl_t * const p_ctrl, uint32_t const period_counts)	周期を更新します。新しい周期はすぐに更新され、カウンタは 0 にリセットされます。
		制御情報へのポインタ
		設定する周期カウンタ値
6	fsp_err_t R_CMT_InfoGet(timer_ctrl_t * const p_ctrl, timer_info_t * const p_info)	タイマ情報を取得し、指定されたポインタ p_info に保存します。
		制御情報へのポインタ
		取得情報を格納する変数へのポインタ
7	fsp_err_t R_CMT_StatusGet(timer_ctrl_t * const p_ctrl, timer_status_t * const p_status)	現在のタイマのステータスを取得し、指定されたポインタ p_status に保存します。
		制御情報へのポインタ
		取得ステータスを格納する変数へのポインタ
8	fsp_err_t R_CMT_CounterSet(timer_ctrl_t * const p_ctrl, uint32_t counter)	カウンタ値を設定します。
		制御情報へのポインタ
		設定するカウンタ値
9	fsp_err_t R_CMT_CallbackSet(timer_ctrl_t * const p_api_ctrl, void (* p_callback)(timer_callback_args_t *), void const * const p_context, timer_callback_args_t * const p_callback_memory)	コールバック引数構造体にメモリを提供するオプションを使用して、ユーザコールバックを更新します。
		API 制御情報へのポインタ
		コールバック関数へのポインタ
		コールバック中にユーザに提供されるコンテキスト
		コールバックパラメータ情報へのポインタ
10	fsp_err_t R_CMT_Close(timer_ctrl_t * const p_ctrl)	カウンタを停止し、内部ドライバデータをクリアします。
		制御情報へのポインタ

【注】 RX FSP v1.0.0 で非サポートの API は省略しています。

4.3.1.2 CMT FIT モジュールと FSP モジュールの比較

CMT FIT モジュールと CMT FSP モジュールの API の機能対応表を以下に示します。FIT モジュール API に記載された API は FSP モジュール API に置き換えることができます。CMT FIT モジュールの API の詳細は [CMT モジュール Firmware Integration Technology](#) の 3. API 関数をご参照ください。CMT FSP モジュールの API の詳細は [RX Flexible Software Package Documentation](#) の API Reference > Modules > Timers > [Compare Match Timer \(r_cmt\)](#) をご参照ください。

表 4.18 CMT モジュールの機能対応

No	FIT モジュール API	FSP モジュール API
1	R_CMT_CreatePeriodic()	R_CMT_Open() p_cfg->mode = TIMER_MODE_PERIODIC R_CMT_Start()
2	R_CMT_CreatePeriodicAssignChannelPriority()	R_CMT_Open() p_cfg->mode = TIMER_MODE_PERIODIC p_cfg->cycle_end_ip1 = (x) (x = 0-15) R_CMT_Start()
3	R_CMT_CreateOneShot()	R_CMT_Open() p_cfg->mode = TIMER_MODE_ONE_SHOT R_CMT_Start()
4	R_CMT_CreateOneShotAssignChannelPriority()	R_CMT_Open() p_cfg->mode = TIMER_MODE_ONE_SHOT p_cfg->cycle_end_ip1 = (x) (x = 0-15) R_CMT_Start()
5	R_CMT_Stop()	R_CMT_Stop()
6	R_CMT_Control()	以下一部該当機能あり。
	Command = CMT_RX_CMD_IS_CHANNEL_COUNTING	R_CMT_StatusGet()
	Command = CMT_RX_CMD_PAUSE	R_CMT_Stop()
	Command = CMT_RX_CMD_RESUME	R_CMT_Start()
	Command = CMT_RX_CMD_RESTART	R_CMT_Start()
	Command = CMT_RX_CMD_GET_NUM_CHANNELS	該当機能なし。(注)
	Command = CMT_RX_CMD_SET_PRIORITY	R_CMT_Open() p_cfg->cycle_end_ip1 = (x) (x = 0-15)
	Command = CMT_RX_CMD_GET_PRIORITY	該当機能なし。(注)
7	R_CMT_GetVersion()	モジュール毎のバージョンはなし。 FSP のバージョンは以下の関数で確認 できます。 R_FSP_VersionGet()

【注】 類似機能が必要な場合、FIT モジュールのソースコードを参考にお客様自身で実装してください。

表 4.19 CMT モジュールの機能対応(2)

No	FIT モジュール割り込み関数・コールバック関数	FSP モジュール割り込み関数・コールバック関数
1	cmt_n_isr() (n = チャネル番号)	cmt_cm_int_isr()
2	コールバック関数は以下の各 API 関数の第 2 引数 callback にユーザ定義関数を指定 R_CMT_CreatePeriodic() R_CMT_CreatePeriodicAssignChannelPriority() R_CMT_CreateOneShot() R_CMT_CreateOneShotAssignChannelPriority()	R_CMT_Open() p_cfg->p_callback にユーザ定義関数を指定

4.3.1.3 コード生成 CMT 用 API 関数と FSP モジュールの比較

コード生成コンペアマッチタイマ(CMT)用 API 関数と CMT FSP モジュールの API の機能対応表を以下に示します。コード生成コンペアマッチタイマ(CMT)用 API に記載された API は FSP モジュール API に置き換えることができます。コード生成コンペアマッチタイマ(CMT)用 API 関数の詳細は[スマート・コンフィグレータ ユーザーズマニュアル RX API リファレンス編](#)の 4.2.6 コンペアマッチタイマをご参照ください。CMT FSP モジュールの API の詳細は [RX Flexible Software Package Documentation](#) の API Reference > Modules > Timers > [Compare Match Timer \(r_cmt\)](#) をご参照ください。

表 4.20 コード生成 CMT 用 API 関数の機能対応

No	コード生成コンペアマッチタイマ(CMT)用 API	FSP モジュール API
1	R_<Config_CMT0>_Create()	R_CMT_Open()
2	R_<Config_CMT0>_Start()	R_CMT_Start()
3	R_<Config_CMT0>_Stop()	R_CMT_Stop()
4	R_<Config_CMT0>_UserInit()	該当機能なし。(注)

【注】 スマート・コンフィグレータの API は引数がない関数が多く（GUI で変数を設定できないため）、ユーザプログラムで変数を活用したい場合の初期化処理のために UserInit 関数がありましたが、FSP ではそのような使用方法はないため、該当機能がありません。

表 4.21 コード生成 CMT 用割り込み関数の機能対応

No	コード生成コンペアマッチタイマ(CMT)用割り込み関数	FSP モジュール割り込み関数
1	r_<Config_CMT0>_cmin_interrupt 関数内ユーザ定義処理	cmt_cm_int_isr() コールバック関数で置き換え。 R_CMT_Open() p_cfg->p_callback にユーザ定義関数を指定

4.3.2 シリアルコミュニケーションインタフェース(RSCI)

4.3.2.1 SCI UART FSP モジュール

表 4.22 SCI UART FSP モジュールの API(1)

No	関数定義	機能概要
1	fsp_err_t R_SCI_B_UART_Open(uart_ctrl_t * const p_api_ctrl, uart_cfg_t const * const p_cfg)	入力設定に基づいて UART ドライバを設定します。コンパイル時に受信が有効になっている場合、この関数の最後に受信が有効になります。
		API 制御情報へのポインタ
		設定情報へのポインタ
2	fsp_err_t R_SCI_B_UART_Read(uart_ctrl_t * const p_api_ctrl, uint8_t * const p_dest, uint32_t const bytes)	指定したバイト数を受信先バッファに受信します。
		API 制御情報へのポインタ
		受信バッファへのポインタ 受信バイト数
3	fsp_err_t R_SCI_B_UART_Write(uart_ctrl_t * const p_api_ctrl, uint8_t const * const p_src, uint32_t const bytes)	送信バッファから指定したバイト数を送信します。
		API 制御情報へのポインタ
		送信バッファへのポインタ 送信バイト数
4	fsp_err_t R_SCI_B_UART_BaudSet(uart_ctrl_t * const p_api_ctrl, void const * const p_baud_setting)	Open 関数で選択したクロックを使用してボーレートを更新します。
		API 制御情報へのポインタ
		ボーレート設定情報へのポインタ
5	fsp_err_t R_SCI_B_UART_InfoGet(uart_ctrl_t * const p_api_ctrl, uart_info_t * const p_info)	一度に受信または送信できる最大バイト数などのドライバ情報を取得します。
		API 制御情報へのポインタ
		取得情報を格納する変数へのポインタ
6	fsp_err_t R_SCI_B_UART_Close(uart_ctrl_t * const p_api_ctrl)	進行中の通信をすべて中止します。割り込み、受信機、送信機を無効にします。使用されている場合は、下位レベルの転送ドライバを閉じます。周辺機能の電源を切ります。
		API 制御情報へのポインタ
7	fsp_err_t R_SCI_B_UART_Abort(uart_ctrl_t * const p_api_ctrl, uart_dir_t communication_to_abort)	進行中の転送を中止します。送信では現在の文字が送信された後、送信は中止されます。
		API 制御情報へのポインタ
		中断する通信方向を指定する変数
8	fsp_err_t R_SCI_B_UART_BaudCalculate(uint32_t baudrate, bool bitrate_modulation, uint32_t baud_rate_error_x_1000, sci_b_baud_setting_t * const p_baud_setting)	ボーレートレジスタ設定を計算します。ボーレート関連レジスタに設定される最適な設定を評価して決定します。
		ボーレート (bps)
		ビットレート変調の有効/無効を指定。
		ボーレートエラーの最大値
		計算したボーレート設定情報を格納するためのボーレート設定情報へのポインタ

表 4.23 SCI UART FSP モジュールの API(2)

No	関数定義	機能概要
9	fsp_err_t R_SCI_B_UART_CallbackSet(uart_ctrl_t * const p_api_ctrl, void (* p_callback)(uart_callback_args_t *), void const * const p_context, uart_callback_args_t * const p_callback_memory)	コールバック引数構造体にメモリを提供するオプションを使用して、ユーザコールバックを更新します。
		API 制御情報へのポインタ
		コールバック関数へのポインタ
		コールバック中にユーザに提供されるコンテキスト
		コールバックパラメータ情報へのポインタ
10	fsp_err_t R_SCI_B_UART_ReadStop(uart_ctrl_t * const p_api_ctrl, uint32_t * remaining_bytes)	進行中の読み取りを中止します。
		API 制御情報へのポインタ
		残りバイト数

【注】 RX FSP v1.0.0 で非サポートの API は省略しています。

4.3.2.2 RSCI FIT モジュールと UART FSP モジュールの比較

RSCI FIT モジュールと SCI UART FSP モジュールの API の機能対応表を以下に示します。FIT モジュール API に記載された API は FSP モジュール API に置き換えることができます。RSCI FIT モジュールの API の詳細は [RSCI モジュール Firmware Integration Technology](#) の 3. API 関数をご参照ください。

UART FSP モジュールの API の詳細は [RX Flexible Software Package Documentation](#) の API Reference > Modules > Connectivity > [UART \(r_sci_b_uart\)](#) をご参照ください。

表 4.24 RSCI モジュールの機能対応(1)

No	FIT モジュール API	FSP モジュール API
1	R_RSCI_Open()	R_SCI_B_UART_Open()
2	R_RSCI_Close()	R_SCI_B_UART_Close()
3	R_RSCI_Send()	R_SCI_B_UART_Write()
4	R_RSCI_Receive()	R_SCI_B_UART_Read()
5	R_RSCI_SendReceive()	該当機能なし。(注)
6	R_RSCI_Control()	以下一部該当機能あり。
	Cmd = SCI_CMD_CHANGE_BAUD	R_SCI_B_UART_BaudSet()
	Cmd = SCI_CMD_CHANGE_TX_FIFO_THRESH	該当機能なし。(注)
	Cmd = SCI_CMD_CHANGE_RX_FIFO_THRESH	該当機能なし。(注)
	Cmd = SCI_CMD_SET_RXI_PRIORITY	R_SCI_B_UART_Open() p_cfg->rx_ipl = (x) (x = 0 – 15)
	Cmd = SCI_CMD_SET_TXI_PRIORITY	R_SCI_B_UART_Open() p_cfg->tx_ipl = (x) (x = 0 – 15)
	Cmd = SCI_CMD_EN_NOISE_CANCEL	該当機能なし。(注)
	Cmd = CMT_RX_CMD_GET_PRIORITY	該当機能なし。(注)
	Cmd = SCI_CMD_OUTPUT_BAUD_CLK	該当機能なし。(注)
	Cmd = SCI_CMD_START_BIT_EDGE	該当機能なし。(注)
	Cmd = SCI_CMD_GENERATE_BREAK	該当機能なし。(注)
	Cmd = SCI_CMD_COMPARE_RECEIVED_DATA	該当機能なし。(注)
	Cmd = SCI_CMD_TX_Q_FLUSH	該当機能なし。(注)
	Cmd = SCI_CMD_RX_Q_FLUSH	該当機能なし。(注)
	Cmd = SCI_CMD_TX_Q_BYTES_FREE	該当機能なし。(注)

【注】 類似機能が必要な場合、FIT モジュールのソースコードを参考にお客様自身で実装してください。

表 4.25 RSCI モジュールの機能対応(2)

No	FIT モジュール API	FSP モジュール API
6	Cmd = SCI_CMD_RX_Q_BYTES_AVAIL_TO_READ	該当機能なし。(注)
	Cmd = SCI_CMD_EN_CTS_IN	該当機能なし。(注)
	Cmd = SCI_CMD_CHECK_XFER_DONE	該当機能なし。(注)
	Cmd = SCI_CMD_ABORT_XFER	該当機能なし。(注)
	Cmd = SCI_CMD_XFER_LSB_FIRST	該当機能なし。(注)
	Cmd = SCI_CMD_XFER_MSB_FIRST	該当機能なし。(注)
	Cmd = SCI_CMD_INVERT_DATA	該当機能なし。(注)
	Cmd = SCI_CMD_CHANGE_SPI_MODE	該当機能なし。(注)
	Cmd = SCI_CMD_CHECK_TX_DONE	該当機能なし。(注)
	Cmd = SCI_CMD_CHECK_RX_DONE	該当機能なし。(注)
	Cmd = SCI_CMD_CHECK_RX_SYNC_DONE	該当機能なし。(注)
7	R_RSCI_GetVersion()	モジュール毎のバージョンはなし。 FSP のバージョンは以下の関数で確認 できます。 R_FSP_VersionGet()

【注】 類似機能が必要な場合、FIT モジュールのソースコードを参考にお客様自身で実装してください。

表 4.26 RSCI モジュールの機能対応(3)

No	FIT モジュール割り込み関数・コールバック関数	FSP モジュール割り込み関数・コール バック関数
1	rscin_txi_isr() (n = チャネル番号)	sci_b_uart_txi_isr()
2	rscin_tei_isr() (n = チャネル番号)	sci_b_uart_tei_isr()
3	rscin_rxi_isr() (n = チャネル番号)	sci_b_uart_rxi_isr()
4	rscin_eri_isr() (n = チャネル番号)	sci_b_uart_eri_isr()
5	コールバック関数は以下の API 関数の第 4 引数 p_callback にユーザ定義関数を指定 R_RSCI_Open()	コールバック関数は以下の API 関数 R_SCI_B_UART_Open() もしくは以下の第 2 引数 p_callback に ユーザ定義関数を指定 R_SCI_B_UART_CallbackSet()

4.3.2.3 コード生成 SCI/SCIF 調歩同期式モード用 API 関数と UART FSP モジュールの比較

コード生成 SCI/SCIF 調歩同期式モード用 API 関数と SCI UART FSP モジュールの API の機能対応表を以下に示します。コード生成 SCI/SCIF 調歩同期式モード用 API に記載された API は FSP モジュール API に置き換えることができます。コード生成 SCI/SCIF 調歩同期式モード用 API 関数の詳細は、[スマート・コンフィグレータ ユーザーズマニュアル RX API リファレンス編](#)の 4.2.31 SCI/SCIF 調歩同期式モードをご参照ください。UART FSP モジュールの API の詳細は [RX Flexible Software Package Documentation](#) の API Reference > Modules > Connectivity > [UART \(r_sci_b_uart\)](#) をご参照ください。

表 4.27 コード生成 SCI/SCIF 調歩同期式モード用 API 関数の機能対応

No	コード生成 SCI/SCIF 調歩同期式モード用 API	FSP モジュール API
1	R_<Config_SCI0>_Create()	R_SCI_B_UART_Open()
2	R_<Config_SCI0>_Start()	R_SCI_B_UART_Open()
3	R_<Config_SCI0>_Stop()	R_SCI_B_UART_Close()
4	R_<Config_SCI0>_Serial_Send()	R_SCI_B_UART_Write()
5	R_<Config_SCI0>_Serial_Receive()	R_SCI_B_UART_Read()
6	R_<Config_SCI0>_Serial_Multiprocessor_Send()	該当機能なし。
7	R_<Config_SCI0>_Serial_Multiprocessor_Receive()	該当機能なし。
8	R_<Config_SCI0>_Create_UserInit()	該当機能なし。(注)

【注】 スマート・コンフィグレータの API は引数がない関数が多く（GUI で変数を設定できないため）、ユーザプログラムで変数を活用したい場合の初期化処理のために UserInit 関数がありましたが、FSP ではそのような使用方法はないため、該当機能がありません。

表 4.28 コード生成 SCI/SCIF 調歩同期式モード用割り込み関数・コールバック関数の機能対応

No	コード生成 SCI/SCIF 調歩同期式モード用割り込み・コールバック関数	FSP モジュール割り込み関数・コールバック関数
1	r_<Config_SCI0>_transmitend_interrupt()	sci_b_uart_tei_isr()
2	r_<Config_SCI0>_transmit_interrupt()	sci_b_uart_txi_isr()
3	r_<Config_SCI0>_receive_interrupt()	sci_b_uart_rxi_isr()
4	r_<Config_SCI0>_receiveerror_interrupt()	sci_b_uart_eri_isr()
5	r_<Config_SCI0>_callback_transmitend() 関数内ユーザ定義処理	ユーザ定義コールバック関数に置き換え。 コールバック関数は以下の API 関数 R_SCI_B_UART_Open() もしくは以下の第 2 引数 p_callback にユーザ定義関数を指定 R_SCI_B_UART_CallbackSet()
6	r_<Config_SCI0>_callback_receiveend() 関数内ユーザ定義処理	ユーザ定義コールバック関数に置き換え。 コールバック関数は以下の API 関数 R_SCI_B_UART_Open() もしくは以下の第 2 引数 p_callback にユーザ定義関数を指定 R_SCI_B_UART_CallbackSet()
7	r_<Config_SCI0>_callback_receiveerror() 関数内ユーザ定義処理	ユーザ定義コールバック関数に置き換え。 コールバック関数は以下の API 関数 R_SCI_B_UART_Open() もしくは以下の第 2 引数 p_callback にユーザ定義関数を指定 R_SCI_B_UART_CallbackSet()

5. FSP 以前のプロジェクトから FSP プロジェクトに移行する手順

FIT モジュールを使用したプロジェクト（以下 FIT プロジェクト）またはコード生成を使用したプロジェクト（以下コード生成プロジェクト）から FSP プロジェクトへ移行は下表に示す手順で行います。

表 5.1 FIT プロジェクトまたはコード生成プロジェクトから FSP プロジェクトに移行する手順

No	手順	条件	参照先
1	FSP プロジェクトを新規作成する。		5.1 参照。
2	FSP プロジェクトの基本設定を行う。		5.2 参照。
3	FSP プロジェクトのクロック設定を行う。		5.3 参照。
4	FSP モジュールを追加する。		5.4 参照。
5	FSP プロジェクトの割り込み設定を行う。	使用する周辺機能において FSP モジュールを使用せず割り込みを使用する場合	5.5 参照。
6	IOPORT モジュールを使用し、端子設定処理を追加する。		5.6 参照。
7	ソースコードの必要部分を移植する。		5.7 参照。
8	iodefine.h を FSP プロジェクトに追加する。	IO レジスタにアクセスする処理を移植する場合	5.8 参照。
9	FIT モジュールまたはコード生成の処理を FSP モジュールの処理に置き換える。		5.9 参照。

スマート・コンフィグレータを使用しないプロジェクトから FSP プロジェクトへ移行する方法について以下に示します。

表 5.2 スマート・コンフィグレータを使用しないプロジェクトから FSP プロジェクトに移行する手順

No	手順	条件	参照先
1	FSP プロジェクトを新規作成する。		5.1 参照。
2	FSP プロジェクトの基本設定を行う。		5.2 参照。
3	FSP プロジェクトのクロック設定を行う。		5.3 参照。
4	FSP プロジェクトの割り込み設定を行う。	割り込み機能を使用する場合	5.5 参照。
5	ソースコードの必要部分を移植する。		5.7 参照。
6	IOPORT モジュールを使用し、端子設定処理を追加する。	端子設定処理を移植しない場合	5.6 参照。
7	iodefine.h を FSP プロジェクトに追加する。	IO レジスタにアクセスする処理を移植する場合	5.8 参照。

5.1 FSP プロジェクトの新規作成

2.3 RX FSP のプロジェクト作成手順を参考に新規の FSP プロジェクトを作成します。

5.2 基本設定の移行

FSP プロジェクトではプロジェクト生成時にプロジェクトに自動的に BSP が組み込まれます。BSP にはスタートアップ処理が含まれます。FSP の BSP のスタートアップ処理の概要については 4.2.3 FSP BSP の初期設定を参照ください。リセット解除後に実行される BSP のスタートアップ処理の中でスタック設定、ベクタテーブルの設定、クロック設定、割り込み設定等の基本設定を行います。

そのため、使用するボードに合わせて BSP ページのプロパティにある"RX Common"の項目の選択を行う必要があります。（設定は必要に応じて移行元のプロジェクトのご使用の条件に合わせて変更してください。）

BSP ページの GUI のプロパティ表示は以下のとおりです。

Summary	BSP	Clocks	Pins	Interrupts	Event Links	Linker Sections	Stacks	Components
プロパティ × 問題 スマート・ブラウザー								
EK-RX74M								
Settings	プロパティ		値					
	▼ RX Common							
	Main stack size (bytes)		0x400					
	Heap size (bytes)		0					
	Protect zone root stack size (bytes)		0x800					
	MCU Vcc (mV)		3300					
	Parameter checking		Disabled					
	Assert Failures		Return FSP_ERR_ASSERTION					
	Clock Registers not Reset Values during Startup		Disabled					
	Main Oscillator Populated		Populated					
	PFS Protect		Enabled					

図 5-1 BSP のプロパティ

"RX Common"の GUI で選択した情報がプロジェクトコード生成（Generate Project Content 実行）後、rx_cfg フォルダに生成される bsp_cfg.h のマクロ定義に反映されます。

上記設定によって生成した後の bsp_cfg.h を以下に示します。

プロジェクト・エクスプローラー ×	bsp_cfg.h ×
FSP_example [Debug] - Includes - rx - rx_gen - src - Debug - rx_cfg - fsp_cfg - bsp - board_cfg.h - bsp_cfg.h - bsp_mcu_device_cfg.h	<pre> 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 @if defined(_RX_BOOT_IMAGE) #define BSP_CFG_BOOT_IMAGE (1) #endif #define BSP_CFG_MCU_VCC_MV (3300) #define BSP_CFG_STACK_MAIN_BYTES (0x400) #define BSP_CFG_HEAP_BYTES (0) #define BSP_CFG_PZ_ROOT_STACK_SIZE (0x800) #define BSP_CFG_PARAM_CHECKING_ENABLE (0) #define BSP_CFG_ASSERT (0) #define BSP_CFG_PFS_PROTECT ((1)) #define BSP_CFG_C_RUNTIME_INIT ((1)) #define BSP_CFG_EARLY_INIT ((0)) #define BSP_CFG_STARTUP_CLOCK_REG_NOT_RESET ((0)) </pre>

図 5-2 bsp_cfg.h のマクロ定義

移行時の設定例を以下に示します。

表 5.3 FIT プロジェクトまたはコード生成プロジェクトから移行時の設定項目と設定値(1)

No	BSP コンポーネントでの設定項目および設定値	RX Common での設定項目および設定値
1	User stack setting を 2 stacks に設定。(USP と ISP)	なし
2	User stack size に 0x200 を指定。	なし
3	Interrupt stack size に 0x200 を指定。	Main stack size (byte)に 0x200 を指定。
4	Heap size に 0x400 を指定。	Heap size (byte)に 0x400 を指定。
5	Initializes C input and output library functions に Disabled を指定。	なし
6	Enable user stdio charget function に Use BSP charget() function を指定。	なし
7	Enable user stdio charput function に Use BSP charput() function を指定。	なし
8	Processor Mode に Stay in Supervisor mode を指定。	なし
9	ID code 1~ID code 4 に任意値を指定。	なし
10	Serial programmer connection select に Enabled を指定。	なし
11	ROM Cache Enable Register に Enabled を指定。	なし
12	TMEF – TM Enable Flag Register に任意値を指定。	なし
13	FAW – Flash Access Window Setting Register に任意値を指定。	なし
14	ROMCODE – ROM Code Protection Register に任意値を指定。	なし
15	Select the startup bank に Bank 1: FFC00000h~FFDFFFFFh; Bank 0: FFE00000h~FFFFFFFh.を指定。	なし
16	Lock function select に Use default locking (non-RTOS)を指定。	なし
17	Enable user warm start callback (PRE)に Unused を指定。	なし
18	Enable user warm start callback (POST)に Unused を指定。	なし
19	Parameter checking に Enabled を指定。	Parameter checking に Enabled を指定。
20	Extended Bus Master 1st Priority Selection に GLCDC graphics 1 data read を指定。	なし
21	Extended Bus Master 2nd Priority Selection に GLCDC graphics 2 data read を指定。	なし
22	Extended Bus Master 3rd Priority Selection に DRW2D texture data read を指定。	なし
23	Extended Bus Master 4th Priority Selection に DRW2D frame buffer data read write and display list data read を指定。	なし
24	Extended Bus Master 5th Priority Selection に EDMAC を指定。	なし

表 5.4 FIT プロジェクトまたはコード生成プロジェクトから移行時の設定項目と設定値(2)

No	BSP コンポーネントでの設定項目および設定値	RX Common での設定項目および設定値
25	Non-cacheable area 0 select に Disabled を指定。	なし
26	Non-cacheable area 1 select に Disabled を指定。	なし
27	Interrupt Priority Level When FIT Module Interrupts Are Disabled に Priority level 15 (highest)を指定。	なし
28	Software Interrupt Unit1 (SWINT1)に Unused を指定。	なし
29	Software Interrupt Unit2 (SWINT2)に Unused を指定。	なし
30	Serial terminal select に Disabled を指定。	なし
31	Select whether to enable bus priority initialization.に Disabled を指定。	なし
32	Select whether it is bootloader project.に Not bootloader project を指定。	なし

表 5.5 FIT プロジェクトまたはコード生成プロジェクトから移行時の設定項目と設定値(3)

No	クロックページでの設定項目および設定値	RX Common での設定項目および設定値
1	クロックのメインクロックにチェック（発振）。	Main Oscillator Populated で Populated （発振）を指定。
2	クロックのメインクロックの発信源に発振子を選択している。	Main Oscillator Clock Source で Crystal or Resonator を指定。
3	クロックのサブクロックに未チェック（発振器しない）。	Subclock Populated で Not Populated （発振しない）を指定。
4	サブクロックの発振器ドライブ能力に標準 CL を指定。	Subclock Drive (Drive capacitance availability varies by MCU)で Standard/Nomal mode を指定。
5	サブクロックの発振安定時間に 2000(ms)を指定。	Subclock Stabilization Time (ms)に 2000 を指定。
6	クロックの IWDG 専用クロックにチェック（発振）。	なし

表 5.6 スマート・コンフィグレータを使用しないプロジェクトから移行時の設定項目と設定値

No	移行元のプロジェクトでの仕様	RX Common での設定項目および設定値
1	ustack(USP)に 0x200 を指定。	なし（USP は使用できません。）
2	istack(ISP)に 0x200 を指定。	Main stack size (byte)に 0x200 を指定。
3	メインクロック発振器を使用。	Main Oscillator Populated で Populated （発振）を指定。
4	メインクロック発振器をボードに搭載している。	Main Oscillator Clock Source で Crystal or Resonator を指定。
5	サブクロック発振器を使用しない。	Subclock Populated で Not Populated （発振しない）を指定。

5.3 クロック設定の移行

FSP プロジェクトでは FIT プロジェクトまたはコード生成プロジェクトと同様にリセット解除後のスタートアップ処理のシステムクロック設定で[Clocks Configuration]ページのクロック設定に基づいたクロック発生回路のレジスタ設定を行います。FSP の BSP のスタートアップ処理の概要については 4.2.3 FSP BSP の初期設定を参照ください。

[Clocks Configuration]ページは以下のとおりです。

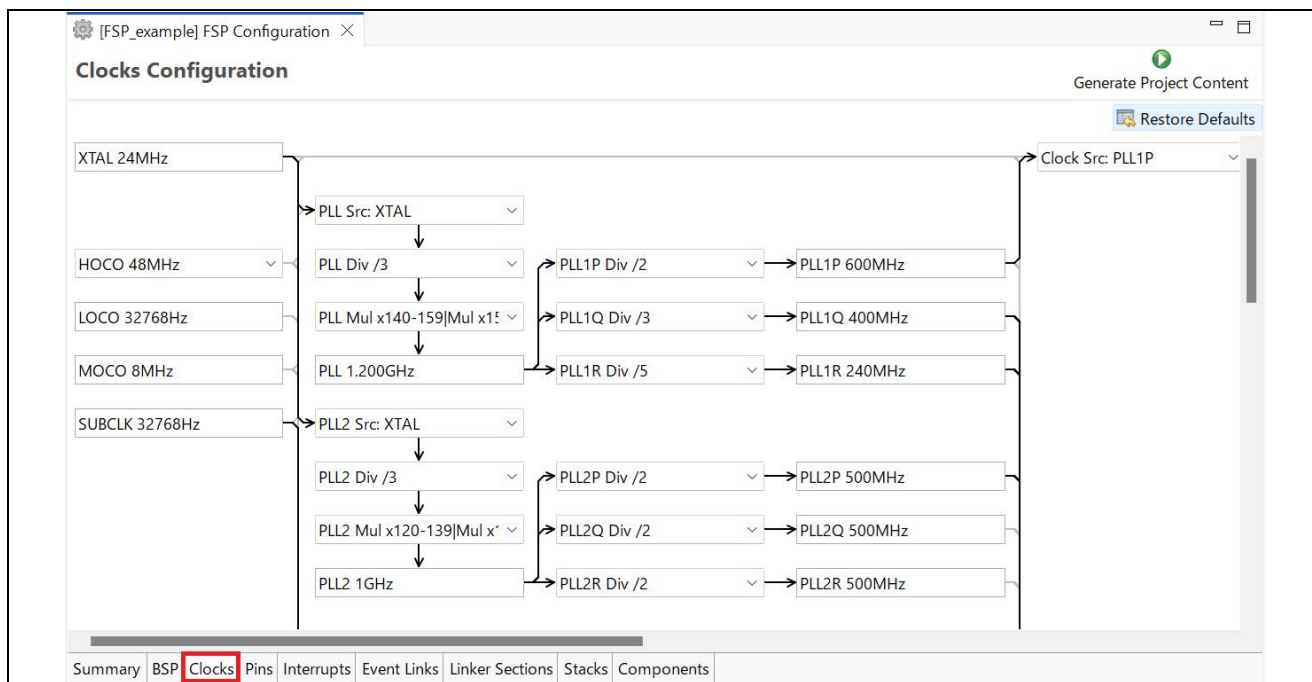


図 5-3 [Clocks Configuration]ページ

[Clocks Configuration]ページで選択した情報がプロジェクトコード生成（Generate Project Content 実行）後、rx_gen フォルダに生成される bsp_clock_cfg.h のマクロ定義に反映されます。

上記設定によって生成した後の bsp_clock_cfg.h を以下に示します。

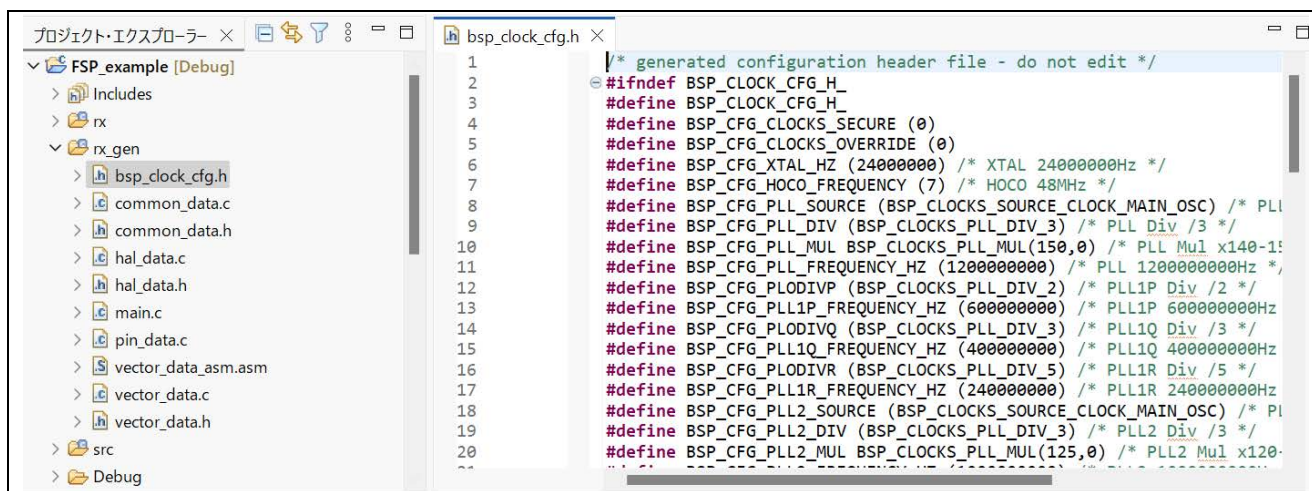


図 5-4 bsp_clock_cfg.h のマクロ定義

bsp_clock_cfg.h の各マクロ定義値によってクロック設定が行われるため、移行元のプロジェクトのクロックページで行っていたクロック設定もしくはクロック設定処理と同じになるように[Clocks Configuration]ページの設定を合わせる必要があります。移行元のプロジェクトでのクロック設定処理は FSP プロジェクトの BSP のクロック設定と競合しますので直接移植しないでください。

5.4 モジュールの移行

移行元のプロジェクトで使用していたモジュールと同機能のモジュールを FSP プロジェクトに追加します。FSP では各モジュールに Stack を割り当てる必要があります。モジュールを追加する場合は[Stacks Configuration]ページから追加します。

[Stacks Configuration]ページは以下のとおりです。プロジェクト生成後の初期状態では IOPORT モジュールのみが追加されています。

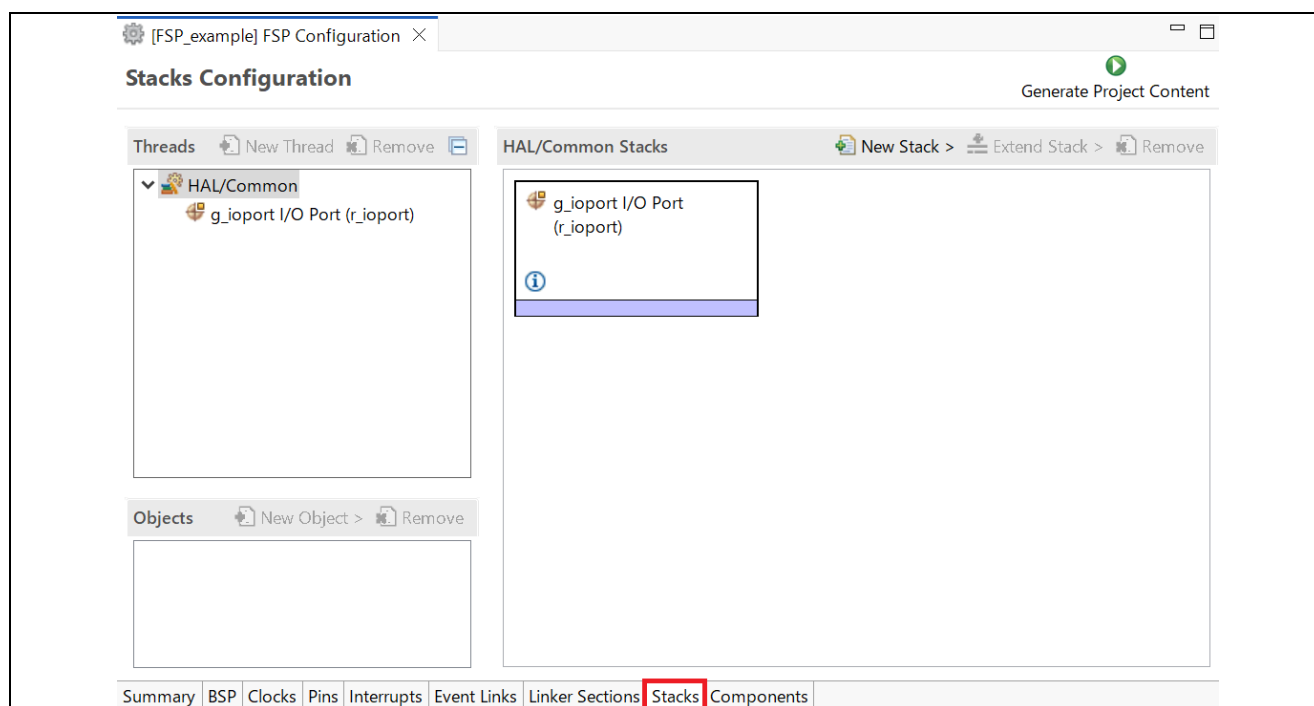


図 5-5 [Stacks Configuration]ページ

モジュールを追加したい場合は、[New Stack] ボタンから該当する機能を選択し、その中から追加するモジュールを選択すると選択したモジュールが HAL/Common Stacks の一覧に追加されます。一例ですが、CMT のモジュールを追加したい場合は[New Stack] → [Timers] → [Timer, Compare Match Timer (r_cmt)] を選択します。

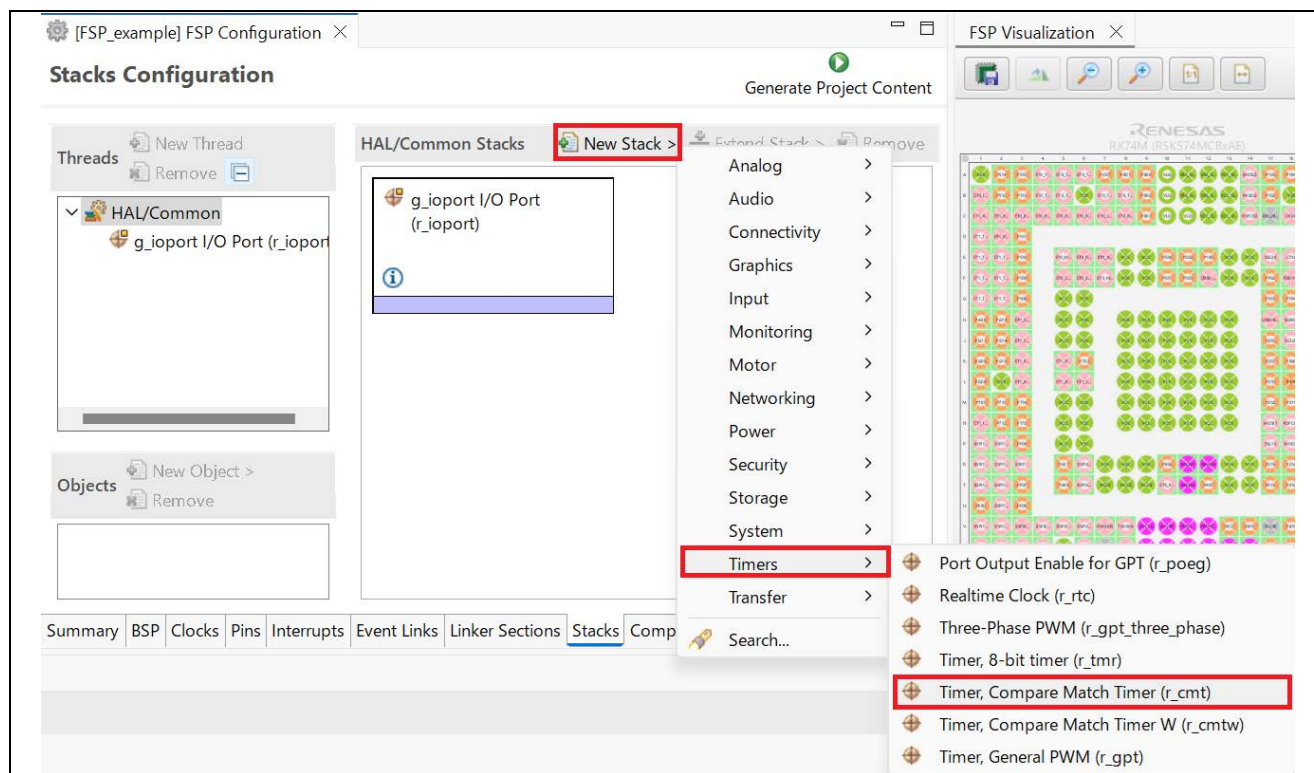


図 5-6 タイマの選択

選択した CMT モジュールが HAL/Common Stacks に追加されます。

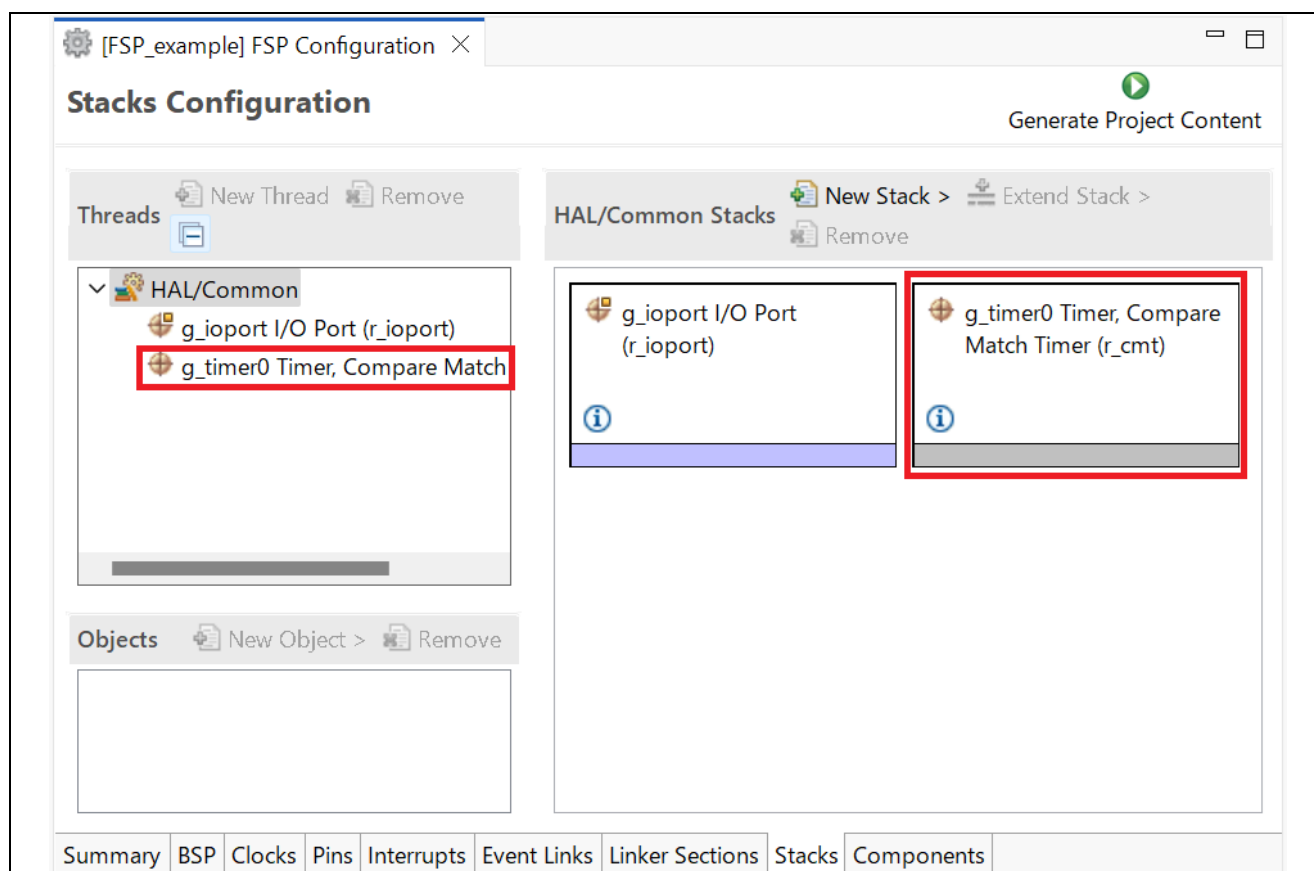


図 5-7 CMT モジュール(r_cmt)の追加

モジュールに個別に設定が必要な場合は、[Stacks Configuration]ページで追加したモジュールを選択し、プロパティから設定項目を変更します。

CMT モジュールの場合は以下のようになります。

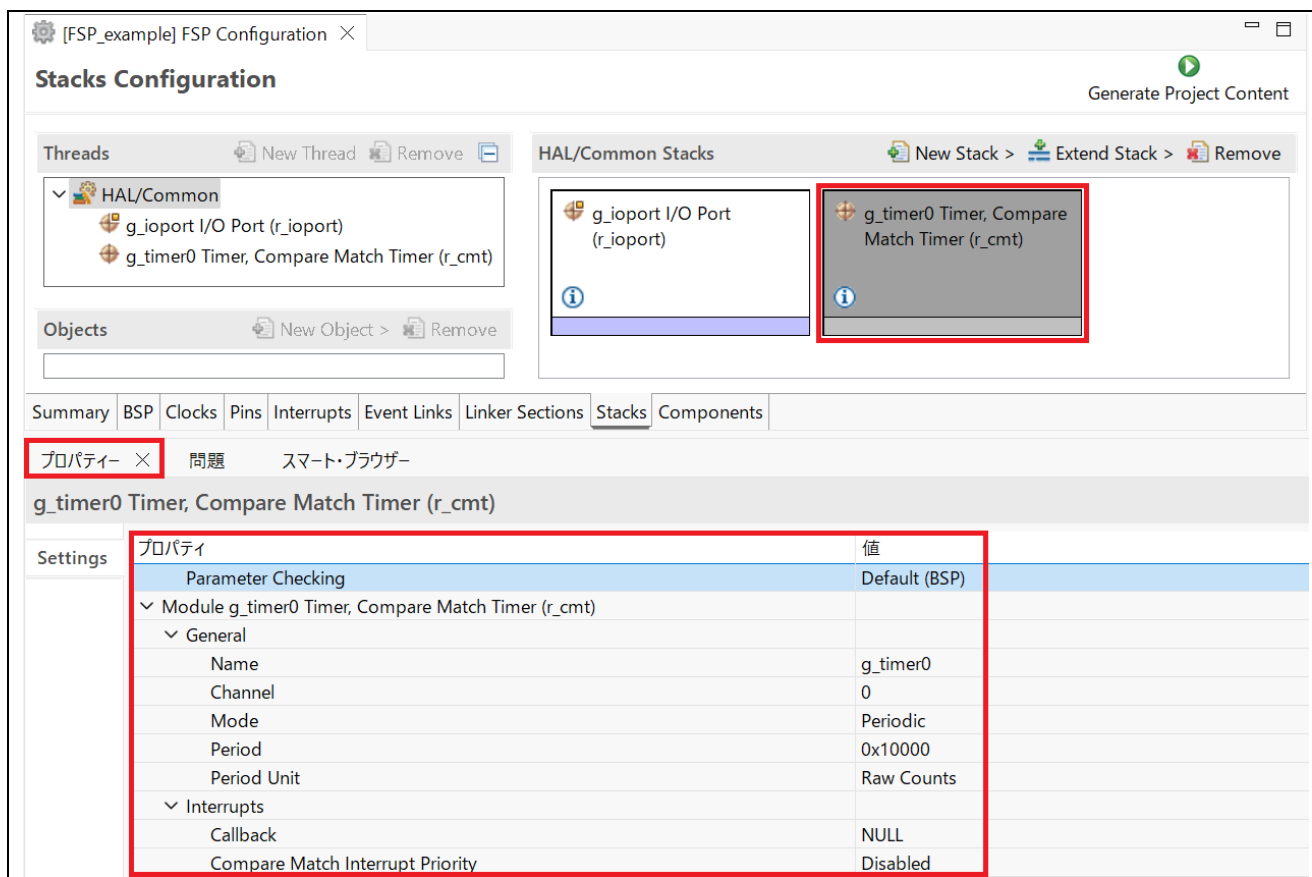


図 5-8 CMT モジュール(r_cmt)のプロパティ

[Generate Project Content]ボタンをクリックすることでプロジェクトにモジュールが追加されます。

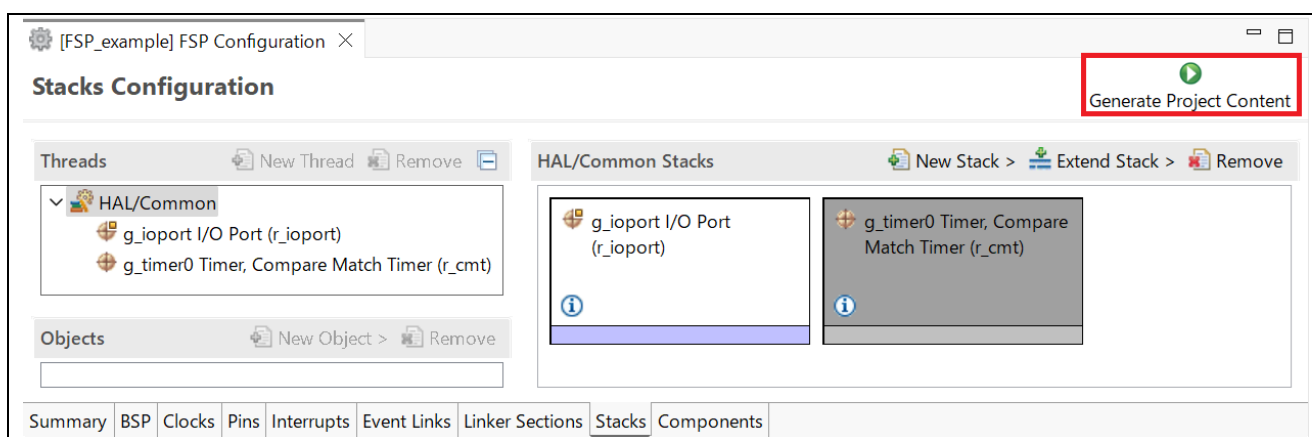


図 5-9 [Generic Project Content]

追加したモジュールにより、rx_gen フォルダの hal_data.h と hal_data.c にモジュールに関連する定義が出力されます。

CMT モジュールを初期状態のプロパティで追加した場合は以下赤字部分が追加されます。追加される内容はモジュールやモジュールのプロパティ設定により異なります。

rx_gen¥hal_data.h

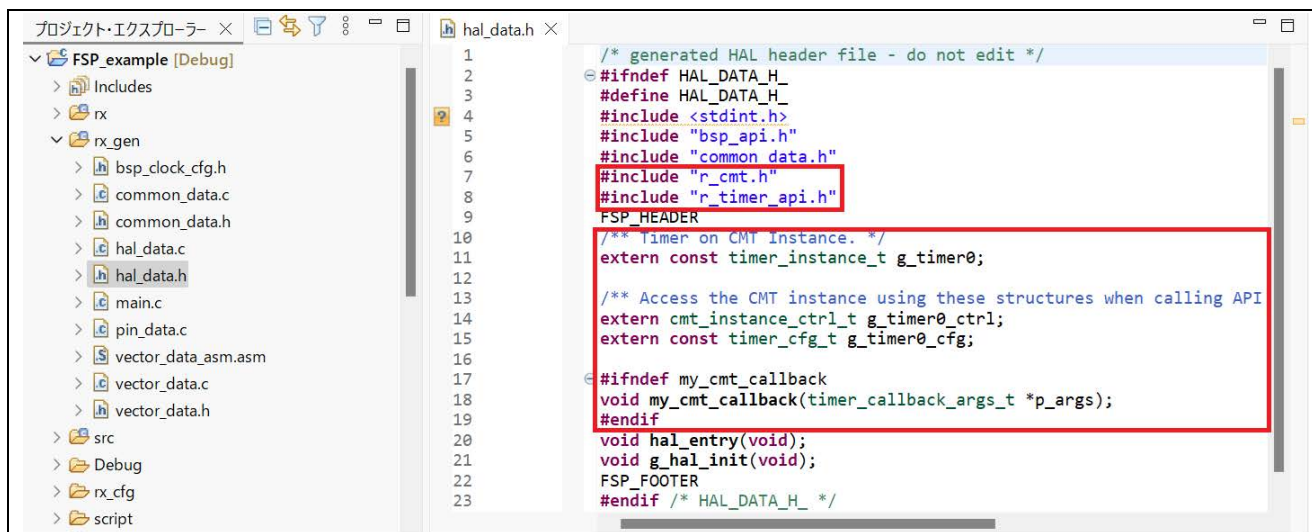


図 5-10 hal_data.h の定義

rx_gen¥hal_data.c

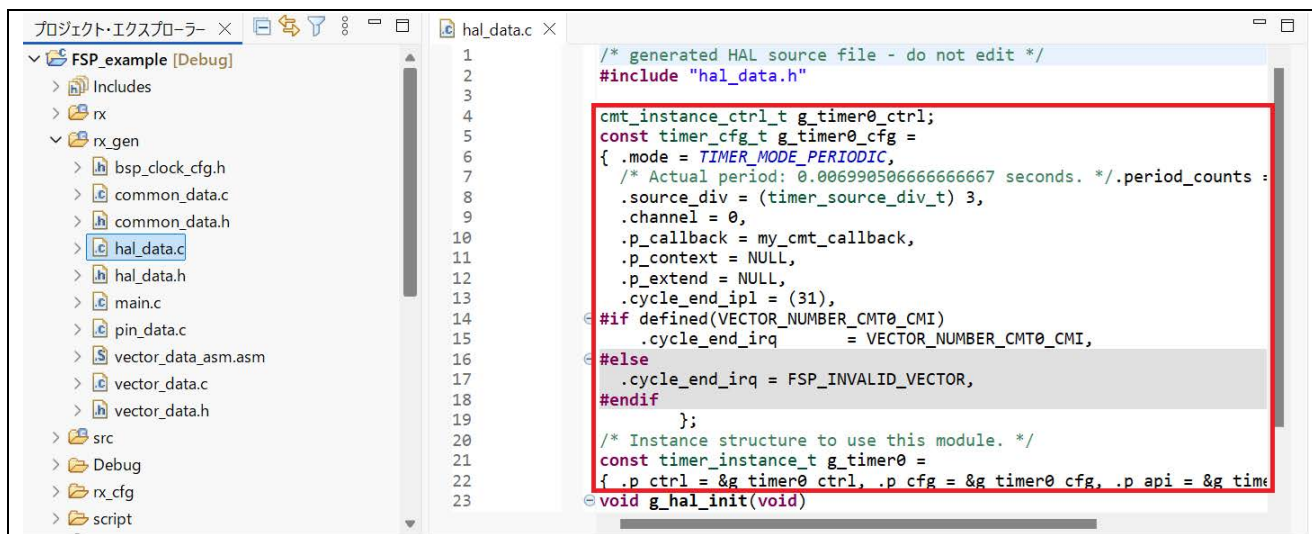


図 5-11 hal_data.c の定義

[Components Configuration]ページからコンポーネントの一覧を確認することができます。

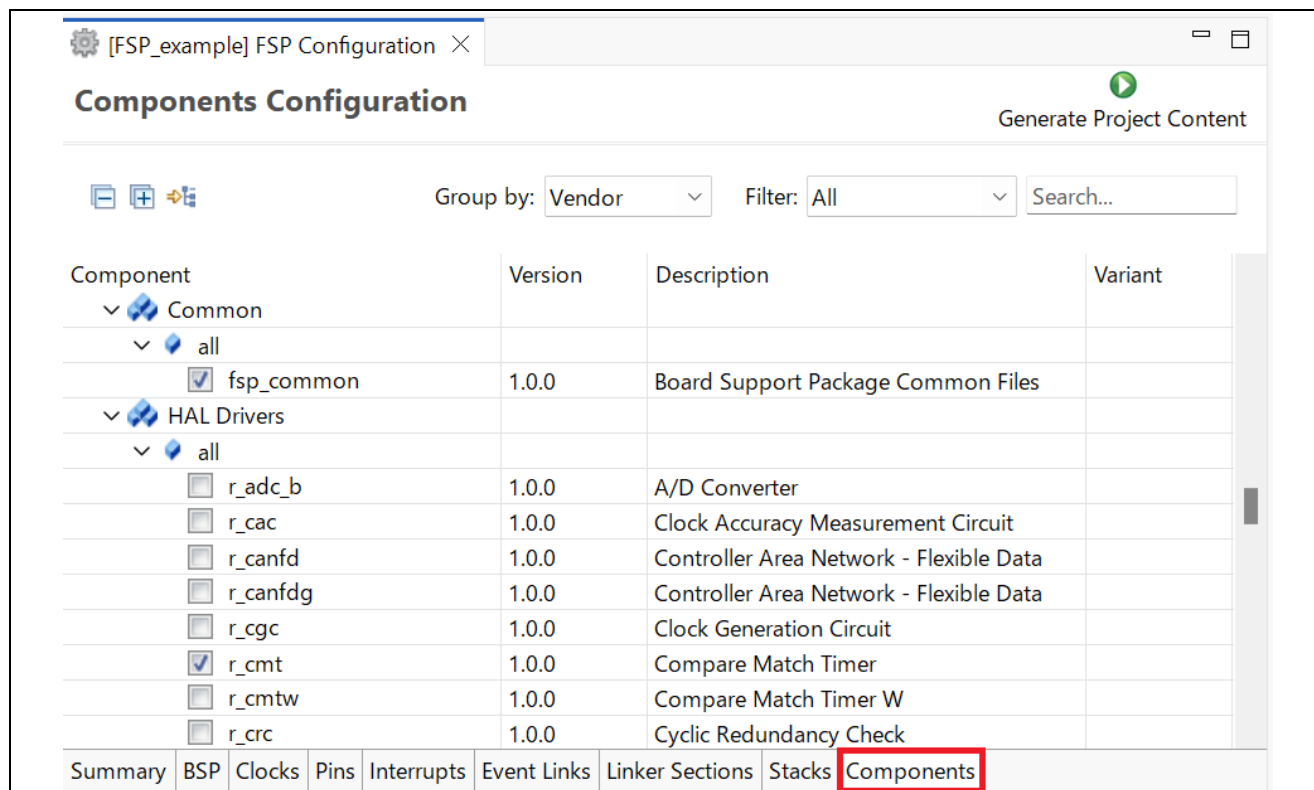


図 5-12 コンポーネント一覧

5.5 割り込みの移行

FSP プロジェクトでは BSP が例外ベクタテーブルと割り込みベクタテーブルの定義を保持します。割り込み設定によってベクタテーブルに登録するための割り込み関数の宣言を自動で生成する機能を有します。割り込み関数の生成機能は割り込みの設定方法により異なります。詳細は後述の 5.5.1 モジュール追加による割り込み設定および 5.5.2 直接追加による割り込み設定を参照ください。

[Interrupts Configuration]ページは以下のとおりです。

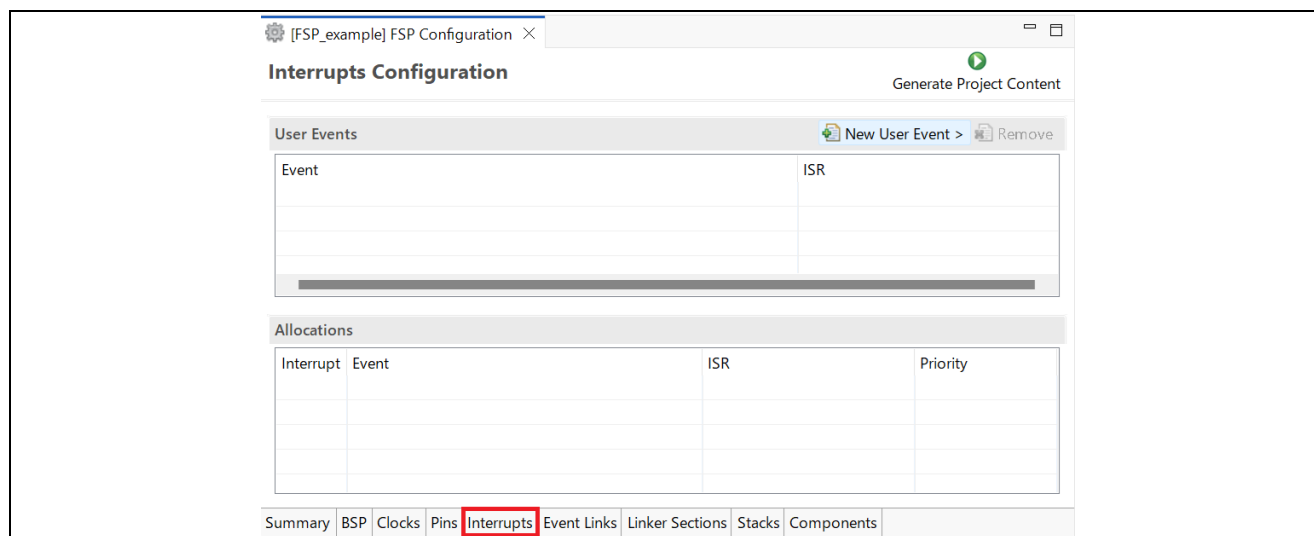


図 5-13 [Interrupts Configuration]ページ

5.5.1 モジュール追加による割り込み設定

モジュール追加時に割り込みに関する設定（コールバック関数や割り込み優先レベルの設定）を有効にするとモジュール追加時にその内容に沿った情報が追加されます。

CMT モジュールのプロパティで割り込みに関する設定を以下のように有効にしていた場合は、

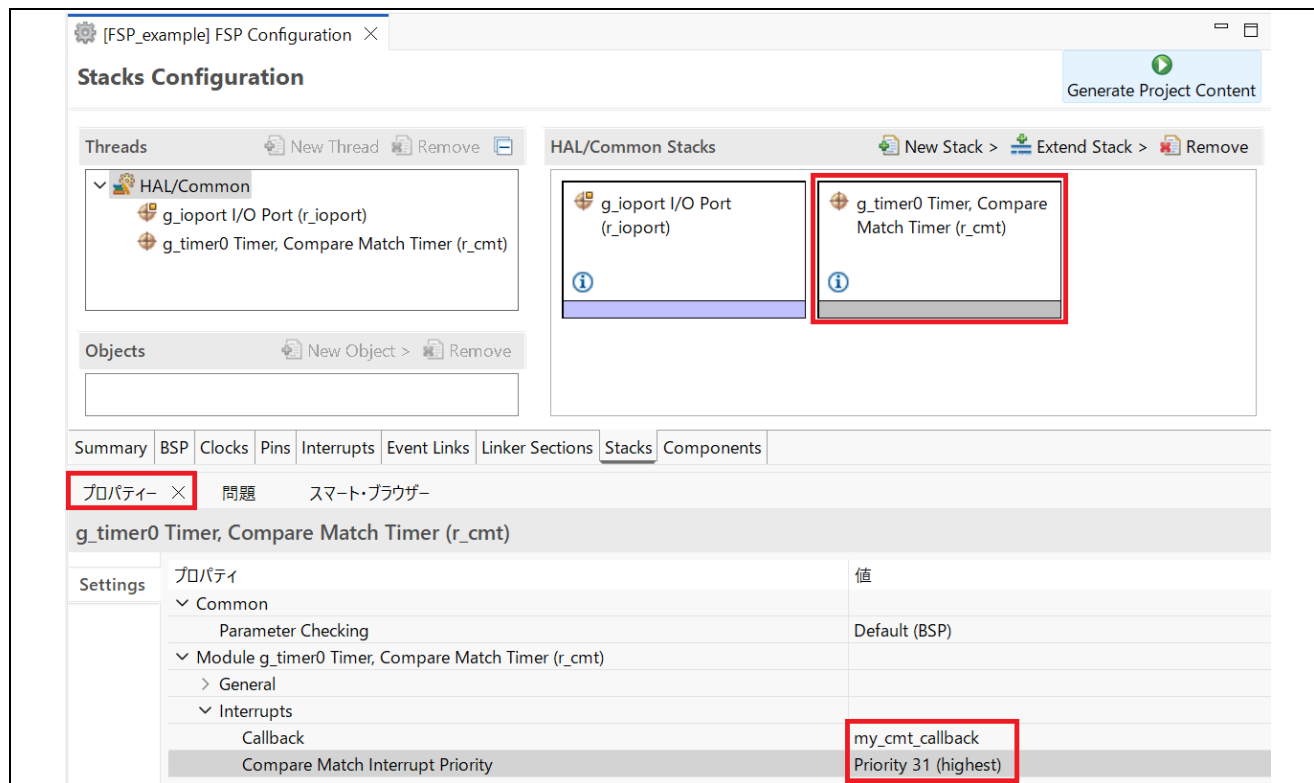


図 5-14 [Stacks Configuration]ページ CMT 割り込み許可設定

以下のように CMT0 の割り込みを使用する情報が登録されます。割り込み関数はモジュールに含まれる割り込み関数名となります。

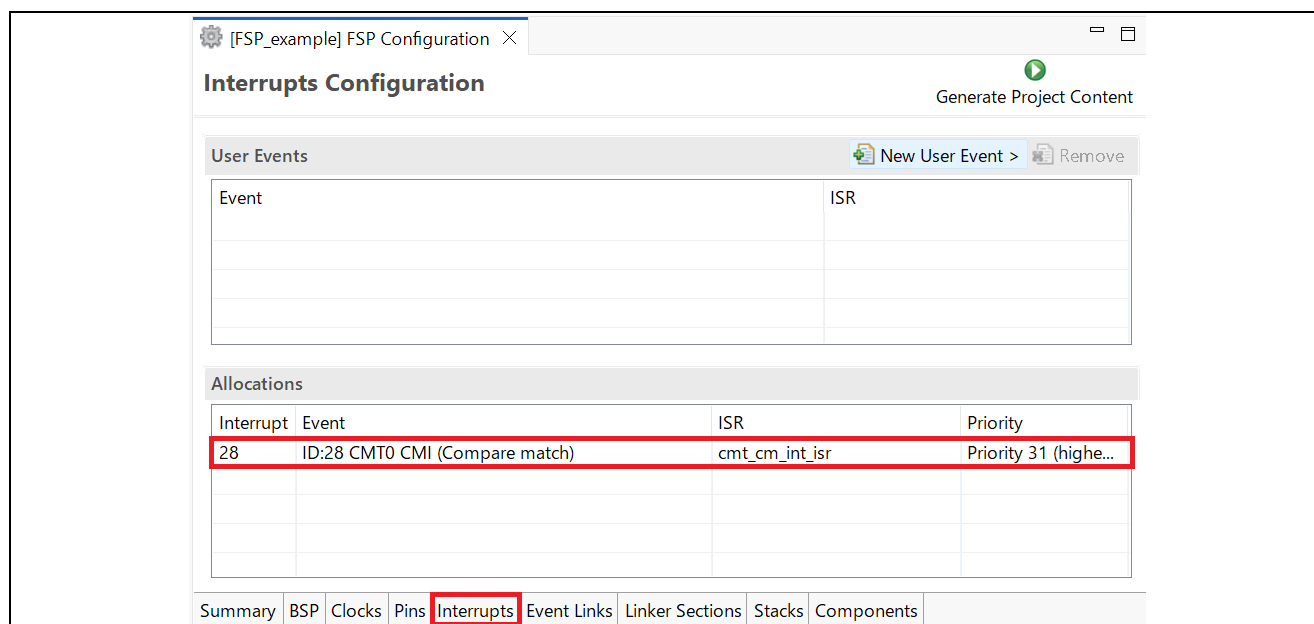


図 5-15 [Interrupts Configuration]ページ CMT 割り込み登録

[Generate Project Content]を実行すると割り込み関数の定義が生成され、vector_data.hに割り込み関数の宣言が出力されます。上記の設定で割り込み関数が出力された vector_data.h は以下ようになります。

rx_gen¥vector_data.h

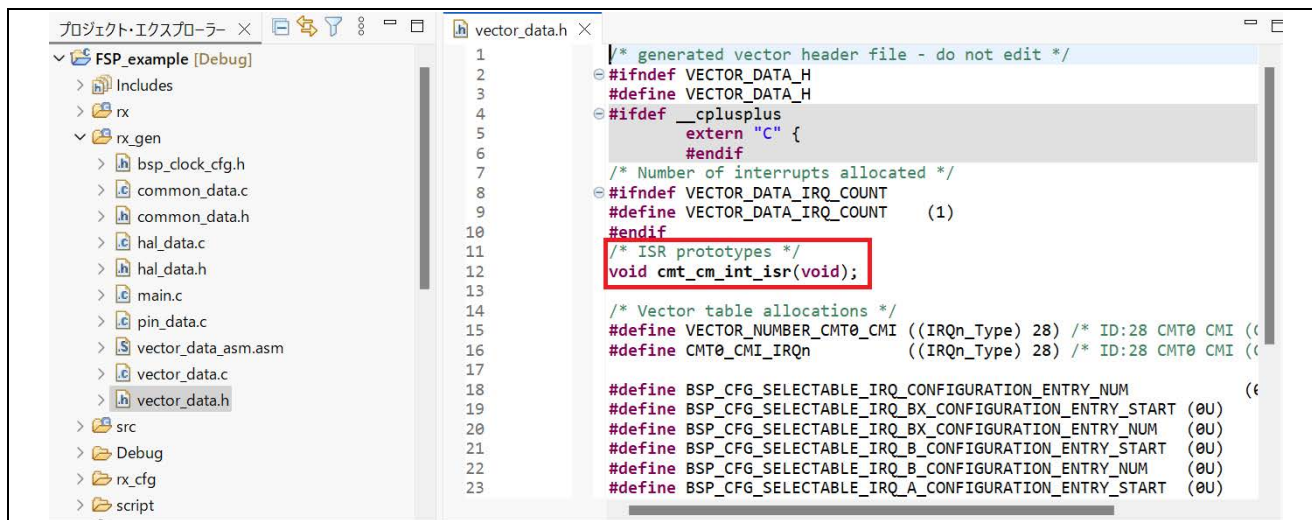


図 5-16 vector_data.h 割り込み関数宣言

コールバック関数の宣言は hal_data.h に出力されます。

rx_gen¥hal_data.h

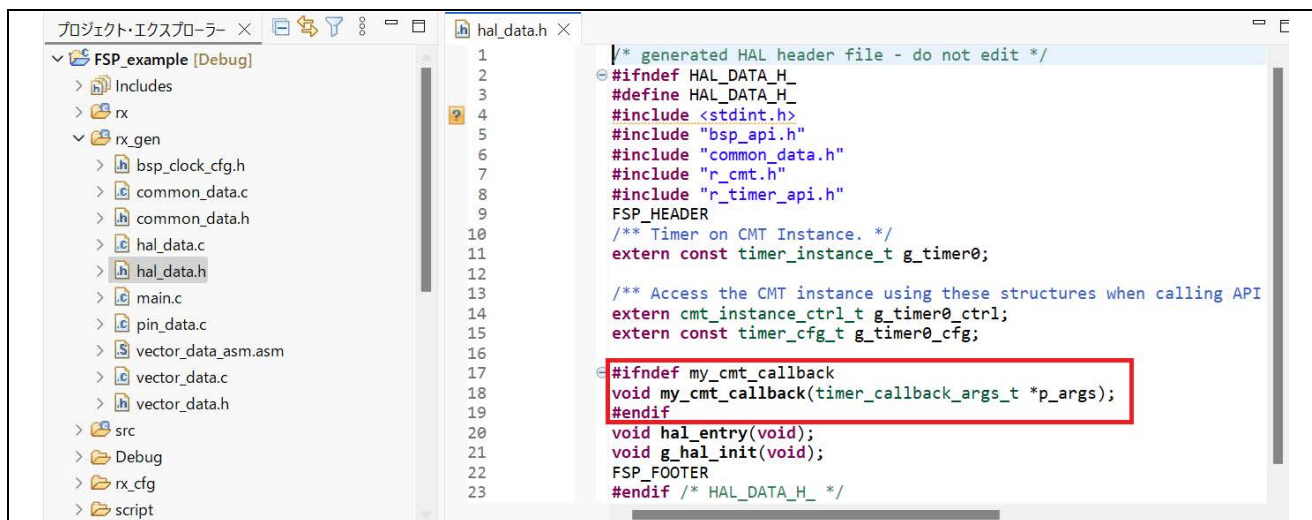


図 5-17 hal_data.h コールバック関数宣言

割り込み関数の実体はモジュール内に定義されていますが、

rx\fsp\src\r_cmt\r_cmt.c

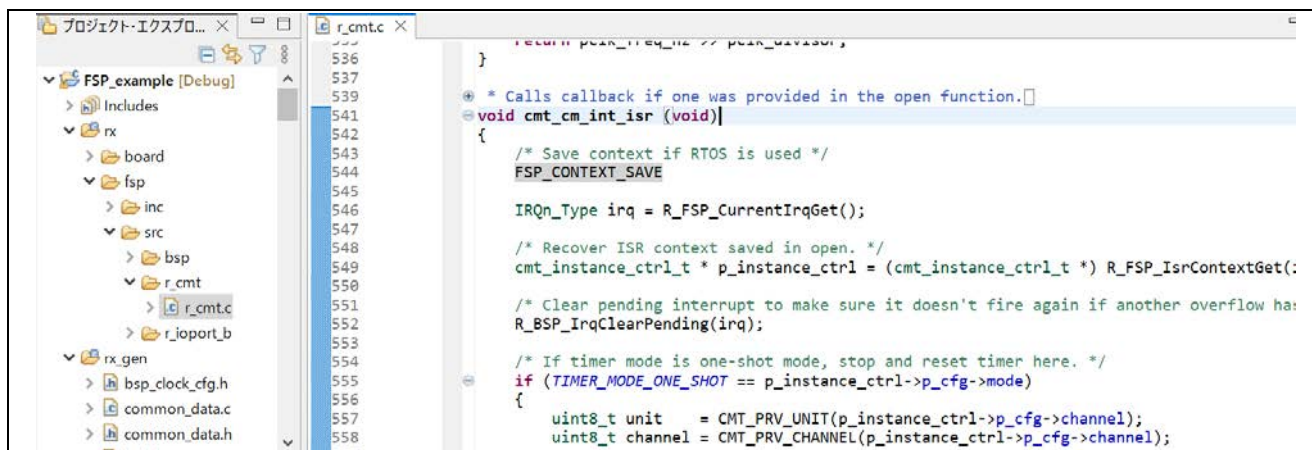


図 5-18 r_cmt.c 割り込み関数

コールバック関数は宣言のみが出力されるため、コールバック関数の実体はユーザ自身で作成する必要があります。

FSP モジュールを使用しない周辺機能で割り込み機能を使用する場合は 5.5.2 直接追加による割り込み設定を参照してください。

5.5.2 直接追加による割り込み設定

[New User Event]から追加する割り込み要因を選択し、割り込み関数名を入力してから [Generate Project Content]を実行すると割り込み関数の定義が生成され、vector_data.h に割り込み関数の宣言が出力されます。

以下のように[Interrupts Configuration]ページから CMT0 の割り込み要因を選択し、CMT0 の割り込み関数を定義した場合

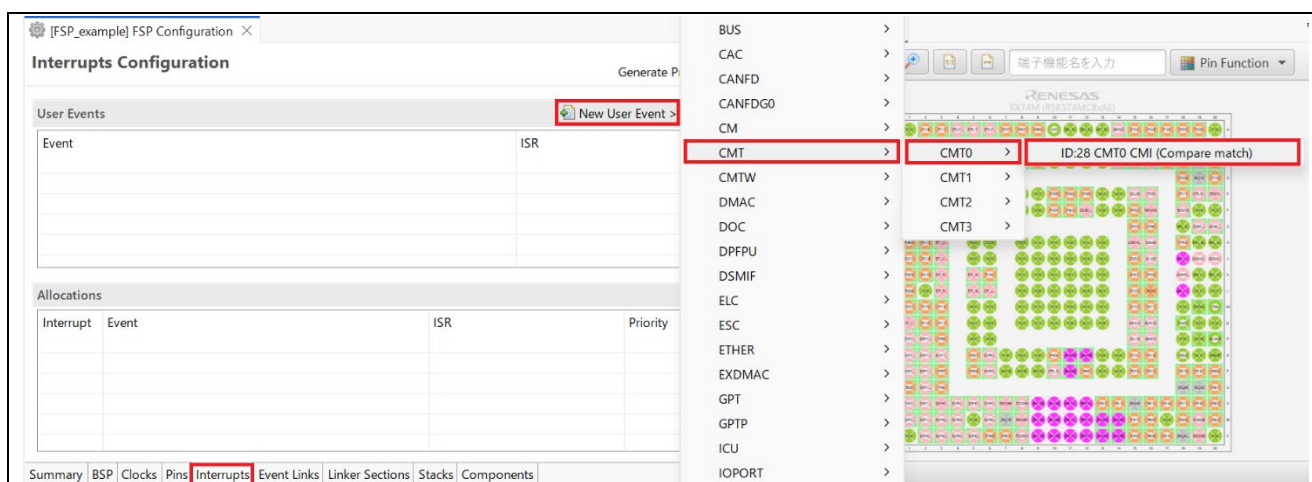


図 5-19 [Interrupts Configuration]ページ CMT0 の割り込み要因の選択

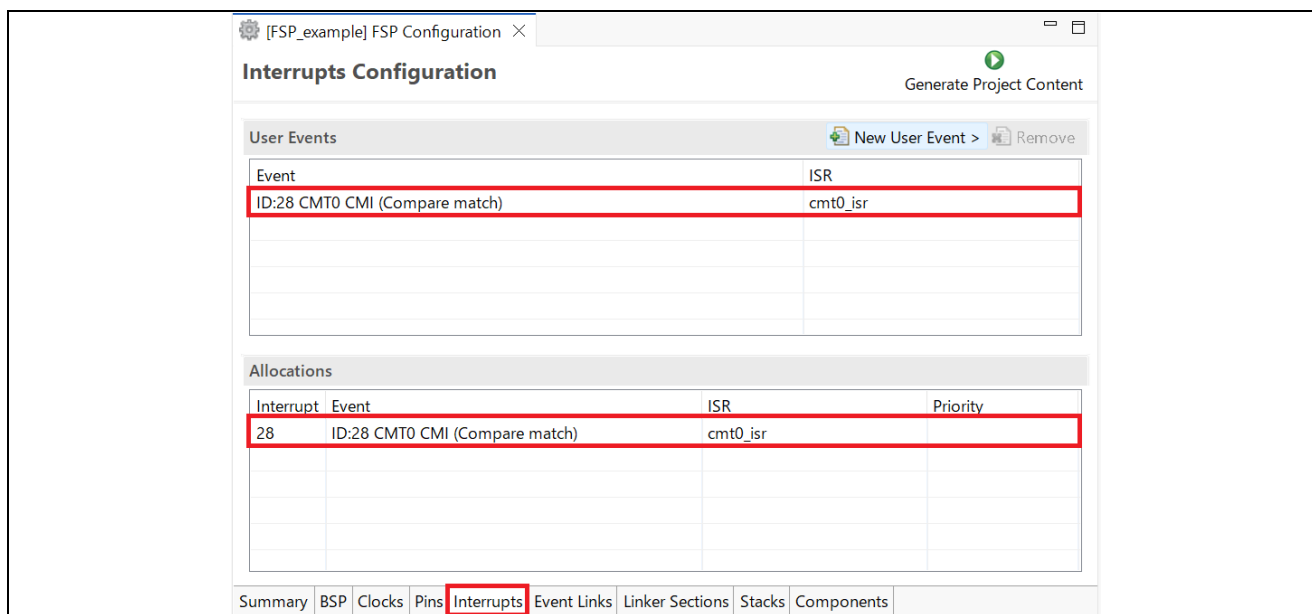


図 5-20 [Interrupts Configuration]ページ CMT0 割り込み追加

生成した後の vector_data.h では以下のように割り込み関数の宣言が出力されます。

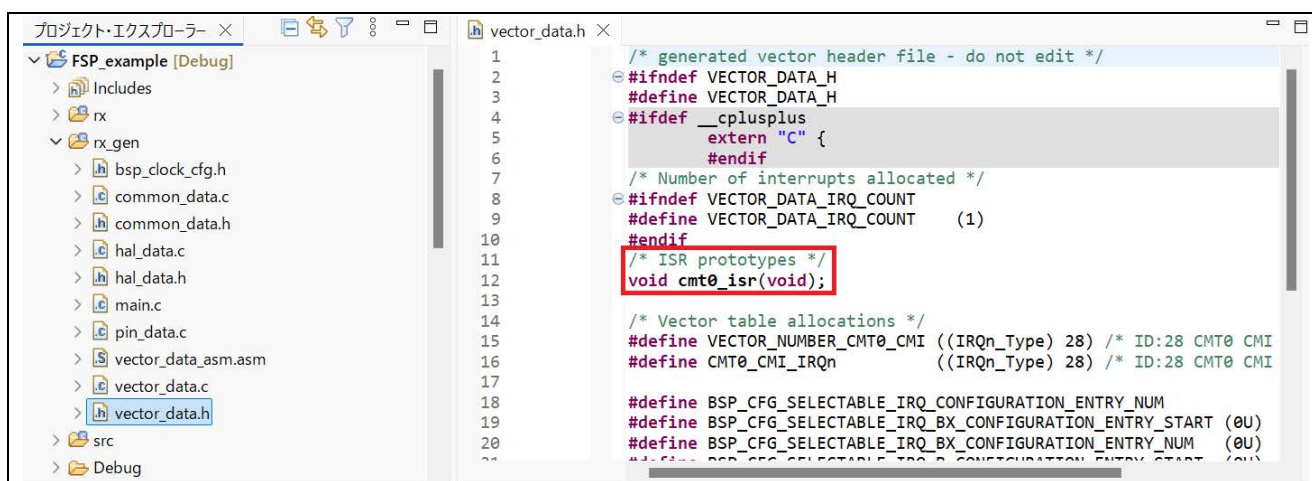


図 5-21 vector_data.h CMT0 割り込み関数の宣言

生成されるのは割り込み関数の宣言のみであるため、割り込み関数の実体はユーザ自身で作成する必要があります。

移行元のプロジェクトでの割り込み関数宣言は FSP プロジェクトの BSP のベクタテーブル定義および割り込み宣言と競合しますので直接移植しないでください。

5.6 端子設定の移行

FSP プロジェクトではプロジェクト生成時にプロジェクトに自動的に端子設定のためのモジュールである IOPORT モジュールが組み込まれます。

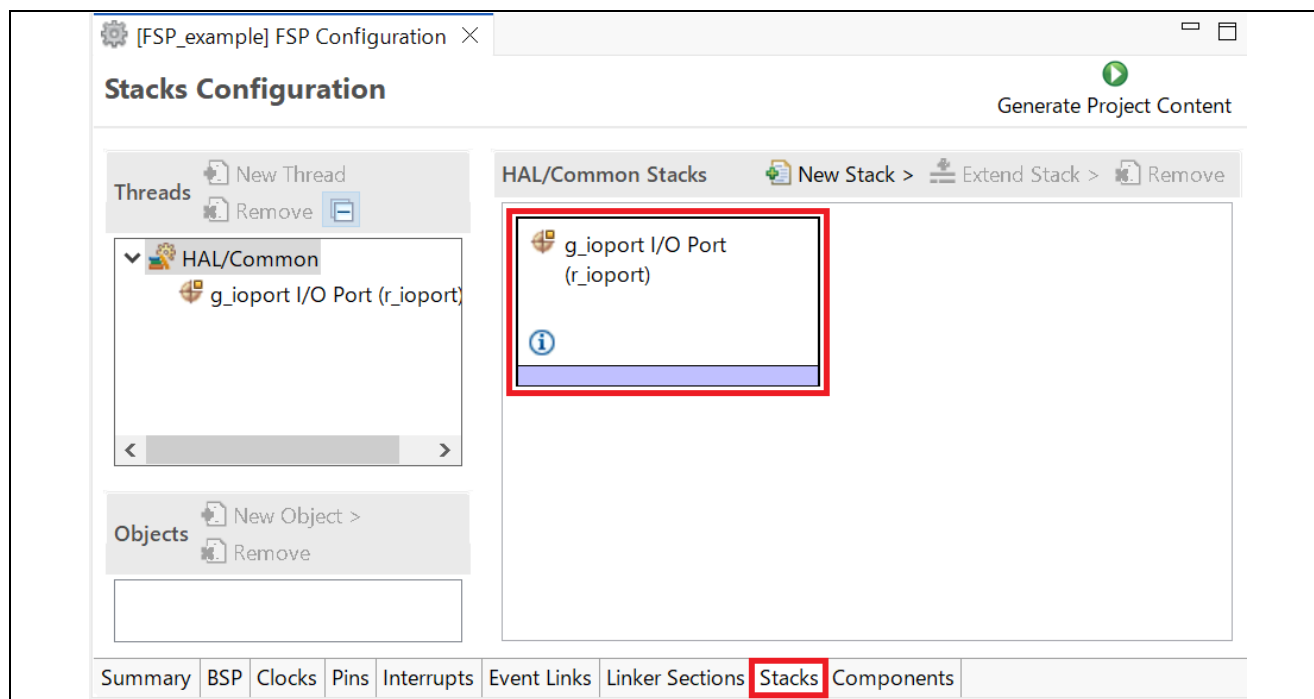


図 5-22 [Stacks Configuration]ページ IOPORT モジュール

各端子の設定は[Pin Configuration]ページから行うことができます。

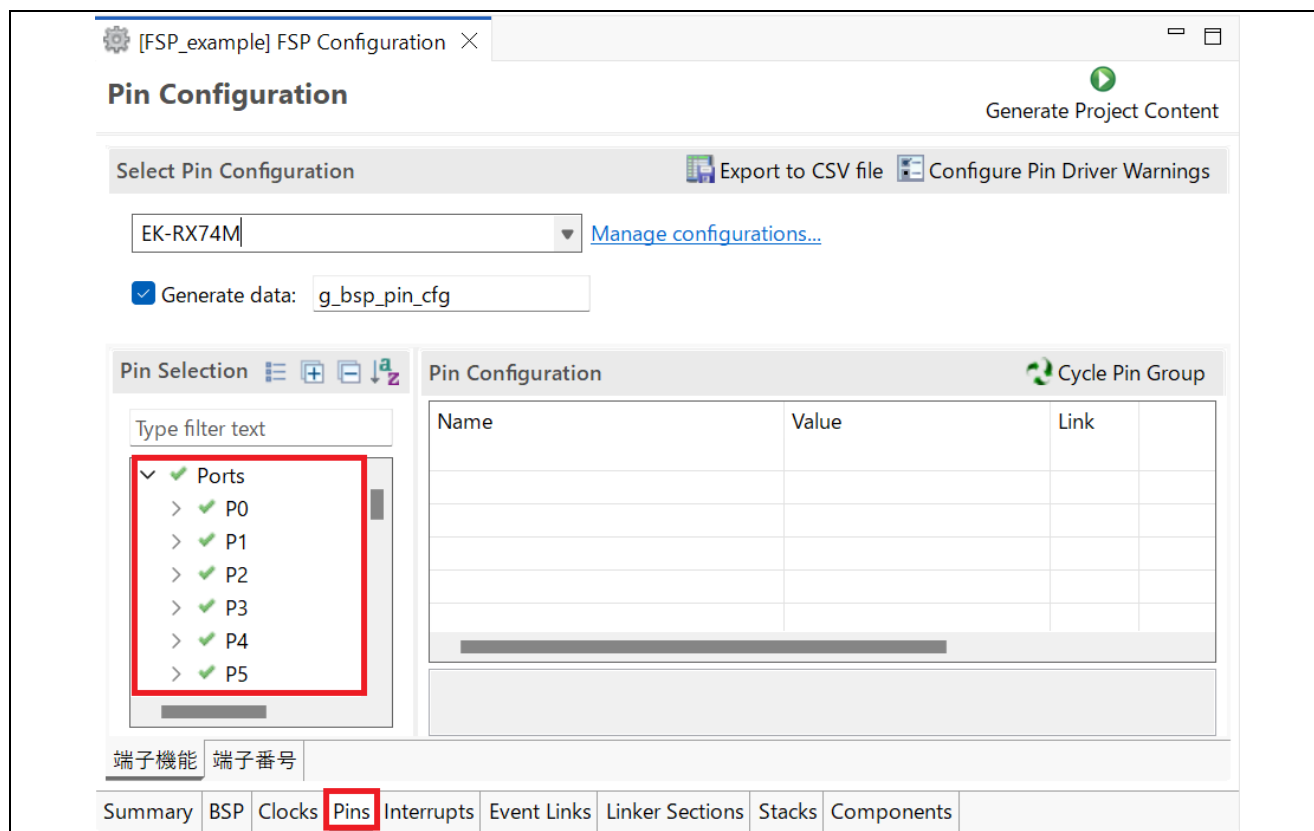


図 5-23 [Pin Configuration]ページ

[Pin Configuration] ページの各ポートを選択し、[Pin Configuration] から設定を行った後

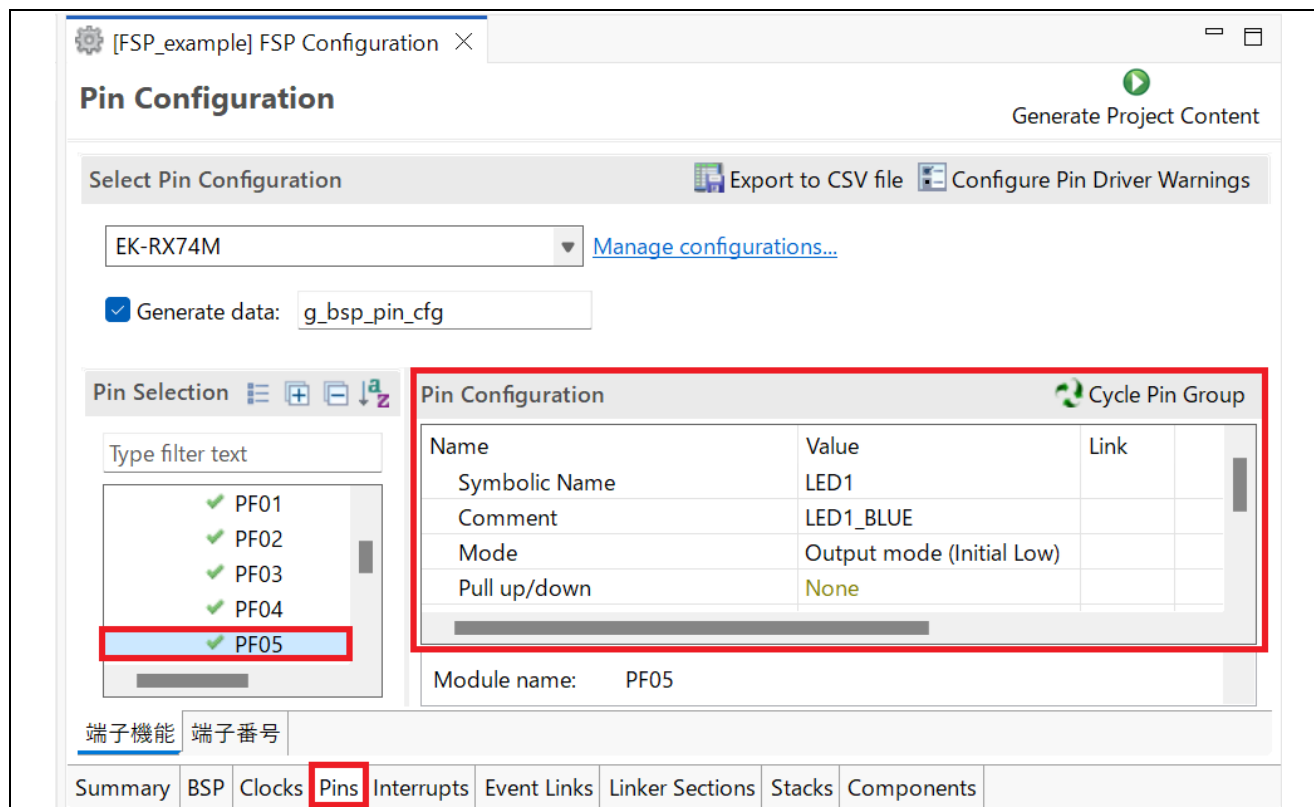


図 5-24 [Pin Configuration]

[Generate Project Content] ボタンをクリックすることで端子設定に関する定義が pin_data.c、common_data.h、common_data.c に出力されます。

rx_gen¥pin_data.c

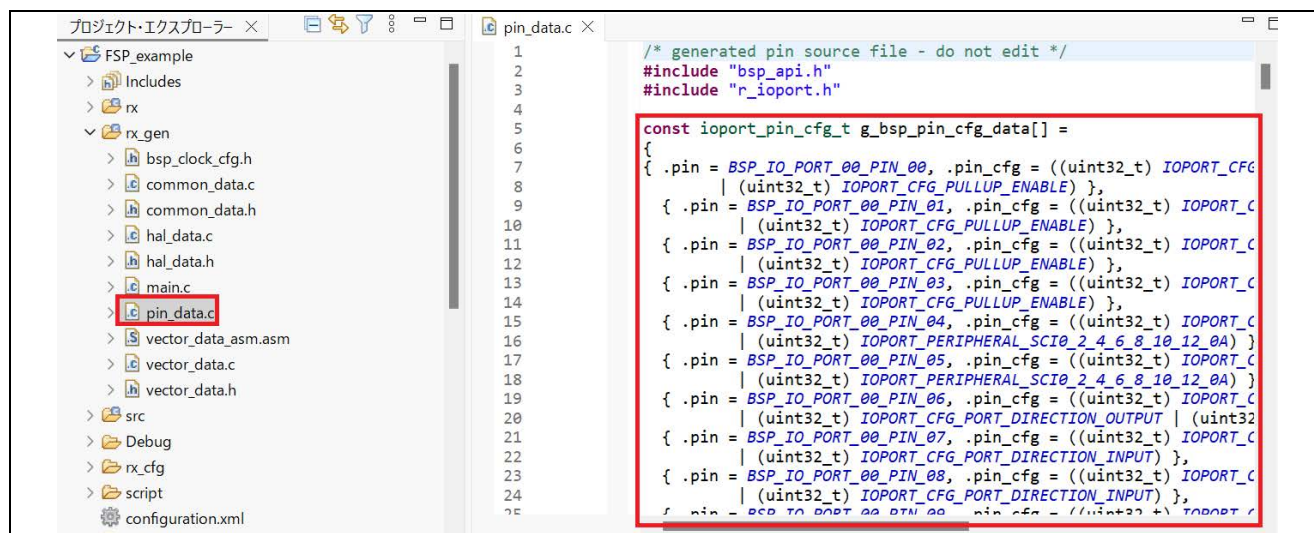


図 5-25 pin_data.c の定義

rx_gen¥common_data.h

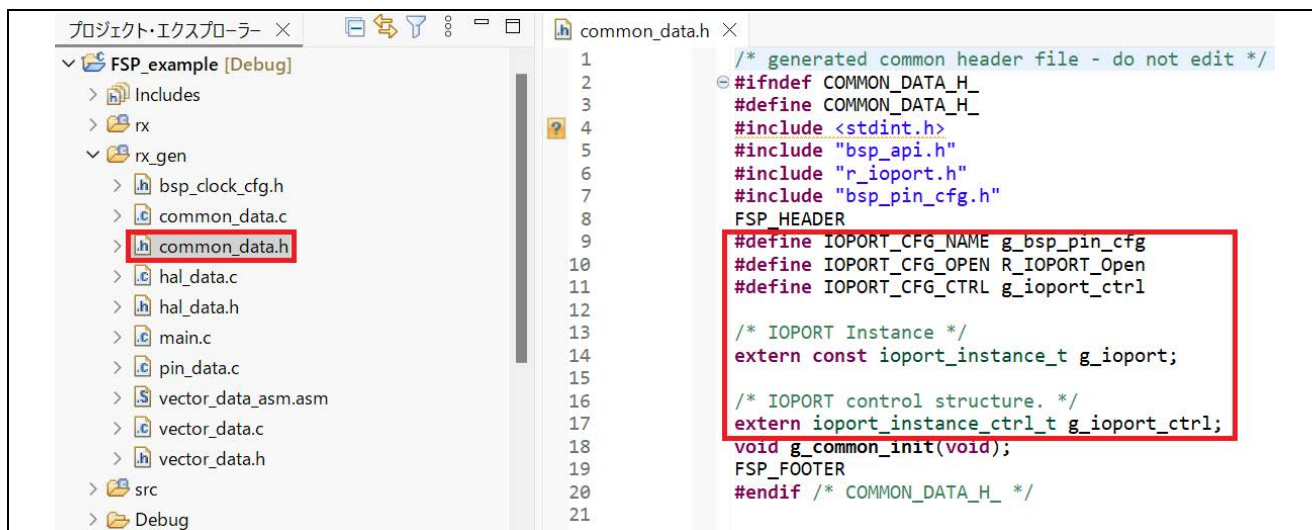


図 5-26 common_data.h の定義

rx_gen¥common_data.c

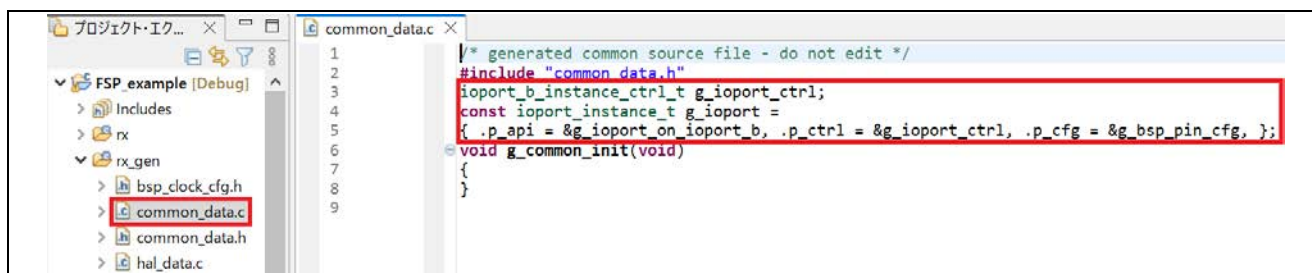


図 5-27 common_data.c の定義

上記定義を用いてスタートアップ処理の中で端子の初期設定が実行されます。BSP のスタートアップ処理の概要については 4.2.3 FSP BSP の初期設定を参照ください。端子設定を変更する場合には IOPORT モジュールの API 関数を使用してください。

5.7 ソースコードの移植

FIT プロジェクトのソースコードの必要部分を hal_entry.c に追加するか、別ファイルとしてプロジェクト直下の src フォルダに追加してください。src フォルダ hal_entry.c については 2.6 RX FSP プロジェクトのファイル構成を参照してください。

5.8 iodef.h の追加

移行先のプロジェクトと同じデバイスで移行先プロジェクトとは別に新規の Executable プロジェクトを作成し、生成されたプロジェクト内の generate フォルダにある iodef.h を移行先プロジェクト直下の src フォルダに追加してください。

5.9 FSP モジュールの処理への置き換え

5.9.1 FIT モジュールの処理の FSP モジュールの処理への置き換え

FIT モジュールの API の仕様、FSP モジュールの API の仕様を確認し、FIT モジュールの処理を FSP モジュールの処理に置き換えてください。

以下は CMT モジュールの場合の置き換え例です。

CMT FIT モジュールの処理

```
bool      cmt_result;
uint32_t  cmt_hdl;

/* Create a timer tick with callback at a rate of 2Hz. */
cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl);
```

CMT FSP モジュールの処理

```
fsp_err_t err = FSP_SUCCESS;

/* Create a timer tick with callback at a rate of 2Hz. */
/* Called the R_CMT_Open to initialize the module for CMT0. */
err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);

/* Handle any errors. This function should be defined by the user. */
assert(FSP_SUCCESS == err);

/* Called the R_CMT_Start to start the timer for CMT0. */
(void) g_timer0.p_api->start(&g_timer0_ctrl);
```

5.9.2 コード生成の処理の FSP モジュールの処理への置き換え

コード生成の API 関数の仕様、FSP モジュールの API の仕様を確認し、コード生成の処理を FSP モジュールの処理に置き換えてください。

以下はコード生成のコンペアマッチタイマ用 API 関数の場合の置き換え例です。

コンペアマッチタイマ用 API 関数の処理

```
R_Config_CMT0_Start();
```

CMT FSP モジュールの処理

```
fsp_err_t err = FSP_SUCCESS;

/* Create a timer tick with callback at a rate of 2Hz. */
/* Called the R_CMT_Open to initialize the module for CMT0. */
err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);

/* Handle any errors. This function should be defined by the user. */
assert(FSP_SUCCESS == err);

/* Called the R_CMT_Start to start the timer for CMT0. */
(void) g_timer0.p_api->start(&g_timer0_ctrl);
```

6. FIT アプリケーションの FSP プロジェクトへの移行例

FIT モジュールを使用したプロジェクトを例として FSP プロジェクトへ移行する方法について説明します。

6.1 CMT FIT のサンプルプログラムを使用した移行例

CMT のデモプロジェクトのサンプルプログラムを FSP プロジェクトに移行する手順について説明します。参照するサンプルプログラムは [CMT FIT](#) の cmt_demo_rskrx72m プロジェクト（以下移行元プロジェクト）を例とします。

6.1.1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

移行元プロジェクトのフォルダ・ファイル構成と移行対象を以下に示します。main.c および rskrx72m.h が移行対象です。

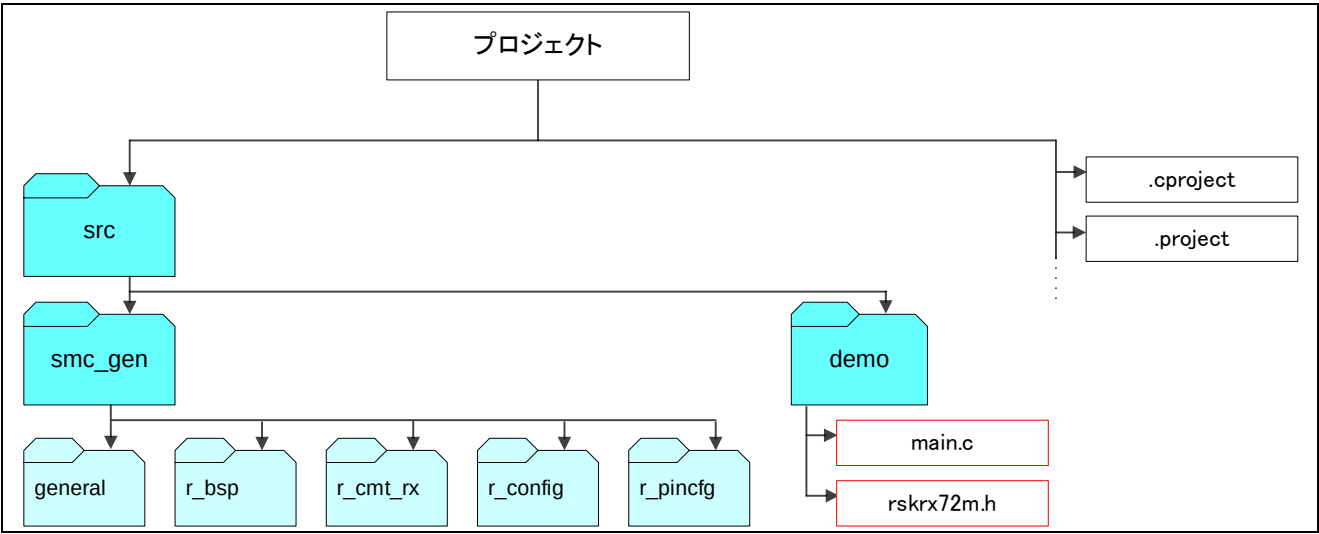


図 6-1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

6.1.2 移行元のプロジェクトの処理概要

移行元プロジェクトの main.c は以下のとおりです。

src¥demo¥main.c

```

/*****
Includes    <System Includes> , "Project Includes"
*****/
#include "r_cmt_rx_if.h"
#include "rskrx72m.h"

/*****
Private global variables and functions
*****/

void main(void);
void cmt_cb(void *pdata);
```

```
/* *****  
* Function Name : main  
* Description   : Assigns a CMT timer channel to the callback function cmt_cb.  
* Arguments    : none  
* Return value  : none  
* *****/  
void main()  
{  
    bool        cmt_result;  
    uint32_t    cmt_hdl;  
  
    /* Enable LEDs. */  
    /* Start with LEDs off. */  
    LED0 = LED_OFF;  
    LED1 = LED_OFF;  
  
    /* Set LED pins as outputs. */  
    LED0_PDR = 1;  
    LED1_PDR = 1;  
  
    /* Create a timer tick with callback at a rate of 2Hz. */  
    cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl);  
    cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl);  
  
    if (false == cmt_result)  
    {  
        R_BSP_NOP(); //Handle your error here.  
    }  
  
    /* End of demo, spin here */  
    while(1);  
}  
  
/* *****  
* Function Name : cmt_cb  
* Description   : This is the callback function that was assigned to the CMT  
*                 driver in main. It is called by the CMT driver at a 2 Hz rate  
*                 and toggles LED0 with each call generating 1Hz blink rate.  
* Arguments    : pdata - pointer to void  
* Return value  : none  
* *****
```

```

*****/
void cmt_cb(void *pdata)
{
    uint32_t channel_num;

    /* This is an example of how to dereference the channel number when
     * more than one channel is used. Channel numbers can be 0 to 3. */
    channel_num = *((uint32_t *)pdata);

    switch(channel_num)
    {
        case 0:
            LED0 = ~LED0; // Toggle LED0 with each callback
            break;

        case 1:
            LED1 = ~LED1; // Toggle LED1 with each callback
            break;

        default:
            break;
    }
}

```

デモに関連するファイルとして rskrx72m.h がインクルードされています。rskrx72m.h の内容は以下のとおりで iodefne.h の定義が用いられています。

src¥demo¥rskrx72m.h

```

#ifndef RSKRX72M_H
#define RSKRX72M_H

/* Local defines */
#define LED_ON          (0)
#define LED_OFF         (1)
#define SET_BIT_HIGH    (1)
#define SET_BIT_LOW     (0)
#define SET_BYTE_HIGH   (0xFF)
#define SET_BYTE_LOW    (0x00)

/* Switches */
#define SW_ACTIVE        0
#define SW1              PORT4.PIDR.BIT.B5

```

```

#define SW2          PORT4.PIDR.BIT.B4
#define SW3          PORT0.PIDR.BIT.B7
#define SW1_PDR      PORT4.PDR.BIT.B5
#define SW2_PDR      PORT4.PDR.BIT.B4
#define SW3_PDR      PORT0.PDR.BIT.B7
#define SW1_PMR      PORT4.PMR.BIT.B5
#define SW2_PMR      PORT4.PMR.BIT.B4
#define SW3_PMR      PORT0.PMR.BIT.B7

/* LEDs */
#define LED0          PORT4.PODR.BIT.B2
#define LED1          PORTH.PODR.BIT.B0
#define LED2          PORTN.PODR.BIT.B4
#define LED3          PORT8.PODR.BIT.B5
#define LED0_PDR      PORT4.PDR.BIT.B2
#define LED1_PDR      PORTH.PDR.BIT.B0
#define LED2_PDR      PORTN.PDR.BIT.B4
#define LED3_PDR      PORT8.PDR.BIT.B5

#endif /* RSKRX72M_H */

```

移行元のプログラムでは R_CMT_CreatePeriodic 関数を用いて空きのある CMT のチャンネルを取得し、2Hz 周期で起動することを 2 回実行します。これにより CMT0 と CMT1 を 2Hz 周期でコンペアマッチ割り込みを発生させ、LED を点滅させる内容となっています。

6.1.3 移行先のプロジェクトの作成

5 FSP 以前のプロジェクトから FSP プロジェクトに移行する手順で示した手順に則り、移行するプロジェクトを作成します。5.1 FSP プロジェクトの新規作成～5.4 モジュールの移行 (FIT→FSP) を参考に新規に作成した FSP プロジェクトをベースにします。

6.1.4 CMT モジュールの追加

FSP の CMT モジュールには CMT FIT モジュールの R_CMT_CreatePeriodic 関数にある空きチャンネルから使用するチャンネルを取得するという機能はありません。したがってあらかじめ使用するチャンネルを決めて定義しておく必要があります。

CMT モジュールは 5.4 モジュールの移行 (FIT→FSP) で示したものを参考にプロパティのコールバック関数と割り込み優先レベルを変更した上で以下の方法でプロジェクトに CMT モジュールを 2 チャンネル分 (CMT0 と CMT1)追加してください。

[New Stack] ボタンから [Timers] → [Timer, Compare Match Timer (r_cmt)]の選択を 2 回実行し、CMT0 用の g_timer0 と CMT1 用の g_timer1 をプロジェクトに追加します。

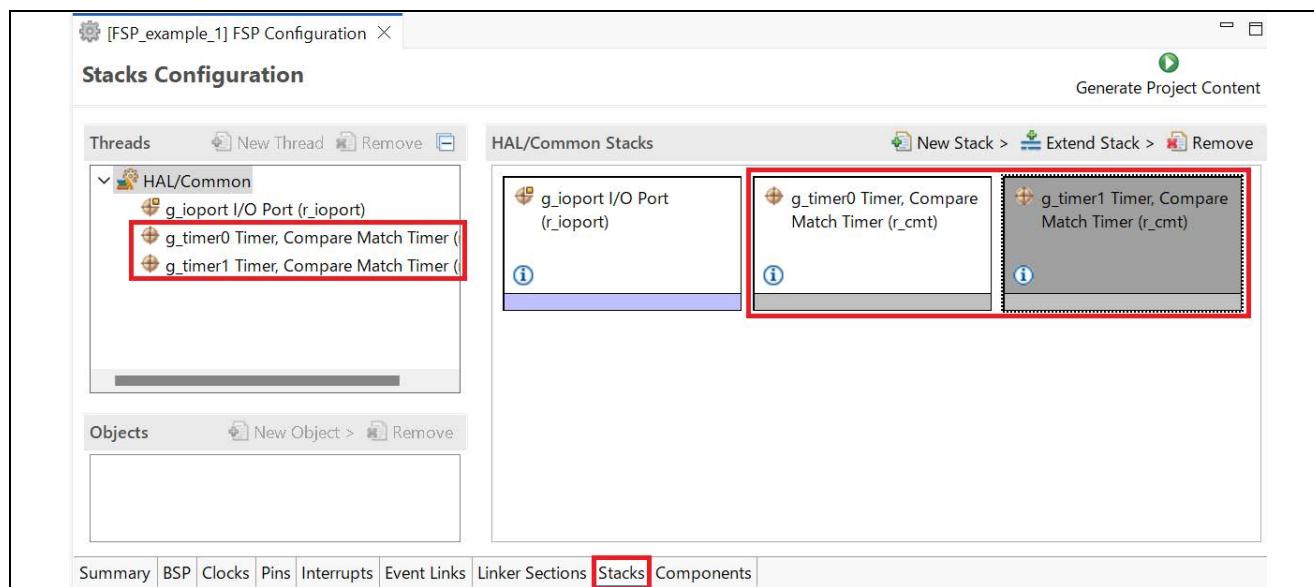


図 6-2 [Stacks Configuration]ページ CMT モジュール(r_cmt)の追加(CMT0、CMT1)

g_timer0 は CMT0 用としてプロパティでは以下のように Period と Period Unit は CMT FIT のサンプルプログラムに合わせるように 2Hz(Hertz)、コールバック関数名と割り込み優先レベルを設定します。

プロパティ × 問題 スマート・ブラウザー		
g_timer0 Timer, Compare Match Timer (r_cmt)		
Settings	プロパティ	値
	▼ Common	
	Parameter Checking	Default (BSP)
	▼ Module g_timer0 Timer, Compare Match Timer (r_cmt)	
	▼ General	
	Name	g_timer0
	Channel	0
	Mode	Periodic
	Period	2
	Period Unit	Hertz
	▼ Interrupts	
	Callback	cmt0_callback
	Compare Match Interrupt Priority	Priority 15

図 6-3 r_cmt g_timer0(CMT0) プロパティ

g_timer1 は CMT1 用としてプロパティは以下のようにチャンネルを 1 にし、Period と Period Unit は CMT FIT のデモに合わせるように 2Hz(Hertz)、コールバック関数名と割り込み優先レベルを設定します。

プロパティ × 問題 スマート・ブラウザー		
g_timer1 Timer, Compare Match Timer (r_cmt)		
Settings	プロパティ	値
	Parameter Checking	Default (BSP)
	▼ Module g_timer1 Timer Driver on r_cmt	
	▼ General	
	Name	g_timer1
	Channel	1
	Mode	Periodic
	Period	2
	Period Unit	Hertz
	▼ Interrupts	
	Callback	cmt1_callback
	Compare Match Interrupt Priority	Priority 15

図 6-4 r_cmt g_timer1(CMT1) プロパティ

6.1.5 移行後のプロジェクトのフォルダ・ファイル構成と変更対象

移行後のプロジェクトのフォルダ・ファイル構成と変更対象を以下に示します。hal_entry.c および移植された rskrx72m.h が変更対象です。iodefine.h は新規の Executable プロジェクトで生成されたファイルを追加します。

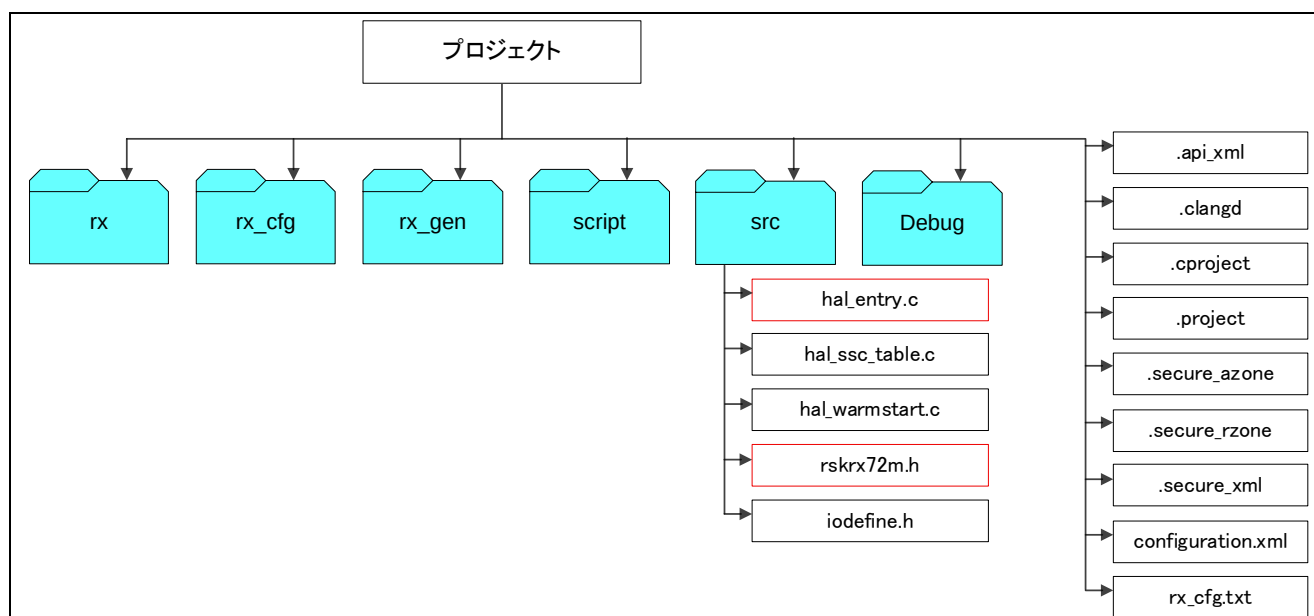


図 6-5 移行後プロジェクトのフォルダ・ファイル構成と変更対象

6.1.6 移行のポイント

5.7 ソースコードの移植に則り、main.c の main 関数本体の処理と CMT のコールバック関数を hal_entry.c に移植します。rskrx72m.h を移植します。rskrx72m.h が iodef.h の定義を使用した定義を含むので 5.8 iodef.h の追加に則り iodef.h を追加します。デバイスが変わるため、使用している定義を一部変更しています。また、FSP が提供するソースコードの定義と競合する定義の競合を回避するため、一部定義名を変更しています。5.9 FSP モジュールの処理への置き換えを参考に FIT モジュールの処理を FSP の処理へ置き換えます。

6.1.7 変更対象のファイルの置き換え後の処理

変更した最終的な hal_entry.c および rskrx72m.h は以下のようになります。

hal_entry.c （赤字が最終的に追加した部分となります。）

```
#include "hal_data.h"
#include "rskrx72m.h"

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err = FSP_SUCCESS;

    /* Enable LEDs. */
    /* Start with LEDs off. */
    LED0 = LED_OFF;
    LED1_PODR = LED_OFF;

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

    /* Create a timer tick with callback at a rate of 2Hz. */
    /* Called the R_CMT_Open to initialize the module for CMT0. */
    err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg);

    /* Handle any errors. This function should be defined by the user. */
```

```
assert(FSP_SUCCESS == err);

/* Called the R_CMT_Start to start the timer for CMT0. */
(void) g_timer0.p_api->start(&g_timer0_ctrl);

/* Called the R_CMT_Open to initialize the module for CMT1. */
err = g_timer1.p_api->open(&g_timer1_ctrl, &g_timer1_cfg);

/* Handle any errors. This function should be defined by the user. */
assert(FSP_SUCCESS == err);

/* Called the R_CMT_Start to start the timer for CMT1. */
(void) g_timer1.p_api->start(&g_timer1_ctrl);

/* End of demo, spin here */
while(1);

#if BSP_PZ_SECURE_BUILD
/* Enter non-secure code */
R_BSP_NonSecureEnter();
#endif
}

void cmt0_callback(timer_callback_args_t *p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */
        LED0 = ~LED0; // Toggle LED0 with each callback
    }
}

void cmt1_callback(timer_callback_args_t *p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */
        LED1_PODR = ~LED1_PODR; // Toggle LED1 with each callback
    }
}
```

rskrx72m.h (赤字が最終的に変更した部分となります。)

```
#include "iodef.h"

#ifndef RSKRX72M_H
#define RSKRX72M_H

/* Local defines */
#define LED_ON          (0)
#define LED_OFF         (1)
#define SET_BIT_HIGH    (1)
#define SET_BIT_LOW     (0)
#define SET_BYTE_HIGH   (0xFF)
#define SET_BYTE_LOW    (0x00)

/* Switches */
#define SW_ACTIVE        0
#define SW1_PIDR         PORT4.PIDR.BIT.B5
#define SW2               PORT4.PIDR.BIT.B4
#define SW3              PORT0.PIDR.BIT.B7
#define SW1_PDR          PORT4.PDR.BIT.B5
#define SW2_PDR          PORT4.PDR.BIT.B4
#define SW3_PDR          PORT0.PDR.BIT.B7
#define SW1_PMR          PORT4.PMR.BIT.B5
#define SW2_PMR          PORT4.PMR.BIT.B4
#define SW3_PMR          PORT0.PMR.BIT.B7

/* LEDs */
#define LED0              PORTF.PODR.BIT.PODR06
#define LED1_PODR        PORT8.PODR.BIT.PODR07
#define LED2_PODR        PORTN.PODR.BIT.B4
#define LED3_PODR        PORT8.PODR.BIT.B5
#define LED0_PDR         PORTF.PDR.BIT.PDR06
#define LED1_PDR         PORT8.PDR.BIT.PDR07
#define LED2_PDR         PORTN.PDR.BIT.B4
#define LED3_PDR         PORT8.PDR.BIT.B5

#endif /* RSKRX72M_H */
```

移行後のプログラムでは R_CMT_Open 関数と R_CMT_Start 関数を用いて CMT0 と CMT1 のチャンネルを 2Hz 周期で起動することを 2 回実行します。これにより CMT0 と CMT1 を 2Hz 周期でコンペアマッチ割り込みを発生させ、LED を点滅させる内容となります。

6.1.8 本移行例における FIT モジュールの処理の FSP モジュールへの処理の置き換え

移行前の FIT モジュールの処理と移行し置き換え後の FSP モジュールの処理の比較を以下に示します。

6.1.8.1 メイン処理の比較

移行前 (FIT)	移行後 (FSP)
<pre> void main() { bool cmt_result; uint32_t cmt_hdl; /* Enable LEDs. */ /* Start with LEDs off. */ LED0 = LED_OFF; LED1 = LED_OFF; /* Set LED pins as outputs. */ LED0_PDR = 1; LED1_PDR = 1; /* Create a timer tick with callback at a rate of 2Hz. */ cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl); cmt_result = R_CMT_CreatePeriodic(2, cmt_cb, &cmt_hdl); if (false == cmt_result) { R_BSP_NOP(); //Handle your error here. } /* End of demo, spin here */ while(1); } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; /* Enable LEDs. */ /* Start with LEDs off. */ LED0 = LED_OFF; LED1_PODR = LED_OFF; /* Set LED pins as outputs. */ LED0_PDR = 1; LED1_PDR = 1; /* Create a timer tick with callback at a rate of 2Hz. */ /* Called the R_CMT_Open to initialize the module for CMT0. */ err = g_timer0.p_api->open(&g_timer0_ctrl, &g_timer0_cfg); /* Handle any errors. This function should be defined by the user. */ assert(FSP_SUCCESS == err); /* Called the R_CMT_Start to start the timer for CMT0. */ (void) g_timer0.p_api->start(&g_timer0_ctrl); /* Called the R_CMT_Open to initialize the module for CMT1. */ err = g_timer1.p_api->open(&g_timer1_ctrl, &g_timer1_cfg); /* Handle any errors. This function should be defined by the user. */ assert(FSP_SUCCESS == err); /* Called the R_CMT_Start to start the timer for CMT1. */ (void) g_timer1.p_api->start(&g_timer1_ctrl); /* End of demo, spin here */ while(1); #if BSP_PZ_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

図 6-6 メイン処理

6.1.8.2 コールバック関数の比較

移行前 (FIT)	移行後 (FSP)
<pre> void cmt_cb(void *pdata) { uint32_t channel_num; /* This is an example of how to dereference the channel number when more than * one channel is used. Channel numbers can be 0 to 3. */ channel_num = *((uint32_t *)pdata); switch(channel_num) { case 0: LED0 = ~LED0; // Toggle LED0 with each callback break; case 1: LED1 = ~LED1; // Toggle LED1 with each callback break; default: break; } } </pre>	<pre> void cmt0_callback(timer_callback_args_t *p_args) { if (TIMER_EVENT_CYCLE_END == p_args->event) { /* Add application code to be called periodically here. */ LED0 = ~LED0; // Toggle LED0 with each callback } } void cmt1_callback(timer_callback_args_t *p_args) { if (TIMER_EVENT_CYCLE_END == p_args->event) { /* Add application code to be called periodically here. */ LED1_PODR = ~LED1_PODR; // Toggle LED1 with each callback } } </pre>

図 6-7 コールバック関数

6.2 RSCI FIT のサンプルプログラムを使用した移行例

RSCI のデモプロジェクトのサンプルプログラムを FSP プロジェクトに移行する手順について説明します。参照するサンプルプログラムは [RSCI FIT](#) の rsci_demo_rskrx671 プロジェクト（以下移行元プロジェクト）を例とします。

6.2.1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

移行元プロジェクトのフォルダ・ファイル構成と移行対象を以下に示します。main.c および rskrx671.h が移行対象です。

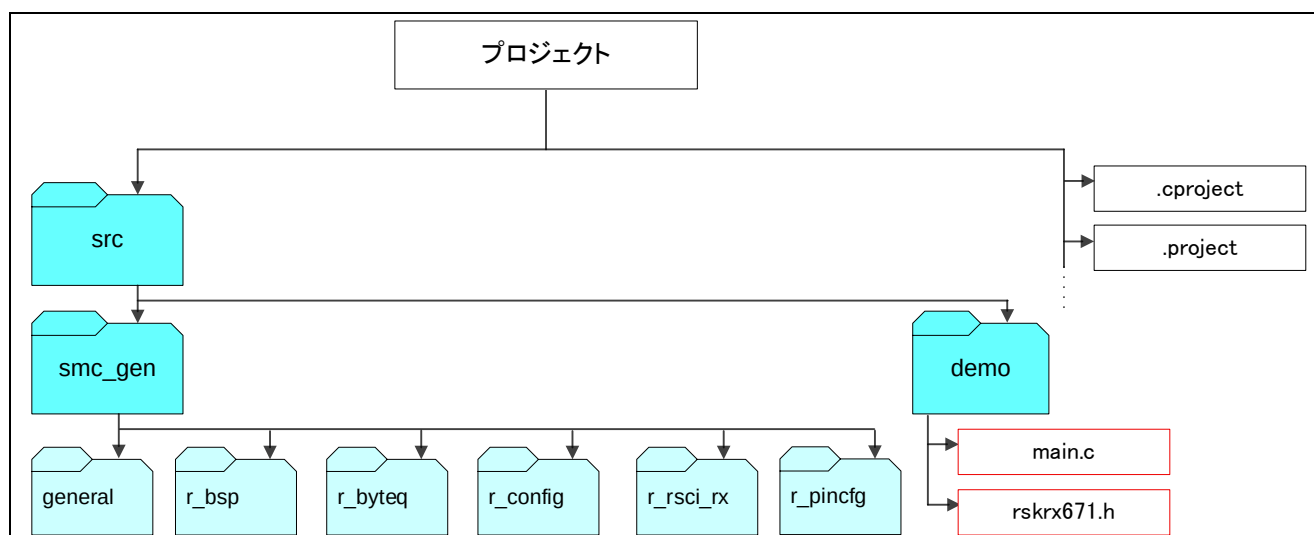


図 6-8 移行元プロジェクトのフォルダ・ファイル構成と移行対象

6.2.2 移行元のプロジェクトの処理概要

移行元プロジェクトの main.c は以下のとおりです。

src¥demo¥main.c

```

/*****
Includes  <System Includes> , "Project Includes"
*****/

#include <stdio.h>          /* For sprintf */
#include "platform.h"       /* Located in the FIT BSP module */
#include "r_rsci_rx_if.h"   /* The RSCI module API interface file. */
#include "r_byteq_if.h"    /* The BYTEQ module API interface file. */
#include "r_rsci_rx_config.h" /* User configurable options for the RSCI
                                module */
#include "rskrx671.h"       /* Include user define */

/*****
Private global variables and functions
*****/

void my_rsci_callback(void *pArgs);
static void rsci_rx671_init_ports(void);

```

```
/* Handle storage. Needs to persist as long as RSCI calls are going to be
   made.*/
static rsci_hdl_t  g_my_rsci_handle;

/*****
 * Function Name: main()
 * Description   : Main loop for demo.
 * Arguments     : none
 * Return Value  : none
 *****/
void main(void)
{
    LED0_PDR = 1;
    rsci_cfg_t  my_rsci_config;
    rsci_err_t  my_rsci_err;
    uint32_t    my_version;
    uint8_t     version_str[12] = {0};
    uint8_t     my_char;

    /* Initialize the I/O port pins for communication on this RSCI channel.
     * This is specific to the MCU and ports chosen. For the RSKRX671 we will
     * use the RSCI channel connected to the USB serial port emulation. */
    rsci_rx671_init_ports();

    /* Set up the configuration data structure for asynchronous (UART)
       operation. */
    my_rsci_config.async.baud_rate    = 115200;
    my_rsci_config.async.clk_src      = RSCI_CLK_INT;
    my_rsci_config.async.data_size    = RSCI_DATA_8BIT;
    my_rsci_config.async.parity_type  = RSCI_NONE_PARITY;
    my_rsci_config.async.msb_first    = false;
    my_rsci_config.async.invert_data  = false;
    my_rsci_config.async.stop_bits    = RSCI_STOPBITS_1;
    my_rsci_config.async.int_priority = 3;    /* 1=lowest, 15=highest */

    /* OPEN ASYNC CHANNEL
     * Provide address of the config structure,
     * the callback function to be assigned,
     * and the location for the handle to be stored.*/
```

```
my_rsci_err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_ASYNC, &my_rsci_config,
                          my_rsci_callback, &g_my_rsci_handle);

/* If there were an error this would demonstrate error detection of API
   calls. */
if (RSCI_SUCCESS != my_rsci_err)
{
    R_BSP_NOP();    /* Your error handling code would go here. */
}

/* Get module software version using the API function provided. */
my_version = R_RSCI_GetVersion();

/* Demonstrate parsing and formatting of version number. */
sprintf((char *)version_str, "RSCIV%1.0hu.%2.2hu%r%r\n",
        ((my_version >> 16)&0xf), (my_version & 0xff));

printf("Initializes the RSCI successfully%r%r\n", version_str);

/* Wait to get a character from the terminal so we know that it is ready. */
do
{
    my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1);
} while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err);

/* Now send the version string out through the serial port to the
   terminal. */
R_RSCI_Send(g_my_rsci_handle, version_str, 12);

/* Repeat forever. */
while(1)
{
    /* Read next character from the serial port receive queue.
       * Repeat read if needed until data is available in the queue. */
    do
    {
        my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1);
    } while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err);

    /* Echo 1 character of the received data back to the sender. */
```

```

        printf("%c", (char)(my_char));
        R_RSCI_Send(g_my_rsci_handle, &my_char, 1);
    }
} /* End of function main. */

/*****
 * Function Name: my_rsci_callback
 * Description  : This is a template for an RSCI Async Mode callback function.
 * Arguments    : pArgs -
 *                pointer to rsci_cb_args_t structure cast to a void. Structure
 *                contains event and associated data.
 * Return Value : none
 *****/
void my_rsci_callback(void *pArgs)
{
    rsci_cb_args_t  *args;

    args = (rsci_cb_args_t *)pArgs;

    if (args->event == RSCI_EVT_RX_CHAR)
    {
        /* From RXI interrupt; received character data is in args->byte */
        LED0 = ~LED0;          // Toggle LED to demonstrate callback execution.
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_RXBUF_OVFL)
    {
        /* From RXI interrupt; rx queue is full; 'lost' data is in args->byte
           You will need to increase buffer size or reduce baud rate */
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_OVFL_ERR)
    {
        /* From receiver overflow error interrupt; error data is in args->byte
           Error condition is cleared in calling interrupt routine */
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_FRAMING_ERR)
    {
        /* From receiver framing error interrupt; error data is in args->byte

```

```

        Error condition is cleared in calling interrupt routine */
        R_BSP_NOP();
    }
    else if (args->event == RSCI_EVT_PARITY_ERR)
    {
        /* From receiver parity error interrupt; error data is in args->byte
           Error condition is cleared in calling interrupt routine */
        R_BSP_NOP();
    }
}

/*****
* Function Name: rsci_rx671_init_ports
* Description  : This function initializes the port pins associated with RSCI
*                channel 10, which is connected to the USB serial port Virtual
*                COM Interface.
* Arguments    : none
* Return Value : none
*****/
static void rsci_rx671_init_ports(void)
{
    /* Use BSP function to unlock the MPC register. */
    R_BSP_RegisterProtectDisable(BSP_REG_PROTECT_MPC);

    /* Initialize channel 2 TXD and RXD pins */
    PORT8.PODR.BIT.B7 = 0;      // Set P87(T16) signal state low
                                // (power-up default)
    MPC.P87PFS.BYTE = 0x2C;     // Assign P87(T16) to TXD010 function
    MPC.P86PFS.BYTE = 0x2C;     // Assign P86(T17) to RXD010 function

    /* Re-lock the MPC register. */
    R_BSP_RegisterProtectEnable(BSP_REG_PROTECT_MPC);

    PORT8.PDR.BIT.B7 = 1;      // Set P87(T16) direction to output
                                // (for use as TXD)
    PORT8.PDR.BIT.B6 = 0;      // Set P86(T17) direction to input
                                // (for use as RXD)
    PORT8.PMR.BIT.B7 = 1;      // Set P87(T16) mode to peripheral operation
    PORT8.PMR.BIT.B6 = 1;      // Set P86(T17) mode to peripheral operation

```

```
    return;  
}
```

デモに関連するファイルとして rskrx671.h がインクルードされています。rskrx671.h の内容は以下のとおりです。

src¥demo¥rskrx671.h

```
#ifndef RSKRX671_H  
#define RSKRX671_H  
  
/* Local defines */  
#define LED_ON          (0)  
#define LED_OFF         (1)  
#define SET_BIT_HIGH    (1)  
#define SET_BIT_LOW     (0)  
#define SET_BYTE_HIGH   (0xFF)  
#define SET_BYTE_LOW    (0x00)  
  
/* Switches */  
#define SW_ACTIVE       0  
#define SW1             PORT9.PIDR.BIT.B1  
#define SW2             PORT9.PIDR.BIT.B2  
#define SW3             PORT0.PIDR.BIT.B7  
#define SW1_PDR         PORT9.PDR.BIT.B1  
#define SW2_PDR         PORT9.PDR.BIT.B2  
#define SW3_PDR         PORT0.PDR.BIT.B7  
#define SW1_PMR         PORT9.PMR.BIT.B1  
#define SW2_PMR         PORT9.PMR.BIT.B2  
#define SW3_PMR         PORT0.PMR.BIT.B7  
  
/* LEDs */  
#define LED0            PORT1.PODR.BIT.B7  
#define LED1            PORTF.PODR.BIT.B5  
#define LED2            PORT0.PODR.BIT.B3  
#define LED3            PORT0.PODR.BIT.B5  
#define LED0_PDR        PORT1.PDR.BIT.B7  
#define LED1_PDR        PORTF.PDR.BIT.B5  
#define LED2_PDR        PORT0.PDR.BIT.B3  
#define LED3_PDR        PORT0.PDR.BIT.B5  
  
#endif /* RSKRX671_H */
```

移行元のプログラムでは UART として設定された RSCI10 を使用しターミナルエミュレーションを実行している PC のターミナルと通信します。ターミナルとの接続確認後に R_RSCI_Receive 関数を実行しターミナルから 1 バイトのデータを受信するのを待機したのち、ターミナルからデータ受信すると RSCI モジュールのバージョン番号の文字列を送信データとして R_RSCI_Send 関数を用いてターミナルに送信します。その後、R_RSCI_Receive 関数による受信待ちと R_RSCI_Send 関数の送信を繰り返すことにより、ターミナルから受信したデータをターミナルに返すループバック動作を行う内容となっています。また、RSCI FIT モジュールは送信バッファおよび受信バッファを BYTEQ FIT モジュールを使用して確保しています。

6.2.3 移行先のプロジェクトの作成

5 FSP 以前のプロジェクトから FSP プロジェクトに移行する手順で示した手順に則り、移行するプロジェクトを作成します。5.1 FSP プロジェクトの新規作成～5.4 モジュールの移行 (FIT→FSP) を参考に新規に作成した FSP プロジェクトをベースにします。

6.2.4 SCI モジュールの追加

以下の方法でプロジェクトに SCI モジュール 1 チャンネル分を追加してください。

[New Stack] ボタンから[Connectivity] → [UART (r_sci_b_uart)]を選択し実行します。

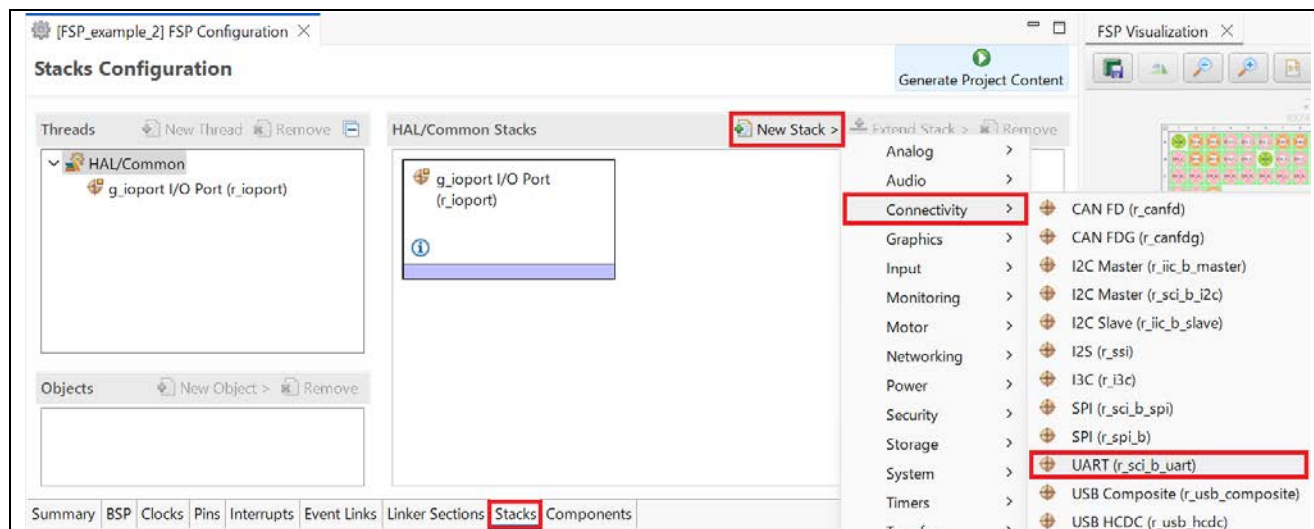


図 6-9 [Stacks Configuration]ページ UART(SCI)の選択

SCI 用の"g_uart0(r_sci_b_uart)"が[Stacks Configuration]ページに追加されます。

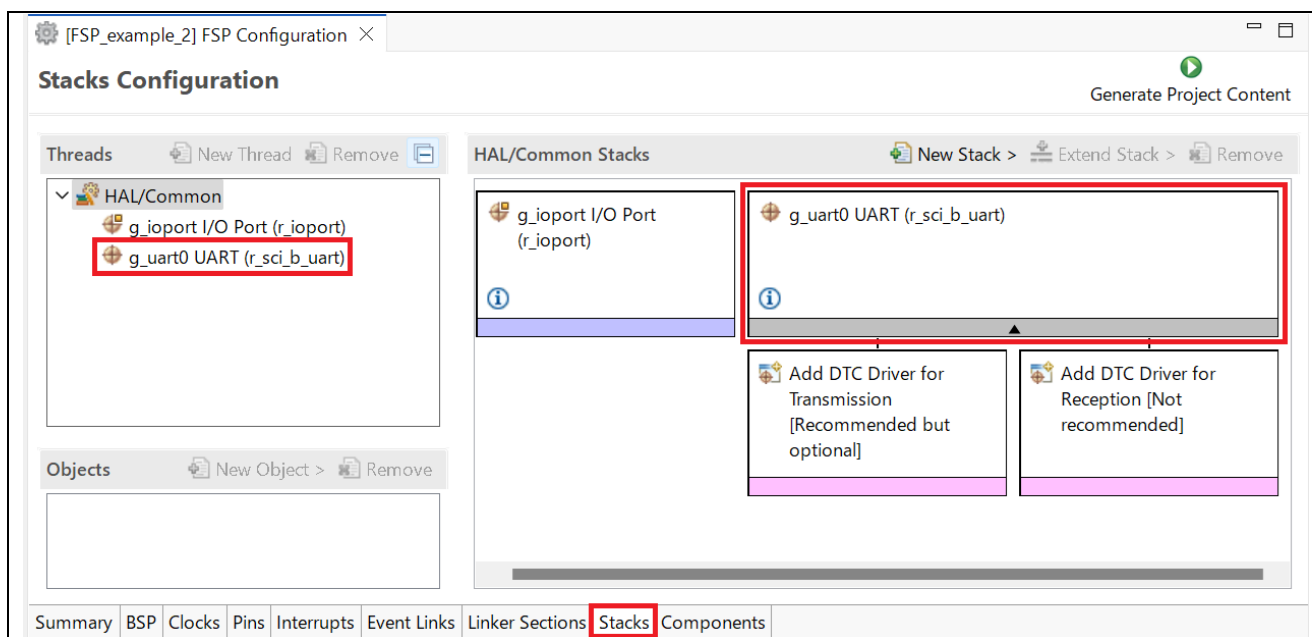


図 6-10 [Stacks Configuration]ページ UART(SCI)モジュール(r_sci_b_uart)の追加

移行元のプロジェクトの RSCI のコンポーネント設定と R_RSCI_Open 関数による設定（115200bps、8 ビットデータ、パリティなし、1 ストップビット、フロー制御なし）に合わせるように SCI モジュールのプロパティを設定します。

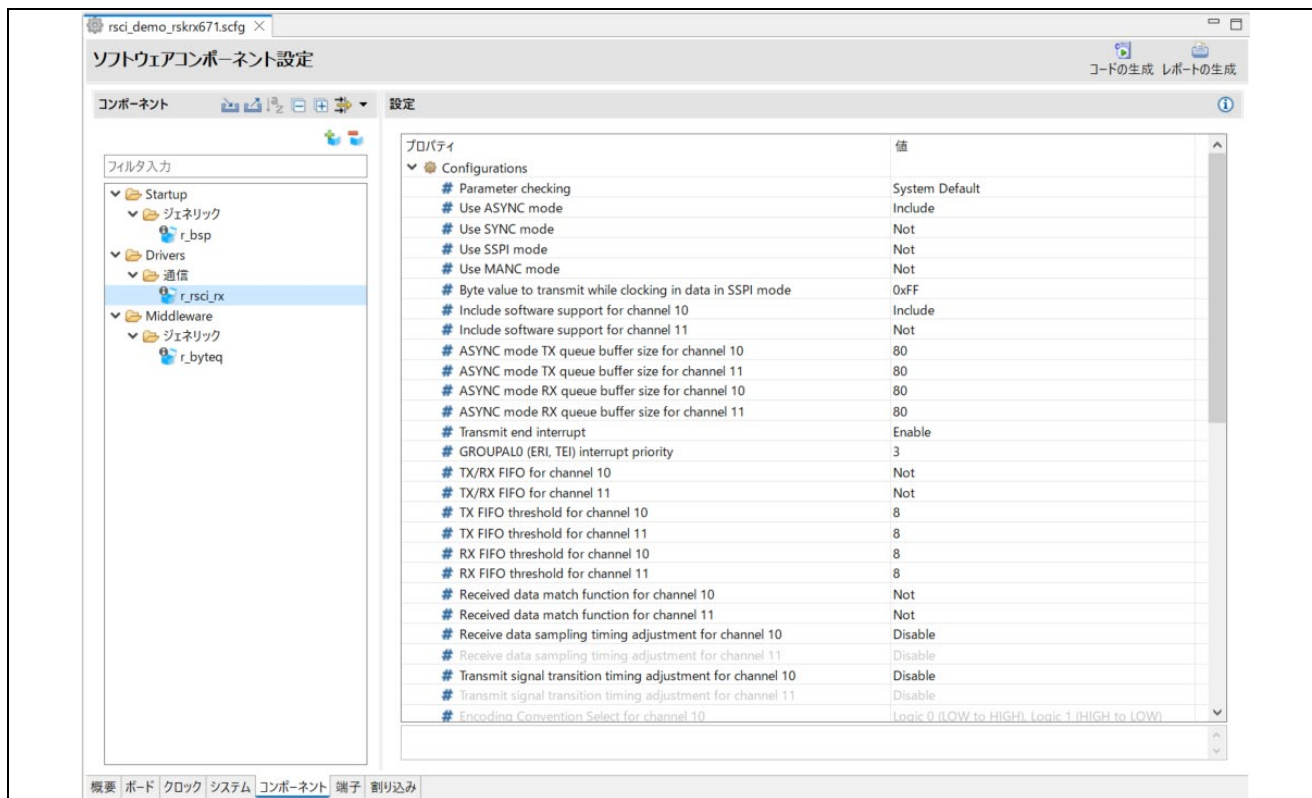


図 6-11 [コンポーネント] r_sci_rx のプロパティ

g_uart0 のプロパティは以下のように設定します。

表 6.1 移行時の設定項目と設定値

No	RSCI コンポーネントでの設定項目および設定値	g_uart0 のプロパティでの設定項目および設定値
1	Parameter checking を System Default に設定。	Parameter checking を Default (BSP)に設定。
2	Use ASYNC mode を Include に設定。	なし
3	Use SYNC mode を Not に設定。	なし
4	Use SSPI mode を Not に設定。	なし
5	Use MANC mode を Not に設定。	なし
6	Byte value to transmit while clocking in data in SSPI mode に 0xFF を指定。	なし
7	Include software support for channel 10 を Include に設定。	General>Channel を 12 に設定。（デバイス・ボード変更により Channel 12 に変更）
8	Include software support for channel 11 を Not に設定。	なし（使用しないものは設定不要）
9	ASYNC mode TX queue buffer size for channel 10 に 80 を指定。	なし
10	ASYNC mode TX queue buffer size for channel 11 に 80 を指定。	なし
11	ASYNC mode RX queue buffer size for channel 10 に 80 を指定。	なし
12	ASYNC mode RX queue buffer size for channel 11 に 80 を指定。	なし
13	Transmit end interrupt を Enable に設定。	なし
14	GROUPAL0 (ERI, TEI) interrupt priority に 3 を指定。	Interrupts>Transmit End Interrupt Priority を 3 に設定。 Interrupts>Error Interrupt Priority を 3 に設定。
15	TX/RX FIFO for channel 10 を Not に設定。	Common>FIFO Support を Disable に設定。
16	TX/RX FIFO for channel 11 を Not に設定。	なし（使用しないものは設定不要）
17	TX FIFO threshold for channel 10 に 8 を指定。	なし
18	TX FIFO threshold for channel 11 に 8 を指定。	なし
19	RX FIFO threshold for channel 10 に 8 を指定。	なし
20	RX FIFO threshold for channel 11 に 8 を指定。	なし
21	Received data match function for channel 10 を Not に設定。	なし
22	Received data match function for channel 11 を Not に設定。	なし
23	Receive data sampling timing adjustment for channel 10 に Disabled を指定。	なし
24	Transmit signal transition timing adjustment for channel 10 に Disabled を指定。	なし

上記の設定と移行前と同じ名前でコールバック関数名を設定した g_uart0 のプロパティは以下のとおりです。

プロパティ × 問題 スマート・ブラウザー		
g_uart0 UART (r_sci_b_uart)		
Settings	プロパティ	値
	▼ Module g_uart0 UART (r_sci_b_uart)	
	▼ General	
	Name	g_uart0
	Channel	12
	Data Bits	8bits
	Parity	None
	Stop Bits	1bit
	▼ Baud	
	Baud Rate	115200
	Baud Rate Modulation	Disabled
	Max Error (%)	5
	> Flow Control	
	> Extra	
	▼ Interrupts	
	Callback	my_rsci_callback
	Receive Interrupt Priority	Priority 3
	Transmit Data Empty Interrupt Priority	Priority 3
	Transmit End Interrupt Priority	Priority 3
	Error Interrupt Priority	Priority 3

図 6-12 UART(SCI)モジュール(r_sci_b_uart)のプロパティ

コールバック関数と割り込み優先レベルの設定により以下のように RSCI12 の割り込みを使用する情報が登録されます。割り込み関数はモジュールに含まれる割り込み関数名となります。

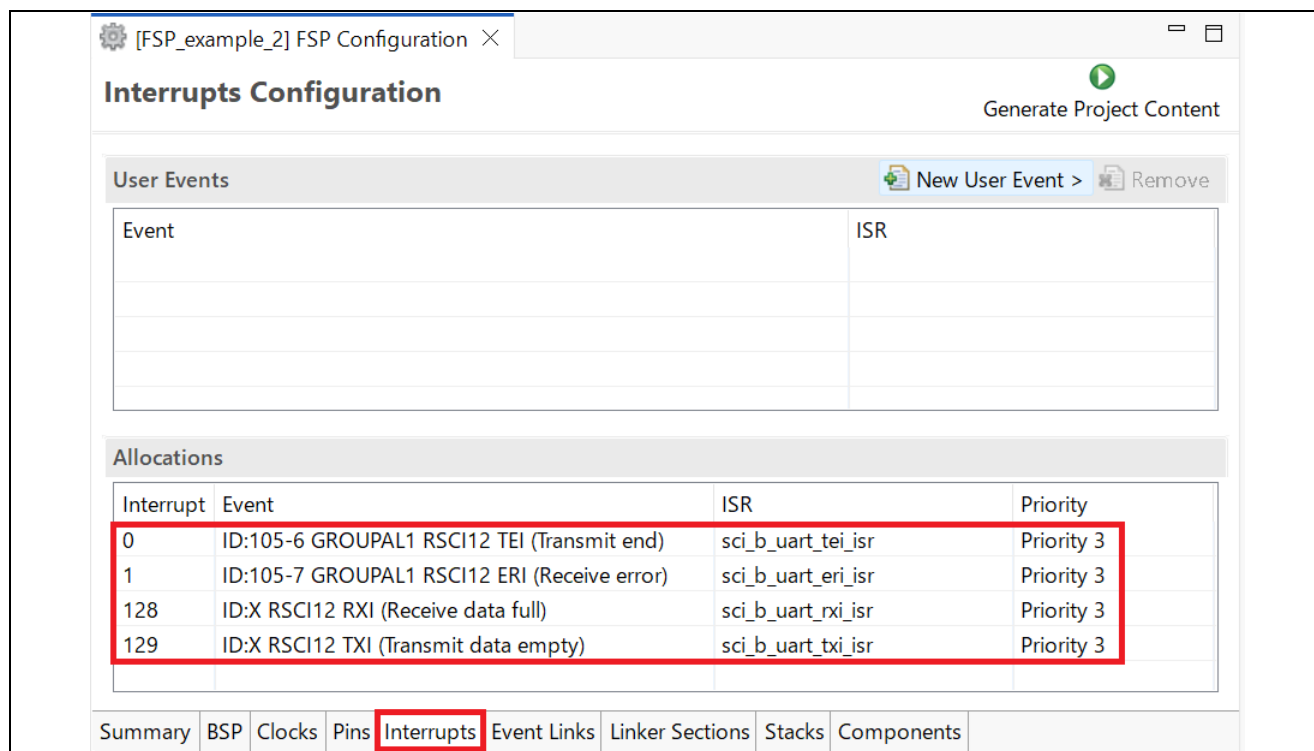


図 6-13 [Interrupts Configuration]ページ UART(SCI)割り込み登録

[Generate Project Content]を実行すると割り込み関数の定義が生成され、vector_data.h に割り込み関数定義の宣言が出力されます。上記の設定で割り込み関数が出力された vector_data.h は以下のようになります。

rx_gen¥vector_data.h

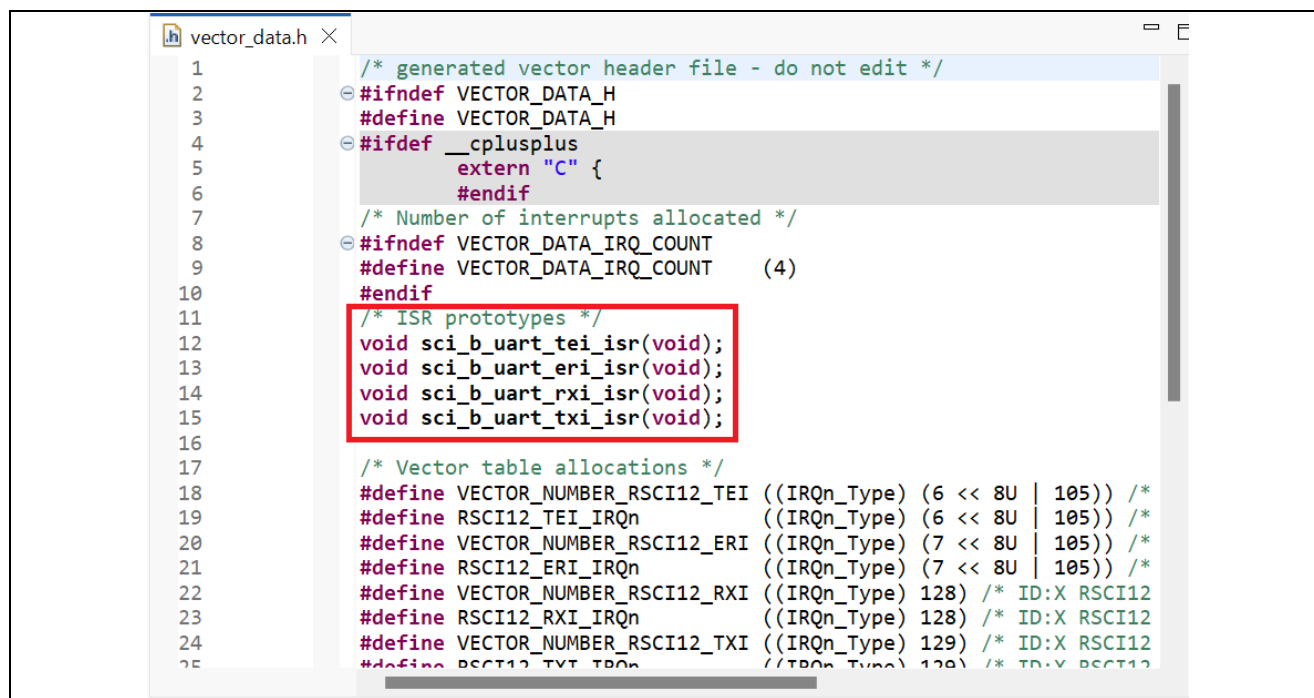


図 6-14 vector_data.h UART(SCI)割り込み関数宣言

コールバック関数の宣言は hal_data.h に出力されます。

rx_gen¥hal_data.h

```

1  /* generated HAL header file - do not edit */
2  #ifndef HAL_DATA_H_
3  #define HAL_DATA_H_
4  #include <stdint.h>
5  #include "bsp_api.h"
6  #include "common_data.h"
7  #include "r_sci_b_uart.h"
8  #include "r_uart_api.h"
9  FSP_HEADER
10 /* UART on SCI Instance. */
11 extern const uart_instance_t g_uart0;
12
13 /* Access the UART instance using these structures when calling API functions directly (::p_ap
14 extern sci_b_uart_instance_ctrl_t g_uart0_ctrl;
15 extern const uart_cfg_t g_uart0_cfg;
16 extern const sci_b_uart_extended_cfg_t g_uart0_cfg_extend;
17
18 #ifndef my_rsci_callback
19 void my_rsci_callback(uart_callback_args_t *p_args);
20 #endif
21 void hal_entry(void);
22 void g_hal_init(void);
23 FSP_FOOTER
24 #endif /* HAL_DATA_H */
25

```

図 6-15 hal_data.h UART(SCI)コールバック関数宣言

割り込み関数の実体はモジュール内に定義されていますが、

rx¥fsp¥src¥r_sci_b_uart¥r_sci_b_uart.c

```

1507
1508 #if (SCI_B_UART_CFG_TX_ENABLE)
1509
1511 * TXI interrupt processing for UART mode. TXI interrupt fires when the data in the data regi
1516 void sci_b_uart_txi_isr (void)
1595
1596 #endif
1597
1598 #if (SCI_B_UART_CFG_RX_ENABLE)
1599
1601 * RXI interrupt processing for UART mode. RXI interrupt happens when data arrives to the dat
1613 void sci_b_uart_rxi_isr (void)
1740
1741 #endif
1742
1743 #if (SCI_B_UART_CFG_TX_ENABLE)
1744
1746 * TEI interrupt processing for UART mode. The TEI interrupt fires after the last byte is tra
1750 void sci_b_uart_tei_isr (void)
1775
1776 #endif
1777
1778 #if (SCI_B_UART_CFG_RX_ENABLE)
1779
1781 * ERI interrupt processing for UART mode. When an ERI interrupt fires, the user callback fun
1784 void sci_b_uart_eri_isr (void)
1834
1835 #endif
1836

```

図 6-16 r_sci_b_uart.c 割り込み関数

コールバック関数は宣言のみが出力されるため、コールバック関数の実体はユーザ自身で作成する必要があります。

6.2.5 PORT 設定の追加

移行元プロジェクトでは SCI 関連の端子の設定のために初期設定関数を用いて行っていました。移行先プロジェクトでは SCI モジュールのプロパティから[Pin Configuration]ページで設定することによりスタートアップで呼び出される IOPORT モジュールの初期設定処理を用いて設定するように変更します。

g_uart0 のプロパティの Pins から RXD と TXD 端子の割り当てを行います。なおプロジェクト作成時にボードを選択した場合はボード情報によって周辺機能および端子が最初から割り当てられていることがあります。



図 6-17 UART(SCI)モジュール(r_sci_b_uart)のプロパティ Pins

端子の変更ボタンから[Pin Configuration]ページに切り替わります。



図 6-18 UART(SCI)モジュール(r_sci_b_uart)のプロパティ Pins 変更

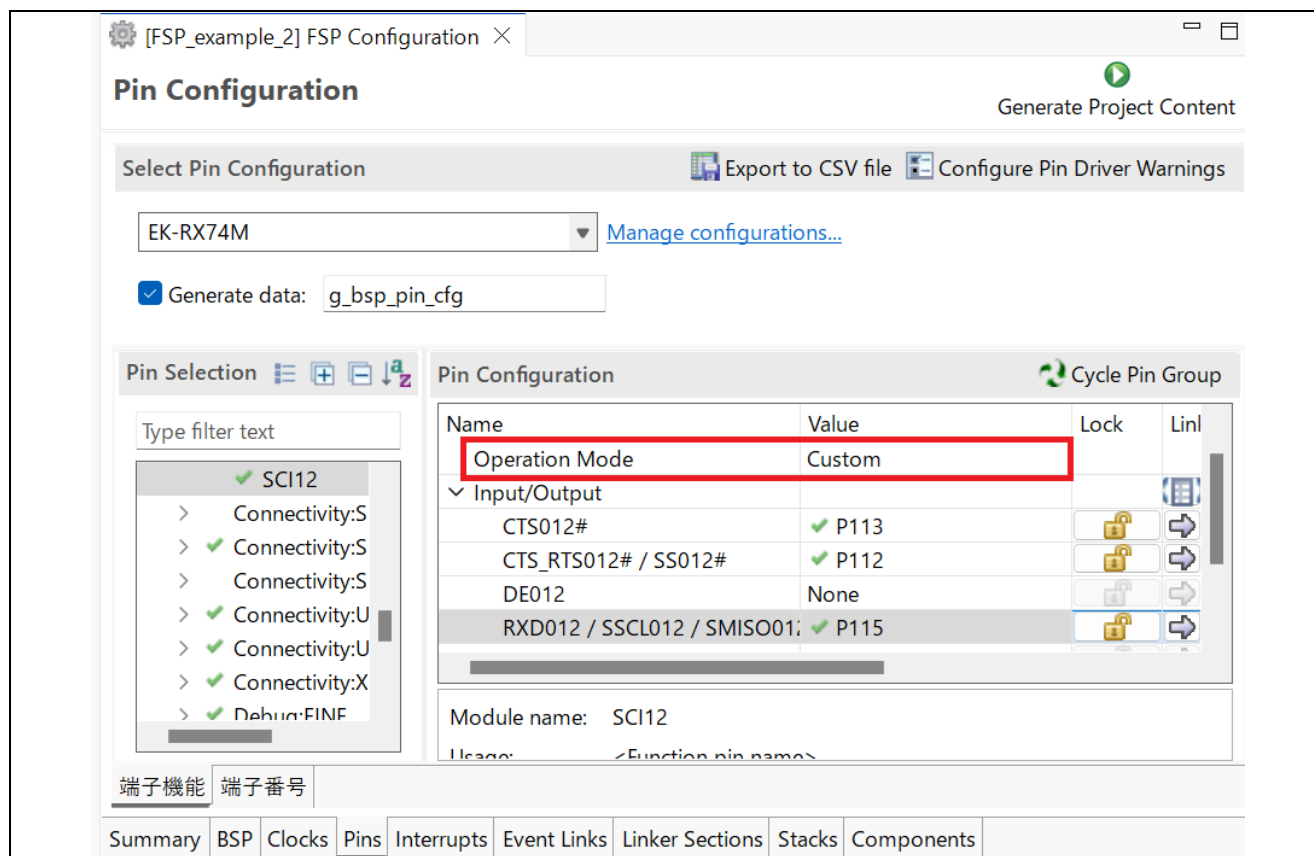


図 6-19 [Pin Configuration]ページ SCI12 Operation Mode 初期設定

EK-RX74M では Operation Mode は”Custom”に初期設定されており、RXD、TXD 端子もボードの仕様に合わせたポートが割り当てられています。Operation Mode を”Asynchronous UART”に変更します。

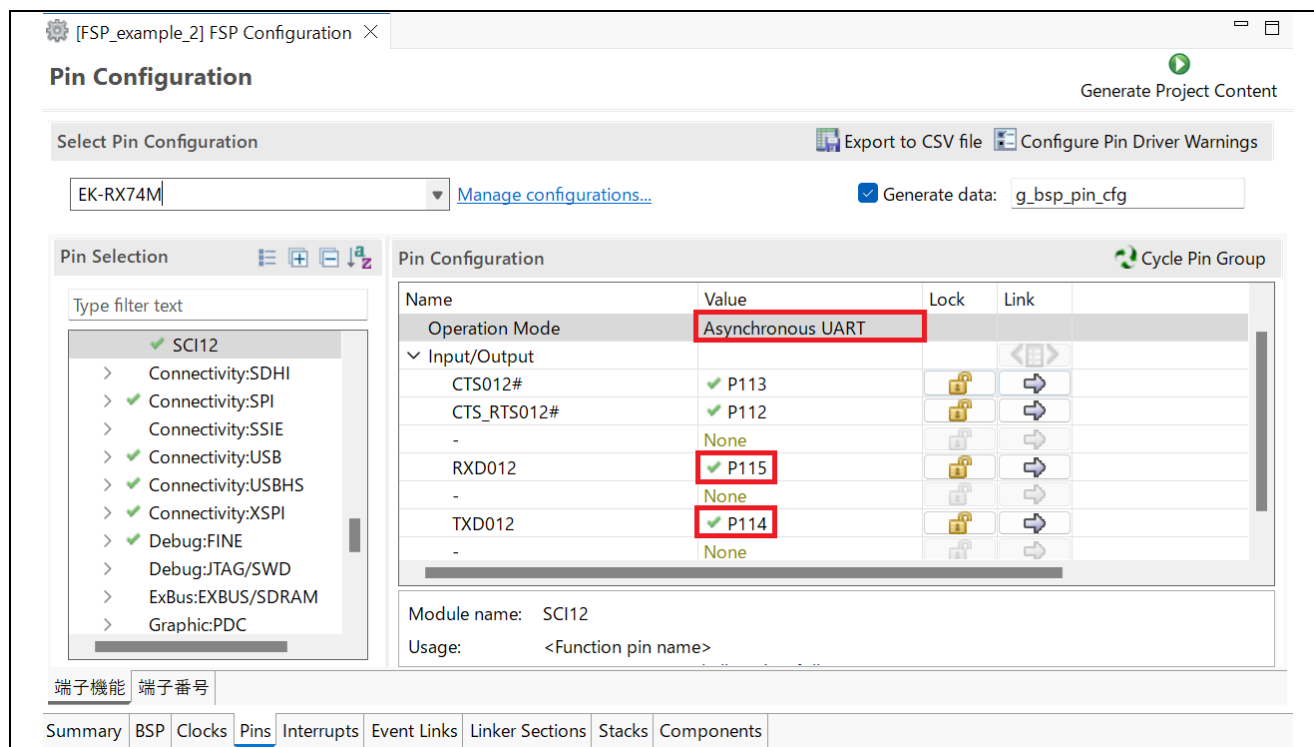


図 6-20 [Pin Configuration]ページ SCI12 Operation Mode UART

6.2.6 移行後のプロジェクトのフォルダ・ファイル構成と変更対象

移行後のプロジェクトのフォルダ・ファイル構成と変更対象を以下に示します。hal_entry.c および移植された rskrx671.h が変更対象です。iodefine.h は新規の Executable プロジェクトで生成されたファイルを追加します。

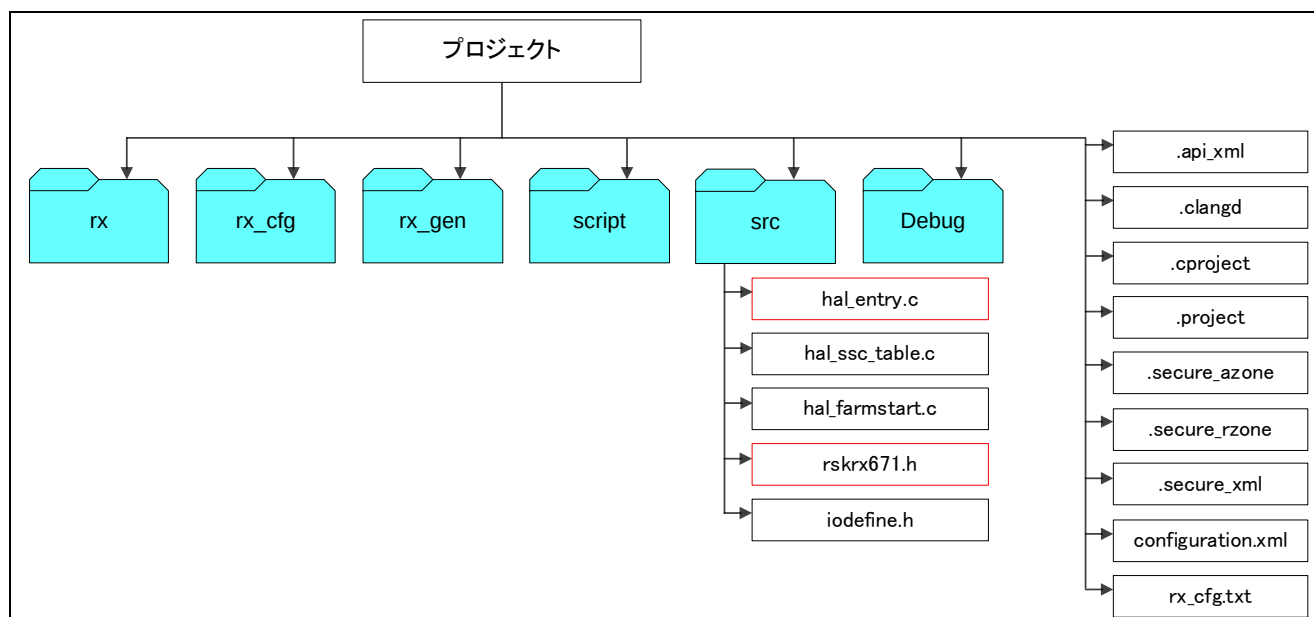


図 6-21 移行後プロジェクトのフォルダ・ファイル構成と変更対象

6.2.7 移行のポイント

5.7 ソースコードの移植に則り、main.c の main 関数本体の処理と SCI のコールバック関数を hal_entry.c に移植します。rskrx671.h を移植します。rskrx671.h が iodefine.h の定義を使用した定義を含むので 5.8 iodefine.h の追加に則り iodefine.h を追加します。SCI に関する端子の初期設定処理は IOPORT モジュールによって行われるため削除します。デバイスが変わるため、使用している定義を一部変更しています。また、FSP が提供するソースコードの定義と競合する定義の競合を回避するため、一部定義名を変更しています。5.9 FSP モジュールの処理への置き換えを参考に FIT モジュールの処理を FSP の処理へ置き換えます。

6.2.8 変更対象のファイルの置き換え後の処理

変更した最終的な hal_entry.c および rskrx671.h は以下のようになります。

hal_entry.c （赤字が最終的に追加した部分となります。）

```

#include "hal_data.h"
#include <stdio.h>          /* For sprintf */
#include "rskrx671.h"       /* Include user define */

uint8_t g_dest[32];
uint8_t g_src[32] = {"Ready"};
uint32_t g_transfer_complete = 0;
uint32_t g_receive_complete = 0;
  
```

```
static void rsci_rx671_init_ports(void);

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    LED0_PDR = 1;

    fsp_err_t err = FSP_SUCCESS;

    /* Called the R_SCI_B_UART_Open to open the transfer instance
       with initial configuration. */
    err = R_SCI_B_UART_Open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);

    /* Called the R_SCI_B_UART_Read to read data from UART. */
    err = R_SCI_B_UART_Read(&g_uart0_ctrl, g_dest, 1);
    assert(FSP_SUCCESS == err);

    /* Wait to get a character from the terminal so we know that it is ready. */
    while (!g_receive_complete)
    {
    }

    /* Called the R_SCI_B_UART_Write to write data to UART. */
    err = g_uart0.p_api->write(&g_uart0_ctrl, g_src, 12);
    assert(FSP_SUCCESS == err);

    while (!g_transfer_complete)
    {
    }

    /* Repeat forever. */
}
```

```
while(1)
{
    g_receive_complete = 0;
    g_transfer_complete = 0;

    /* Read next character from the serial port receive queue.
     * Repeat read if needed until data is available in the queue. */
    /* Called the R_SCI_B_UART_Read to read data from UART. */
    err = g_uart0.p_api->read(&g_uart0_ctrl, g_dest, 1);
    assert(FSP_SUCCESS == err);

    while (!g_receive_complete)
    {

    }

    /* Echo 1 character of the received data back to the sender. */
    printf("%c", (char)(g_dest[0]));
    g_src[0] = g_dest[0];
    /* Called the R_SCI_B_UART_Write to write data to UART. */
    err = g_uart0.p_api->write(&g_uart0_ctrl, g_src, 1);
    assert(FSP_SUCCESS == err);

    while (!g_transfer_complete)
    {

    }

}

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
 * Function Name: my_rsci_callback
 * Description  : This is a template for an RSCI Async Mode callback function.
 * Arguments    : pArgs -
 *                pointer to rsci_cb_args_t structure cast to a void. Structure
 *                contains event and associated data.
 * Return Value : none
 *****/
```

```
*****/  
void my_rsci_callback(uart_callback_args_t *p_args)  
{  
    /* Handle the UART event */  
    switch (p_args->event)  
    {  
        /* Received a character */  
        case UART_EVENT_RX_CHAR:  
        {  
            LED0 = ~LED0;        // Toggle LED to demonstrate callback execution.  
            break;  
        }  
  
        /* Receive complete */  
        case UART_EVENT_RX_COMPLETE:  
        {  
            g_receive_complete = 1;  
            break;  
        }  
  
        /* Transmit complete */  
        case UART_EVENT_TX_COMPLETE:  
        {  
            g_transfer_complete = 1;  
            break;  
        }  
  
        default:  
        {  
        }  
    }  
}
```

rskrx671.h (赤字が最終的に変更した部分となります。)

```
#include "iodefine.h"  
  
#ifndef RSKRX671_H  
#define RSKRX671_H  
  
/* Local defines */
```

```

#define LED_ON          (0)
#define LED_OFF         (1)
#define SET_BIT_HIGH    (1)
#define SET_BIT_LOW     (0)
#define SET_BYTE_HIGH   (0xFF)
#define SET_BYTE_LOW    (0x00)

/* Switches */
#define SW_ACTIVE        0
#define SW1_PIDR         PORT9.PIDR.BIT.B1
#define SW2              PORT9.PIDR.BIT.B2
#define SW3              PORT0.PIDR.BIT.B7
#define SW1_PDR          PORT9.PDR.BIT.B1
#define SW2_PDR          PORT9.PDR.BIT.B2
#define SW3_PDR          PORT0.PDR.BIT.B7
#define SW1_PMR          PORT9.PMR.BIT.B1
#define SW2_PMR          PORT9.PMR.BIT.B2
#define SW3_PMR          PORT0.PMR.BIT.B7

/* LEDs */
#define LED0             PORTF.PODR.BIT.PODR06
#define LED1_PODR        PORTF.PODR.BIT.B5
#define LED2_PODR        PORT0.PODR.BIT.B3
#define LED3_PODR        PORT0.PODR.BIT.B5
#define LED0_PDR         PORTF.PDR.BIT.PDR06
#define LED1_PDR         PORTF.PDR.BIT.B5
#define LED2_PDR         PORT0.PDR.BIT.B3
#define LED3_PDR         PORT0.PDR.BIT.B5

#endif /* RSKRX671_H */

```

移行後のプログラムでは UART として設定された RSCI12 を使用しターミナルエミュレーションを実行している PC のターミナルと通信します。ターミナルとの接続確認後に R_RSCI_Read 関数を実行しターミナルから 1 バイトのデータを受信するのを待機します。ターミナルからデータ受信すると FSP のモジュールには個別のバージョン番号はないので、特定の文字列を送信データとして R_RSCI_Write 関数を用いてターミナルに送信します。その後、R_RSCI_Read 関数による受信待ちと R_RSCI_Write 関数の送信を繰り返すことにより、ターミナルから受信したデータをターミナルに返すループバック動作を行う内容となっています。

6.2.9 本移行例における FIT モジュールの処理の FSP モジュールへの処理の置き換え

移行前の FIT モジュールの処理と移行し置き換え後の FSP モジュールの処理の比較を以下に示します。

6.2.9.1 メイン処理の比較

移行前 (FIT)	移行後 (FSP)
<pre> void main(void) { LED0_PDR = 1; rsci_cfg_t my_rsci_config; rsci_err_t my_rsci_err; uint32_t my_version; uint8_t version_str[12] = {0}; uint8_t my_char; /* Initialize the I/O port pins for communication on this RSCI channel. * This is specific to the MCU and ports chosen. For the RSKRX671 we will use the * RSCI channel connected to the USB serial port emulation. */ rsci_rx671_init_ports(); /* Set up the configuration data structure for asynchronous (UART) operation. */ my_rsci_config.async.baud_rate = 115200; my_rsci_config.async.clk_src = RSCI_CLK_INT; my_rsci_config.async.data_size = RSCI_DATA_8BIT; my_rsci_config.async.parity_type = RSCI_NONE_PARITY; my_rsci_config.async.msb_first = false; my_rsci_config.async.invert_data = false; my_rsci_config.async.stop_bits = RSCI_STOPBITS_1; my_rsci_config.async.int_priority = 3; /* 1=lowest, 15=highest */ /* OPEN ASYNC CHANNEL * Provide address of the config structure, * the callback function to be assigned, * and the location for the handle to be stored.*/ my_rsci_err = R_RSCI_Open(RSCI_CH10, RSCI_MODE_ASYNC, &my_rsci_config, my_rsci_callback, &g_my_rsci_handle); /* If there were an error this would demonstrate error detection of API calls. */ if (RSCI_SUCCESS != my_rsci_err) { R_BSP_NOP(); /* Your error handling code would go here. */ } } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ LED0_PDR = 1; fsp_err_t err = FSP_SUCCESS; /* Called the R_SCI_B_UART_Open to open the transfer instance * with initial configuration. */ err = g_uart0.p_api->open(&g_uart0_ctrl1, &g_uart0_cfg); assert(FSP_SUCCESS == err); } </pre>

図 6-22 メイン処理(1)

移行前 (FIT)	移行後 (FSP)
<pre> /* Get module software version using the API function provided. */ my_version = R_RSCI_GetVersion(); /* Demonstrate parsing and formatting of version number. */ sprintf((char *)version_str, "RSCIv%1.0hu.%2.2hu\r\n", ((my_version >> 16)&0xf), (my_version & 0xff)); printf("Initializes the RSCI successfully\r\n", version_str); /* Wait to get a character from the terminal so we know that it is ready. */ do { my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1); } while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err); /* Now send the version string out through the serial port to the terminal. */ R_RSCI_Send(g_my_rsci_handle, version_str, 12); /* Repeat forever. */ while(1) { /* Read next character from the serial port receive queue. * Repeat read if needed until data is available in the queue. */ do { my_rsci_err = R_RSCI_Receive(g_my_rsci_handle, &my_char, 1); } while (RSCI_ERR_INSUFFICIENT_DATA == my_rsci_err); /* Echo 1 character of the received data back to the sender. */ printf("%c", (char)(my_char)); R_RSCI_Send(g_my_rsci_handle, &my_char, 1); } /* End of function main. */ </pre>	<pre> /* Called the R_SCI_B_UART_Read to read data from UART. */ err = g_uart0.p_api->read(&g_uart0_ctrl1, g_dest, 1); assert(FSP_SUCCESS == err); /* Wait to get a character from the terminal so we know that it is ready. */ while (!g_receive_complete) { } /* Called the R_SCI_B_UART_Write to write data to UART. */ err = g_uart0.p_api->write(&g_uart0_ctrl1, g_src, 12); assert(FSP_SUCCESS == err); while (!g_transfer_complete) { } /* Repeat forever. */ while(1) { g_receive_complete = 0; g_transfer_complete = 0; /* Read next character from the serial port receive queue. * Repeat read if needed until data is available in the queue. */ /* Called the R_SCI_B_UART_Read to read data from UART. */ err = g_uart0.p_api->read(&g_uart0_ctrl1, g_dest, 1); assert(FSP_SUCCESS == err); while (!g_receive_complete) { } /* Echo 1 character of the received data back to the sender. */ printf("%c", (char)(g_dest[0])); g_src[0] = g_dest[0]; /* Called the R_SCI_B_UART_Write to write data to UART. */ err = g_uart0.p_api->write(&g_uart0_ctrl1, g_src, 1); assert(FSP_SUCCESS == err); while (!g_transfer_complete) { } } #if BSP_P2_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

図 6-23 メイン処理(2)

6.2.9.2 コールバック関数の比較

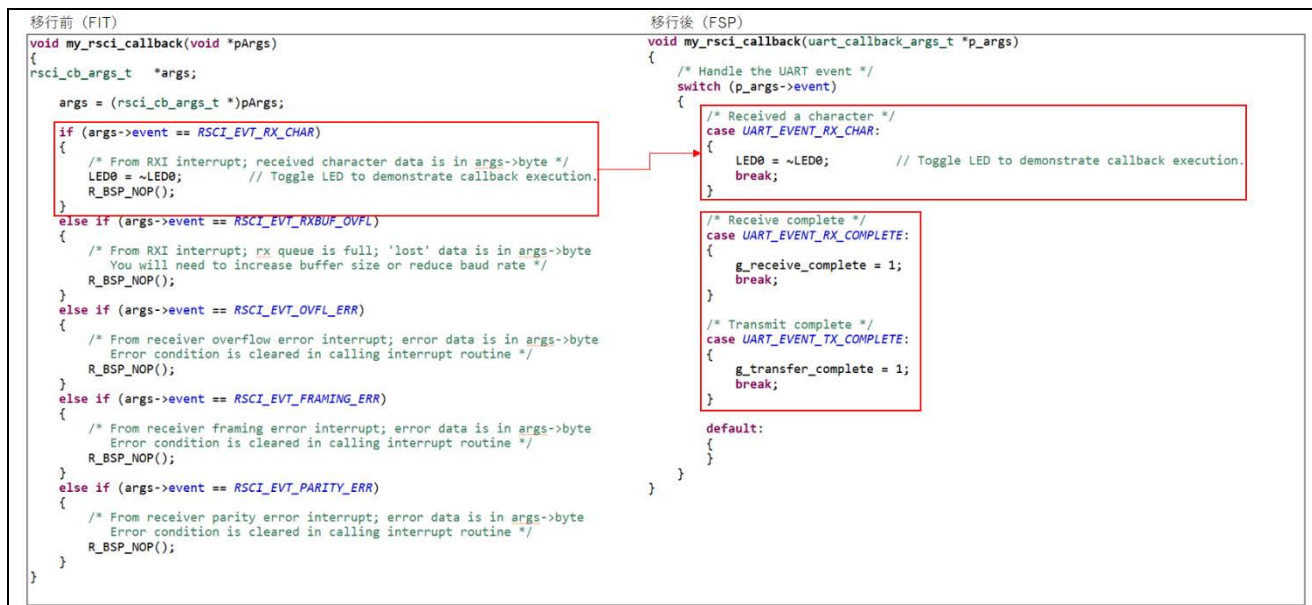


図 6-24 コールバック関数

7. コード生成アプリケーションの FSP プロジェクトへの移行例

コード生成のアプリケーションを例として FSP プロジェクトへ移行する方法について説明します。

7.1 CMT コード生成のサンプルプログラムを使用した移行例

コード生成の CMT の機能を使用したサンプルプログラムを FSP プロジェクトに移行する手順について説明します。参照するサンプルプログラムは [RX140 グループ Fast Prototyping Board for RX140 LED 点灯制御プログラムの](#) FPB_RX140_Blink プロジェクト（以下移行元プロジェクト）を例とします。

7.1.1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

移行元プロジェクトのフォルダ・ファイル構成と移行対象を以下に示します。

FPB_RX140_Blink_main.c、FPB_RX140_Blink_def.h、Config_CMT0_user.c、Config_ICU_user.c および Config_PORT_user.c が移行対象です。

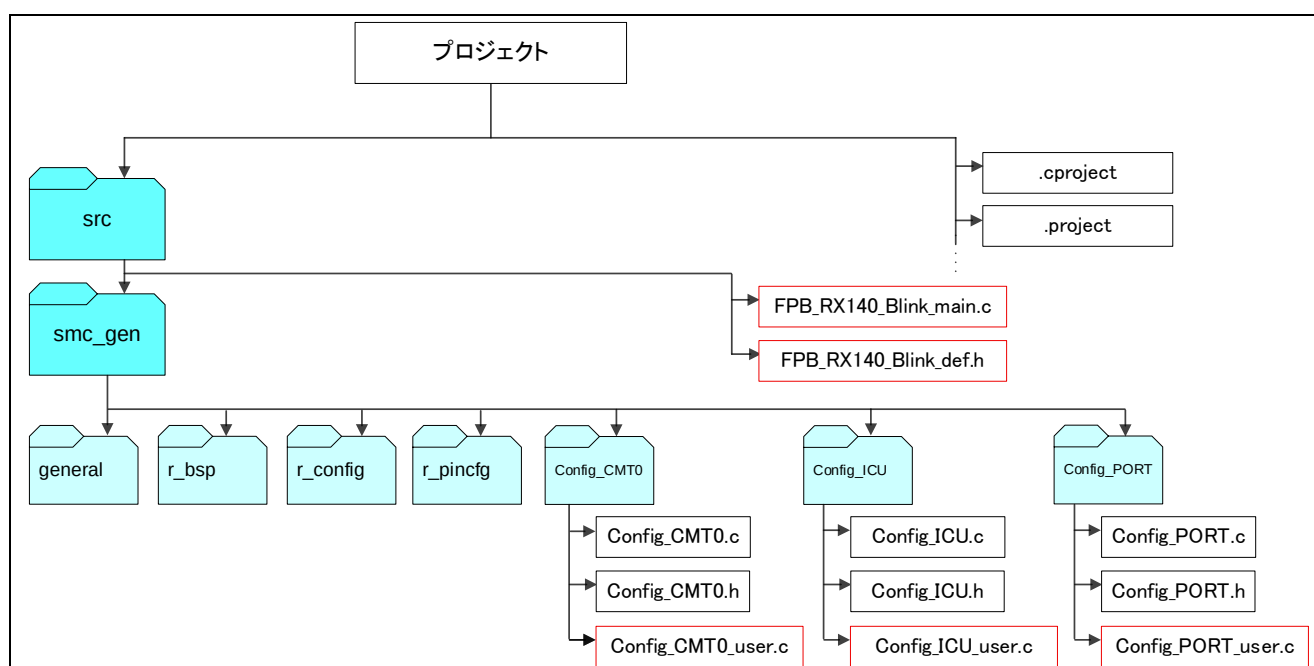


図 7-1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

7.1.2 移行元のプロジェクトの処理概要

移行元プロジェクトの FPB_RX140_Blink_main.c は以下のとおりです。

src\FPB_RX140_Blink_main.c

```

/*****
Includes  <System Includes> , "Project Includes"
*****/

#include "r_smc_entry.h"
#include "FPB_RX140_Blink_def.h"

void main(void);

void main(void)

```

```

{
    LED1 = LED_OFF;
    LED2 = LED_OFF;

    if (SW1_PUSH == SW1) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/
    {
        R_Config_ICU_IRQ0_Start();
    }
    else
    {
        R_Config_CMT0_Start();
    }
    while(1)
    {
        ; /* Blink LED0, LED1 Alternately by Interrupt */
    }
}

```

関連するファイルとして FPB_RX140_Blink_def.h がインクルードされています。
FPB_RX140_Blink_def.h の内容は以下のとおりです。

src¥FPB_RX140_Blink_def.h

```

/*****
Macro definitions
*****/
#ifndef FPB_RX140_BLINK_DEF_H_
#define FPB_RX140_BLINK_DEF_H_

/* General Values */
#define LED_ON      (1)
#define LED_OFF     (0)
#define SW1_PUSH    (0)
#define SW1_RELEASE (1)

/* Switches */
#define SW1          (PORT3.PIDR.BIT.B0)

/* LED port settings */
#define LED1          (PORT2.PODR.BIT.B0)
#define LED2          (PORT3.PODR.BIT.B2)
(中略)

```

```
#endif /* FPB_RX140_BLINK_DEF_H_ */
```

CMT の初期化は以下の関数でスタートアップ時に実行されます。

src¥smc_gen¥Config_CMT0¥Config_CMT0.c

```
void R_Config_CMT0_Create(void)
{
    /* Disable CMT0 interrupt */
    IEN(CMT0, CMT0) = 0U;

    /* Cancel CMT stop state in LPC */
    MSTP(CMT0) = 0U;

    /* Set control registers */
    CMT0.CMCR.WORD = _0003_CMT_CMCR_CLOCK_PCLK512 | _0040_CMT_CMCR_CMIE_ENABLE |
_0080_CMT_CMCR_DEFAULT;

    /* Set compare match register */
    CMT0.CMCOR = _7A11_CMT0_CMCOR_VALUE;

    /* Set CMT0 priority level */
    IPR(CMT0, CMT0) = _0F_CMT_PRIORITY_LEVEL15;

    R_Config_CMT0_Create_UserInit();
}
```

ICU の初期化は以下の関数でスタートアップ時に実行されます。

src¥smc_gen¥Config_ICU¥Config_ICU.c

```
void R_Config_ICU_Create(void)
{
    /* Disable IRQ interrupts */
    ICU.IER[0x08].BYTE = 0x00U;

    /* Disable software interrupt */
    IEN(ICU, SWINT) = 0U;

    /* Disable IRQ digital filter */
    ICU.IRQFLTE0.BYTE &= ~(_01_ICU_IRQ0_FILTER_ENABLE);

    /* Set IRQ digital filter sampling clock */
    ICU.IRQFLTC0.BIT.FCLKSEL0 = _0003_ICU_IRQ_FILTER_PCLK_64;
```

```
/* Set IRQ0 pin */
MPC.P30PFS.BYTE = 0x40U;
PORT3.PDR.BYTE &= 0xFEU;
PORT3.PMR.BYTE &= 0xFEU;

/* Set IRQ detection type */
ICU.IRQCR[0].BYTE = _04_ICU_IRQ_EDGE_FALLING;
IR(ICU, IRQ0) = 0U;

/* Enable IRQ digital filter */
ICU.IRQFLTE0.BYTE |= (_01_ICU_IRQ0_FILTER_ENABLE);

/* Set IRQ priority level */
IPR(ICU, IRQ0) = _0F_ICU_PRIORITY_LEVEL15;

R_Config_ICU_Create_UserInit();
}
```

PORT の初期化は以下の関数でスタートアップ時に実行されます。

src¥smc_gen¥Config_Port¥Config_Port.c

```
void R_Config_PORT_Create(void)
{
    /* Set PORT2 registers */
    PORT2.PODR.BYTE = _00_Pm0_OUTPUT_0;
    PORT2.ODR0.BYTE = _00_Pm0_CMOS_OUTPUT | _00_Pm1_CMOS_OUTPUT;
    PORT2.ODR1.BYTE = _00_Pm6_CMOS_OUTPUT | _00_Pm7_CMOS_OUTPUT;
    PORT2.PMR.BYTE = _00_Pm0_PIN_GPIO;
    PORT2.PDR.BYTE = _01_Pm0_MODE_OUTPUT | _3C_PDR2_DEFAULT;

    /* Set PORT3 registers */
    PORT3.PODR.BYTE = _00_Pm2_OUTPUT_0;
    PORT3.ODR0.BYTE = _00_Pm0_CMOS_OUTPUT | _00_Pm1_CMOS_OUTPUT |
    _00_Pm2_CMOS_OUTPUT;
    PORT3.ODR1.BYTE = _00_Pm4_CMOS_OUTPUT | _00_Pm6_CMOS_OUTPUT |
    _00_Pm7_CMOS_OUTPUT;
    PORT3.PMR.BYTE = _00_Pm2_PIN_GPIO;
    PORT3.PDR.BYTE = _04_Pm2_MODE_OUTPUT | _08_PDR3_DEFAULT;

    R_Config_PORT_Create_UserInit();
}
```

その他に関連する処理としてコード生成のユーザ処理で以下の処理が実装されています。

src¥smc_gen¥Config_CMT0¥Config_CMT0_user.c

```
#if FAST_INTERRUPT_VECTOR == VECT_CMT0_CMI0
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0),fint)
#else
#pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0))
#endif
static void r_Config_CMT0_cmi0_interrupt(void)
{
    /* Start user code for r_Config_CMT0_cmi0_interrupt.
       Do not edit comment generated here */
    LED1 = LED2;
    LED2 = ~LED2;
    /* End user code. Do not edit comment generated here */
}
```

src¥smc_gen¥Config_ICU¥Config_ICU_user.c

```
#if FAST_INTERRUPT_VECTOR == VECT_ICU_IRQ0
#pragma interrupt r_Config_ICU_irq0_interrupt(vect=VECT(ICU,IRQ0),fint)
#else
#pragma interrupt r_Config_ICU_irq0_interrupt(vect=VECT(ICU,IRQ0))
#endif
static void r_Config_ICU_irq0_interrupt(void)
{
    /* Start user code for r_Config_ICU_irq0_interrupt.
       Do not edit comment generated here */
    LED1 = LED2;
    LED2 = ~LED2;
    /* End user code. Do not edit comment generated here */
}
```

端子設定には他に特別な初期化処理はありません。

src¥smc_gen¥Config_Port¥Config_Port_user.c

```
void R_Config_PORT_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}
```

移行元のプログラムでは SW と LED を使用し、リセット後の SW 入力でサンプルプログラムの動作モードを切り替えます。リセット後、SW 入力がない場合はオートモードとして 2Hz 周期で LED を点滅させ、SW 入力があればマニュアルモードとして SW 入力毎に LED の点灯、消灯を切り替える仕様です。コード生成の機能として CMT、IRQ 割り込み、ポート制御の 3 つのコード生成機能を使用しています。

7.1.3 移行先のプロジェクトの作成

5 FSP 以前のプロジェクトから FSP プロジェクトに移行する手順で示した手順に則り、移行するプロジェクトを作成します。5.1 FSP プロジェクトの新規作成～5.4 モジュールの移行 (FIT→FSP) を参考に新規に作成した FSP プロジェクトをベースにします。

ポート制御は FSP では I/O Port モジュール(r_ioport_b)が既に組み込まれているため、追加するモジュールはありません。CMT と IRQ についてはそれぞれ FSP のモジュールを追加する必要があります。

7.1.4 CMT モジュールの追加

コード生成の CMT の設定に合わせたプロパティ設定で以下の方法でプロジェクトに CMT モジュールを 1 チャンネル分(CMT0)追加してください。

[New Stack] ボタンから[Timers] → [Timer, Compare Match Timer (r_cmt)]の選択を実行し、CMT0 用の g_timer0 をプロジェクトに追加します。

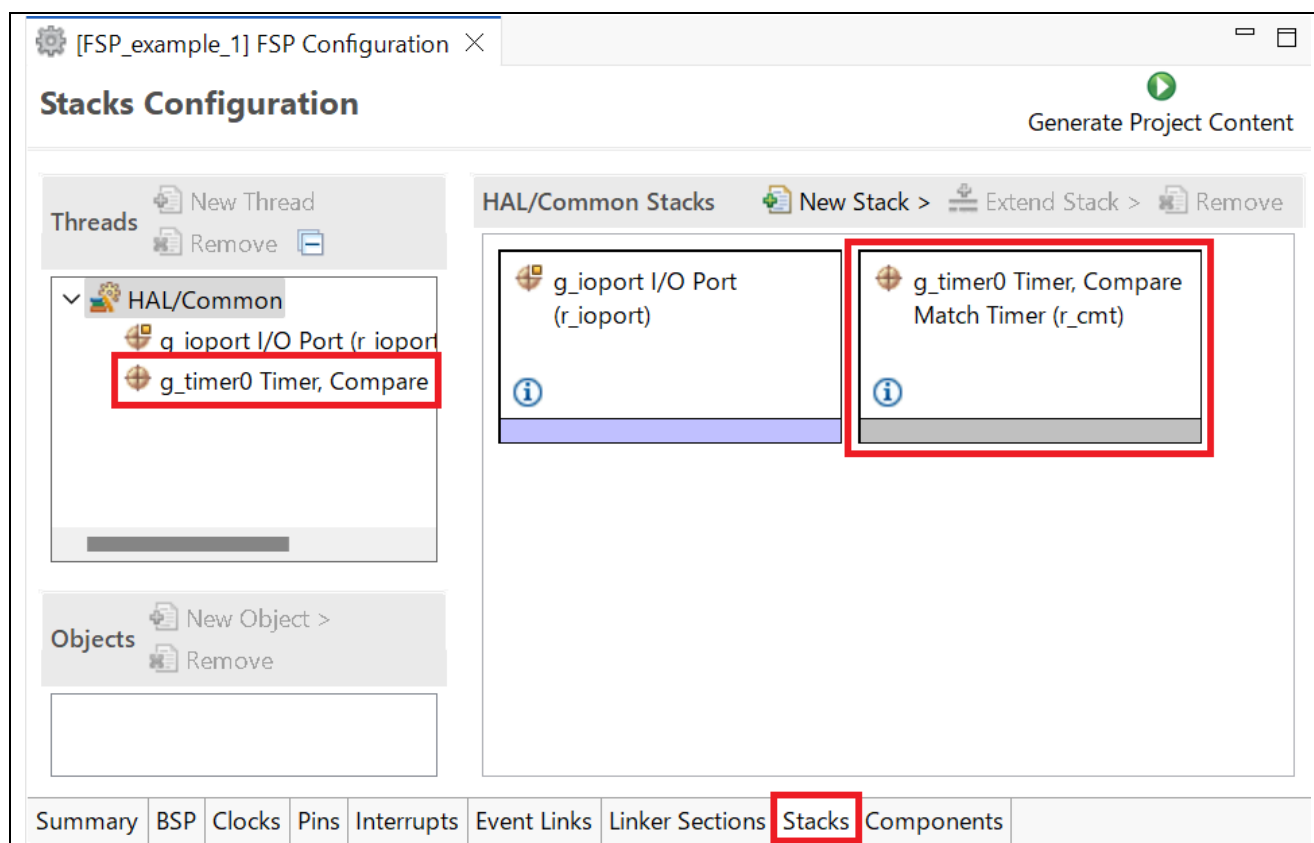


図 7-2 [Stacks Configuration]ページ CMT モジュール(r_cmt)の追加(CMT0)

コード生成の CMT の設定に合わせて CMT0 用の g_timer0 のプロパティを設定します。

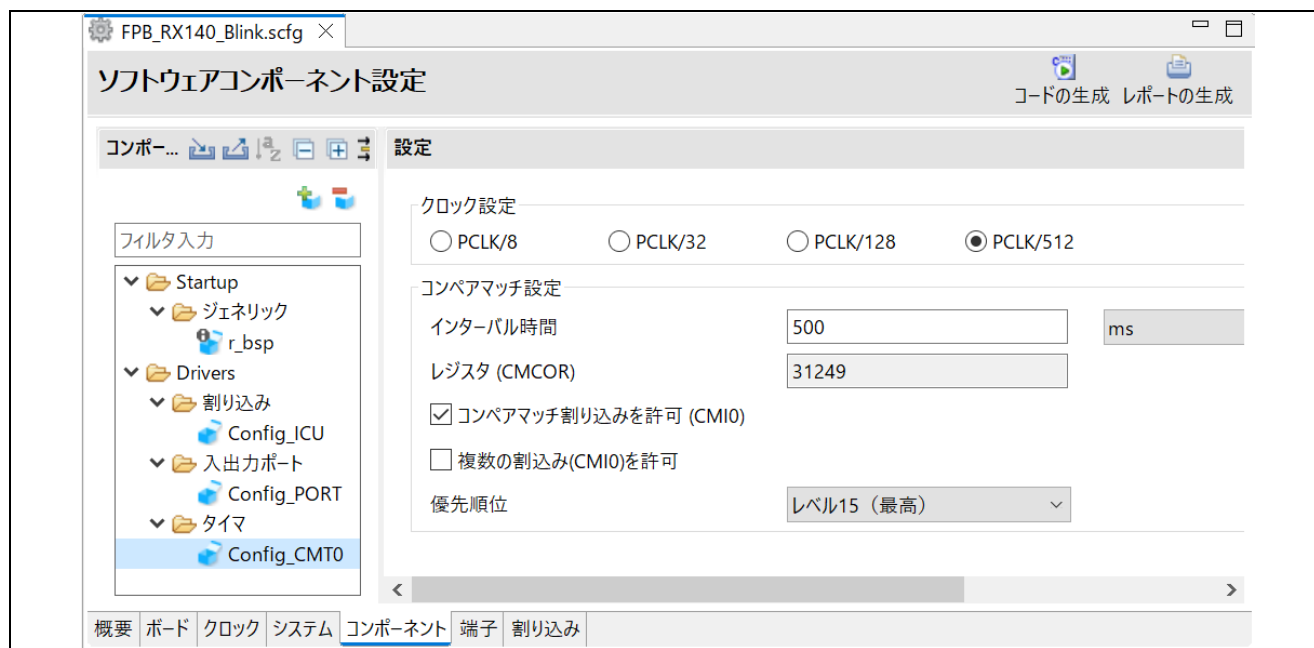


図 7-3 [コンポーネント] CMT コード生成の設定

g_timer0 は CMT0 用としてプロパティでは以下のように Period と Period Unit はサンプルプログラムに合わせるように 2Hz(Hertz)、コールバック関数名と割り込み優先レベルを設定します。

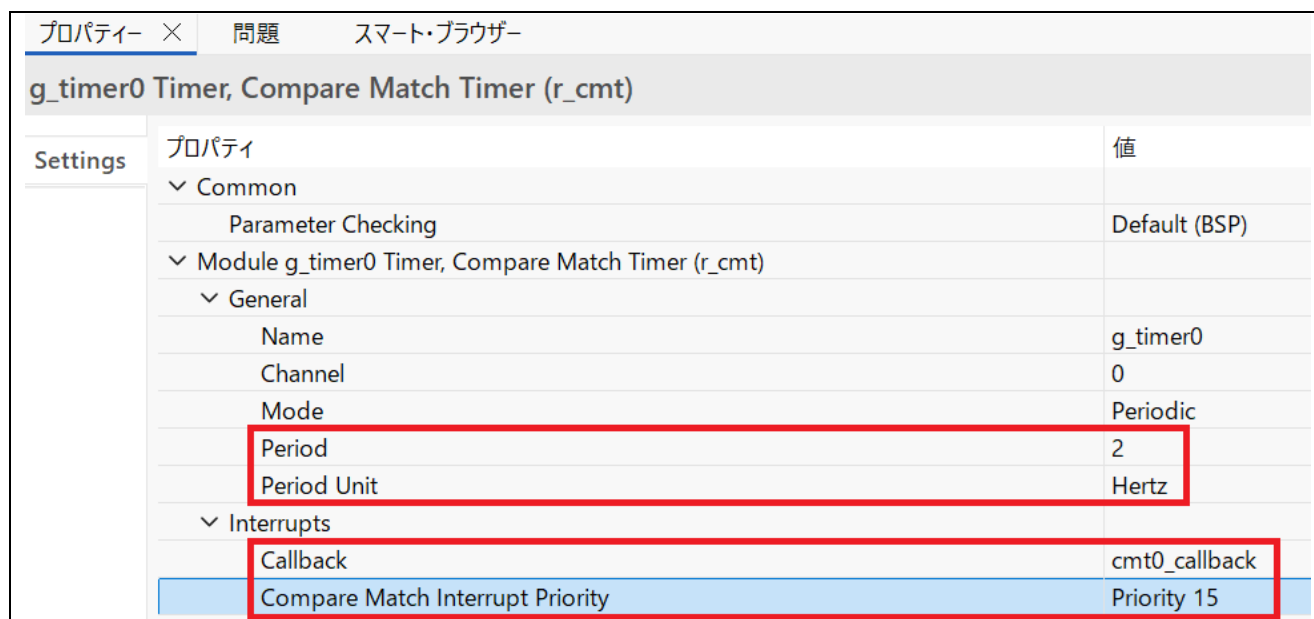


図 7-4 r_cmt g_timer0(CMT0) プロパティ

コールバック関数と割り込み優先レベルの設定により以下のように CMT0 の割り込みを使用する情報が登録されます。割り込み関数はモジュールに含まれる割り込み関数名となります。

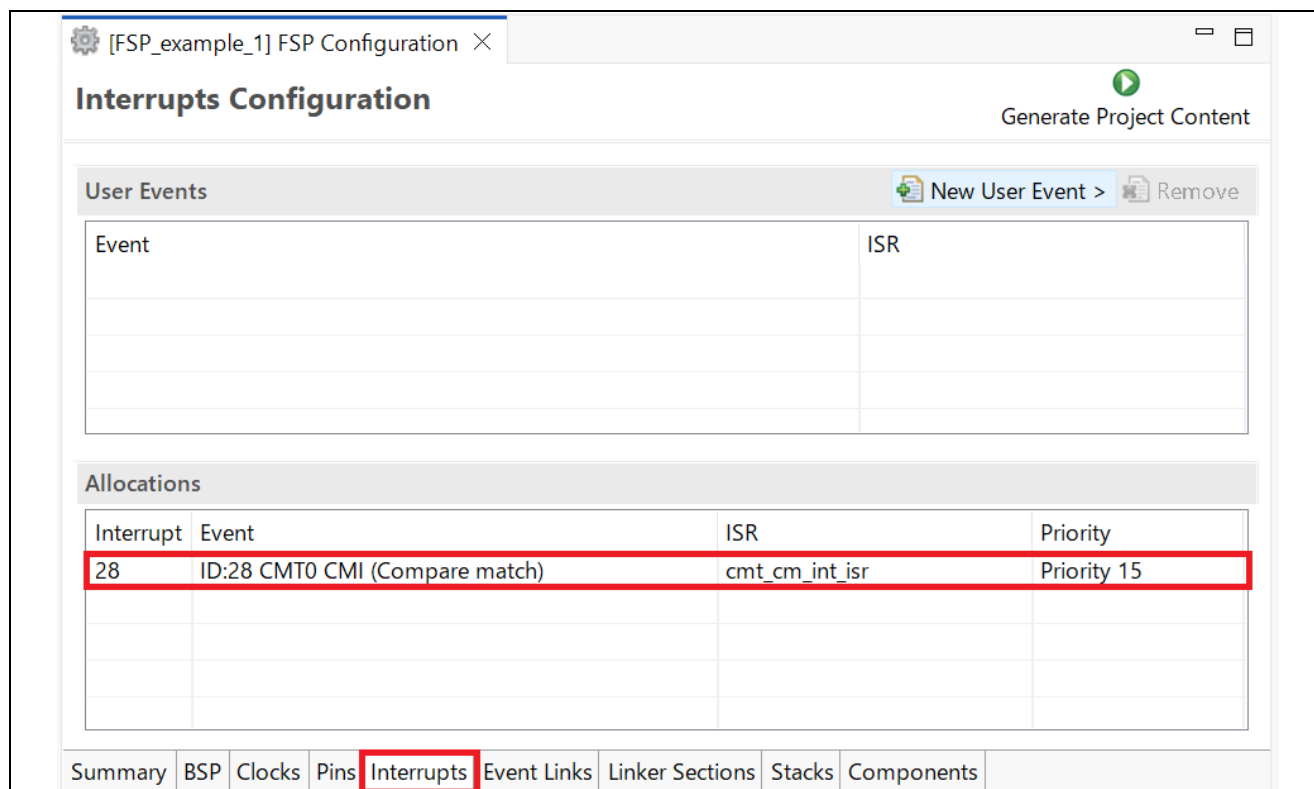


図 7-5 [Interrupts Configuration]ページ CMT0 割り込み登録

7.1.5 ICU モジュールの追加

続いて[New Stack] ボタンから[Input] → [External IRQ (r_icu)] を選択し実行します。

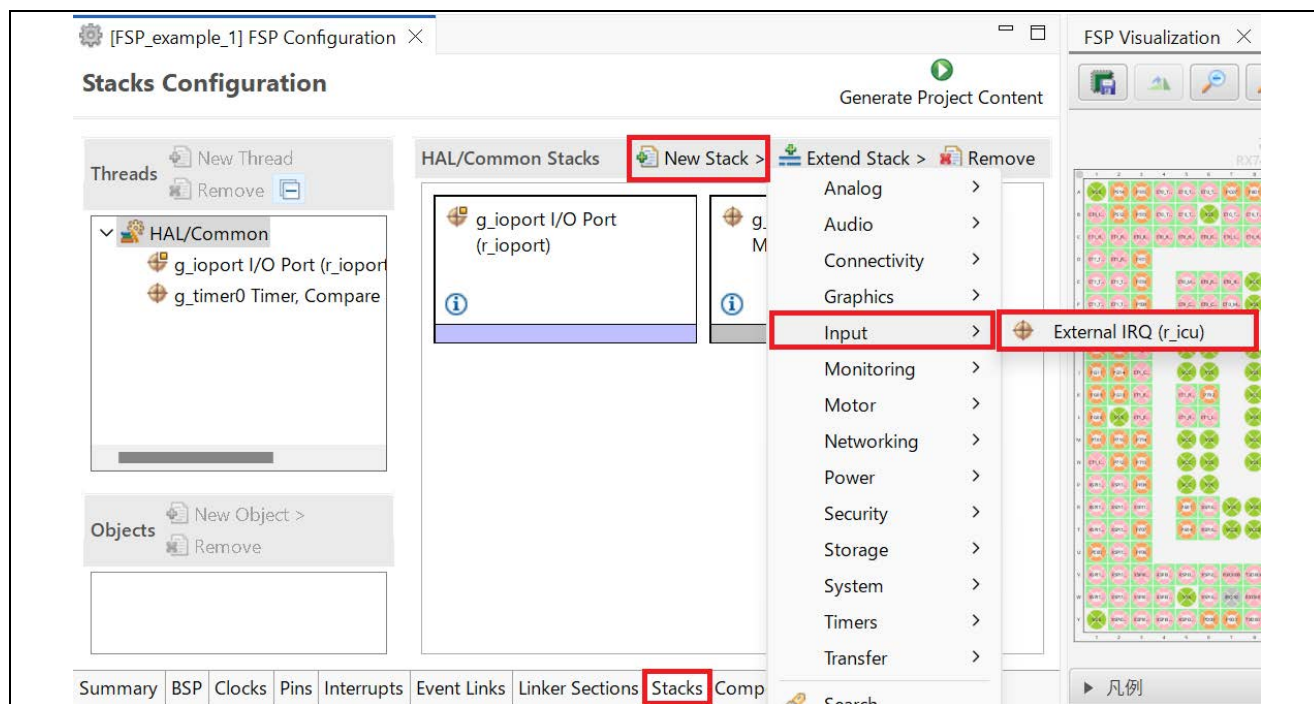


図 7-6 [Stacks Configuration]ページ External IRQ の選択

External IRQ 用の"g_external_irq0(r_icu)"が[Stacks Configuration]ページに追加されます。

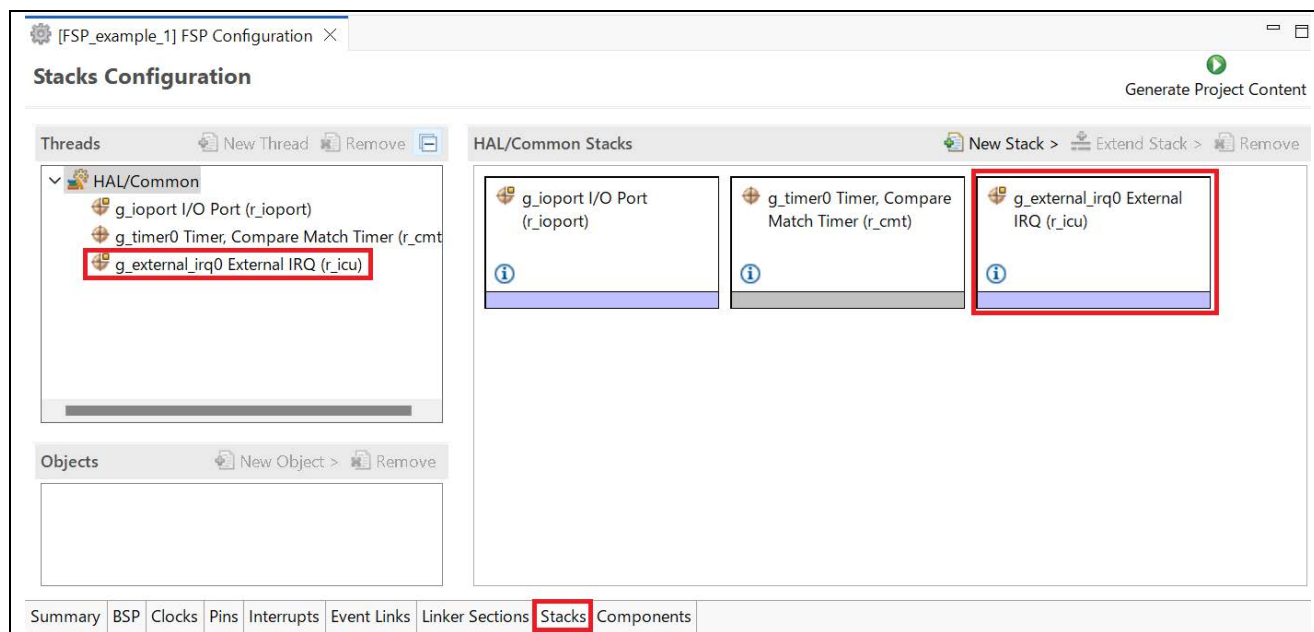


図 7-7 [Stacks Configuration]ページ External IRQ モジュール(r_icu)の追加

コード生成の IRQ の設定に合わせて IRQ0 用の g_external_irq0 のプロパティを設定します。



図 7-8 [コンポーネント] IRQ コード生成の設定

g_external_irq0 のプロパティは以下のように設定します。

表 7.1 移行時の設定項目と設定値

No	IRQ コンポーネントでの設定項目および設定値	g_external_irq0 のプロパティでの設定項目および設定値
1	IRQ0 設定 IRQ0 にチェック	Channel を 30 に設定。(EK-RX74M で SW が割り当てられた P414 を IRQ 端子として使用するため変更。)
2	IRQ0 設定 検出タイプを立ち下がりエッジに設定	Trigger を Falling に設定。
3	IRQ0 設定 デジタルフィルタを PCLK/64 に設定	Digital Filtering を Enabled に設定。 Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)を PCLK / 64 に設定。
4	IRQ0 設定 優先順位をレベル 15(最高)に設定	Pin Interrupt Priority を Priority 15 に設定。

上記の設定とコールバック関数名を設定した g_external_irq0 のプロパティは以下のとおりです。

プロパティ × 問題 スマート・ブラウザー		
g_external_irq0 External IRQ (r_icu)		
Settings	プロパティ	値
	▼ Common	
	Parameter Checking	Default (BSP)
	▼ Module g_external_irq0 External IRQ (r_icu)	
	Name	g_external_irq0
	Channel	30
	Trigger	Falling
	Digital Filtering	Enabled
	Filter Source	PCLK filter
	Filter Mode	3 consecutive levels match
	Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
	Callback	irq30_callback
	Pin Interrupt Priority	Priority 15

図 7-9 ICU(IRQ)モジュール(r_icu)のプロパティ

IRQ30 のコールバック関数と割り込み優先レベルの設定により以下のように IRQ の割り込みを使用する情報が登録されます。割り込み関数はモジュールに含まれる割り込み関数名となります。

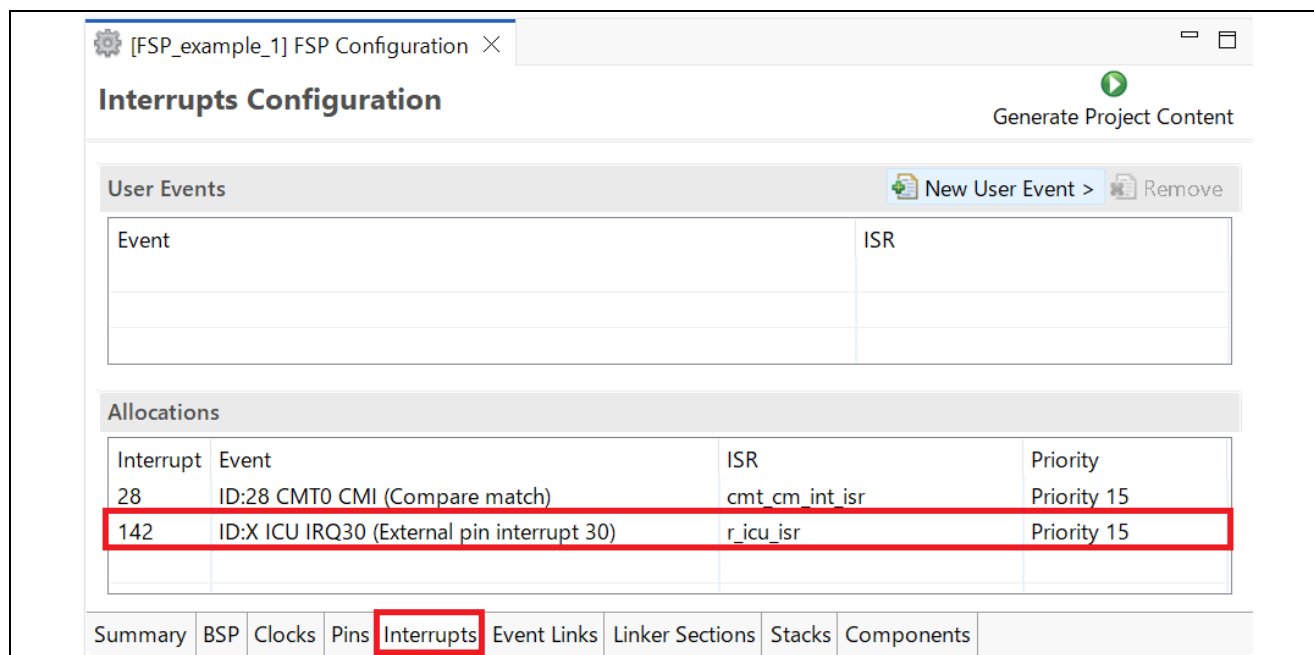


図 7-10 [Interrupts Configuration]ページ IRQ30 の割り込み登録

上記の設定で割り込み関数が出力された vector_data.h は以下のようになります。

rx_gen¥vector_data.h

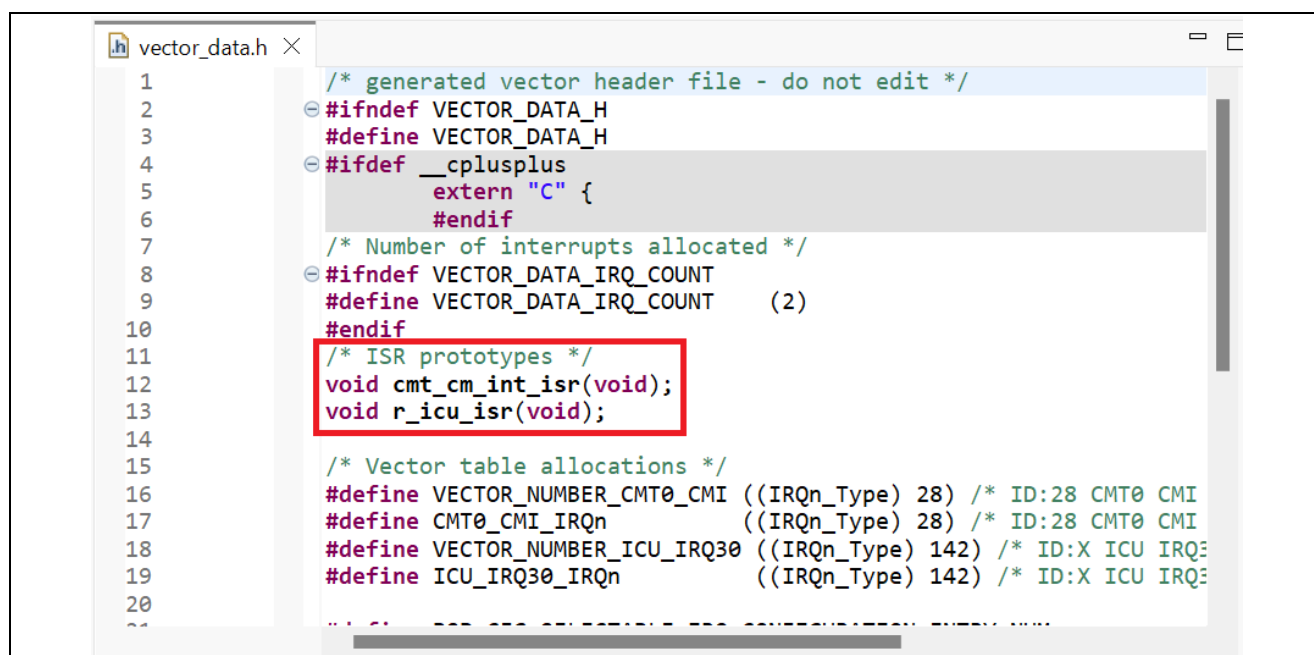


図 7-11 vector_data.h CMT と IRQ 割り込み関数宣言

CMT のコールバック関数の宣言は hal_data.h に出力されます。

rx_gen¥hal_data.h

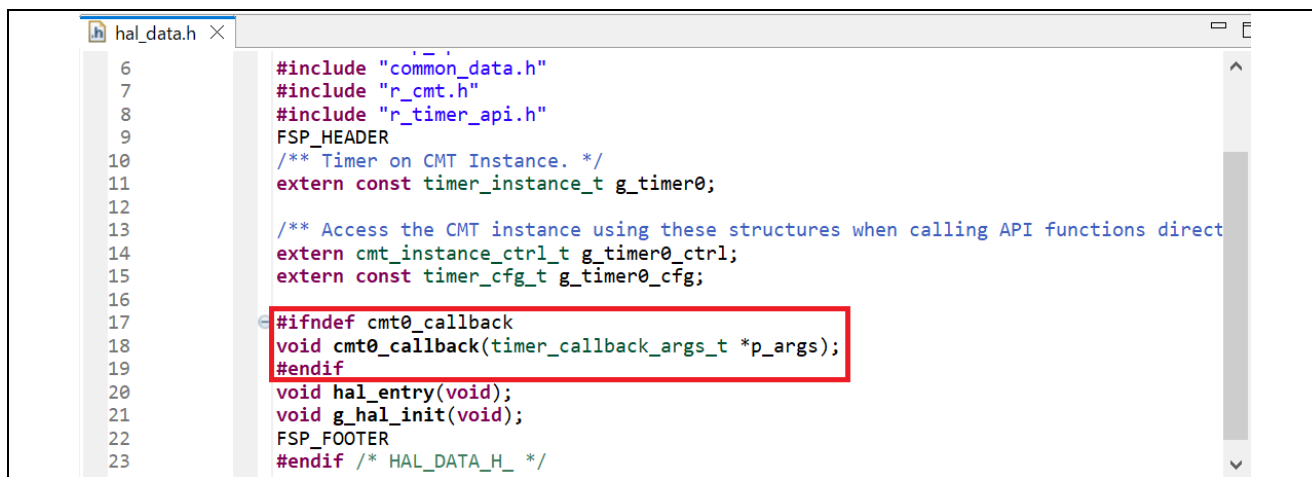


図 7-12 hal_data.h CMT コールバック関数宣言

IRQ30 のコールバック関数の宣言は common_data.h に出力されます。

rx_gen¥common_data.h

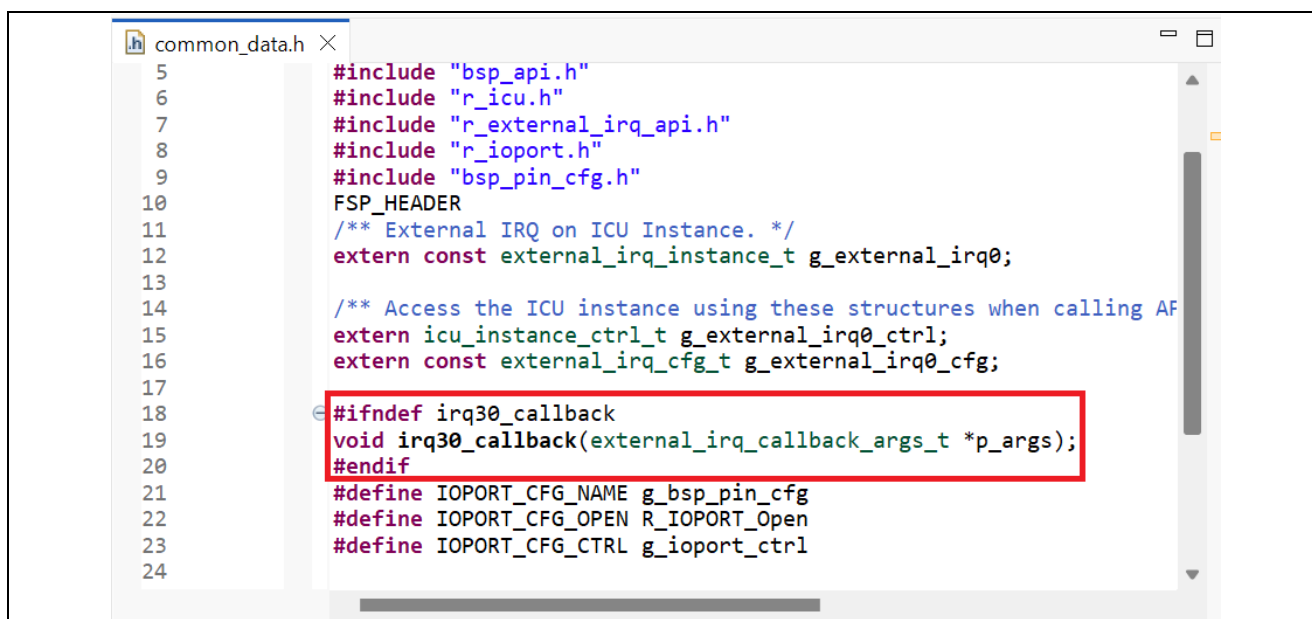


図 7-13 common_data.h IRQ コールバック関数宣言

割り込み関数の実体はモジュール内に定義されていますが、

rx¥fsp¥src¥r_cmt¥r_cmt.c

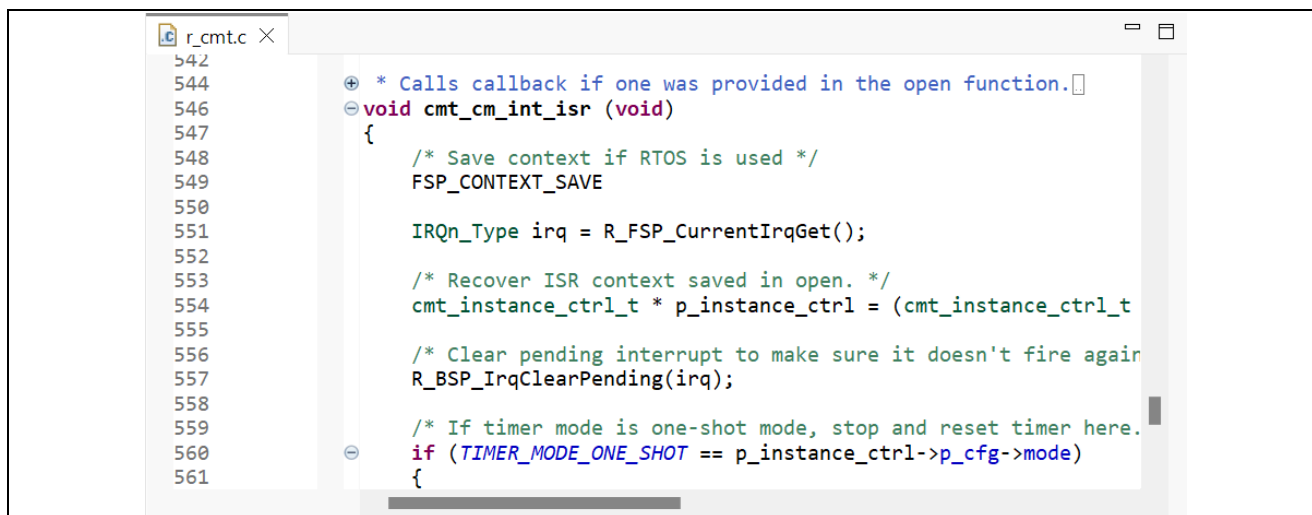


図 7-14 r_cmt.c 割り込み関数

rx¥fsp¥src¥r_icu¥r_icu.c

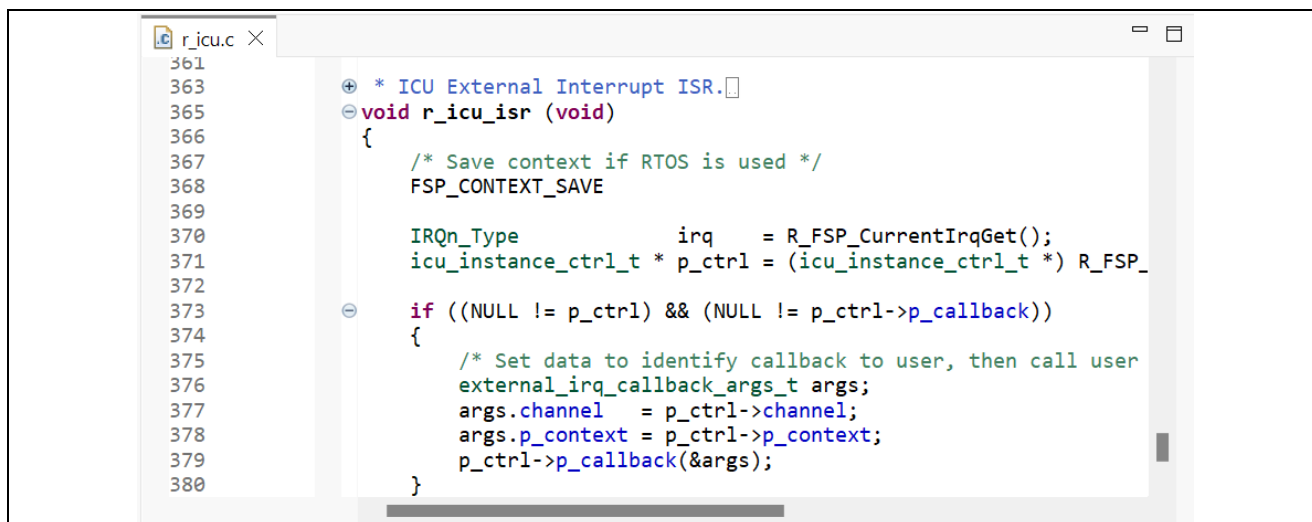


図 7-15 r_icu.c 割り込み関数

コールバック関数は宣言のみが出力されるため、コールバック関数の実体はユーザ自身で作成する必要があります。

7.1.6 PORT 設定の追加

移行元プロジェクトでは LED を使用する出力端子の設定のために Config_Port を使用していました。(SW で使用する端子は初期状態で使えるため、Config_Port による初期設定は未実施。) 移行先プロジェクトでは [Pin Configuration] ページで設定することによりスタートアップで呼び出される IOPORT モジュールの初期設定処理を用いて設定するように変更します。

Config_Port では P20 と P32 が設定されていましたが、



図 7-16 Config_Port Port2 設定

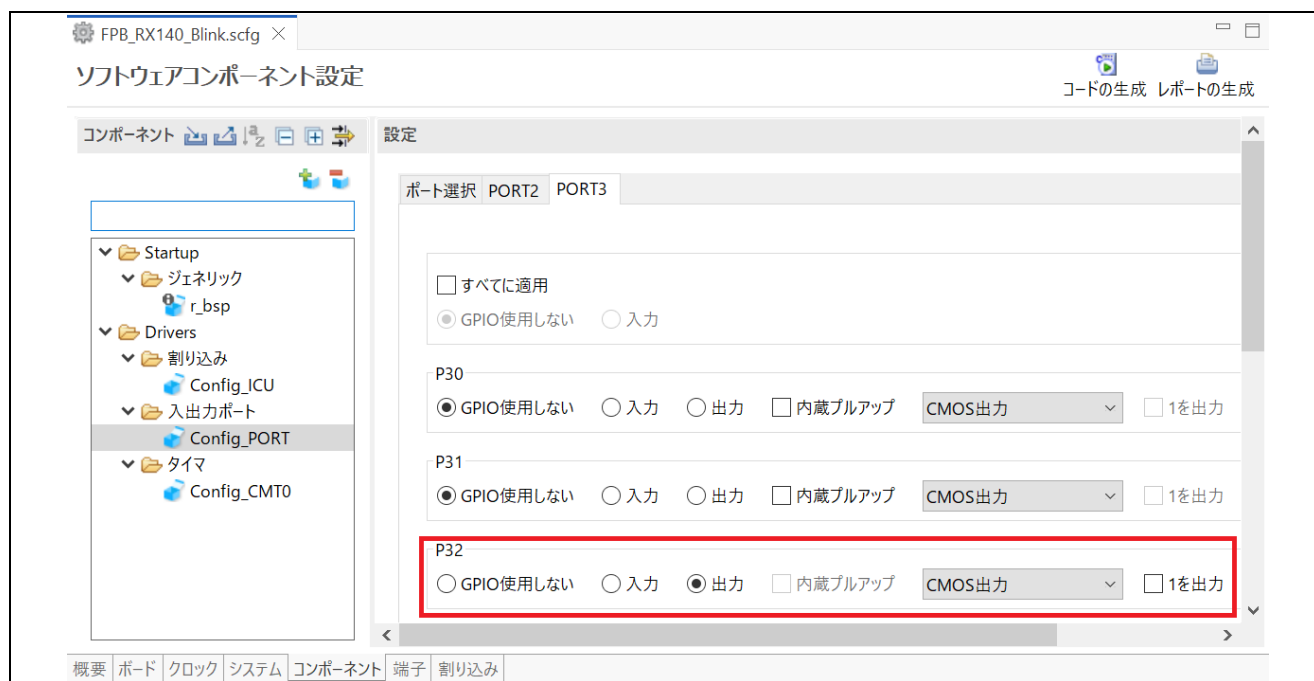


図 7-17 Config_Port Port3 設定

移行先プロジェクトではデバイスとボードが変わるため、LED 端子として PF05、P807 を使用するものとして[Pin Configuration]ページで設定します。EK-RX74M では PF05、P807 は LED 用の端子として初期設定されています。

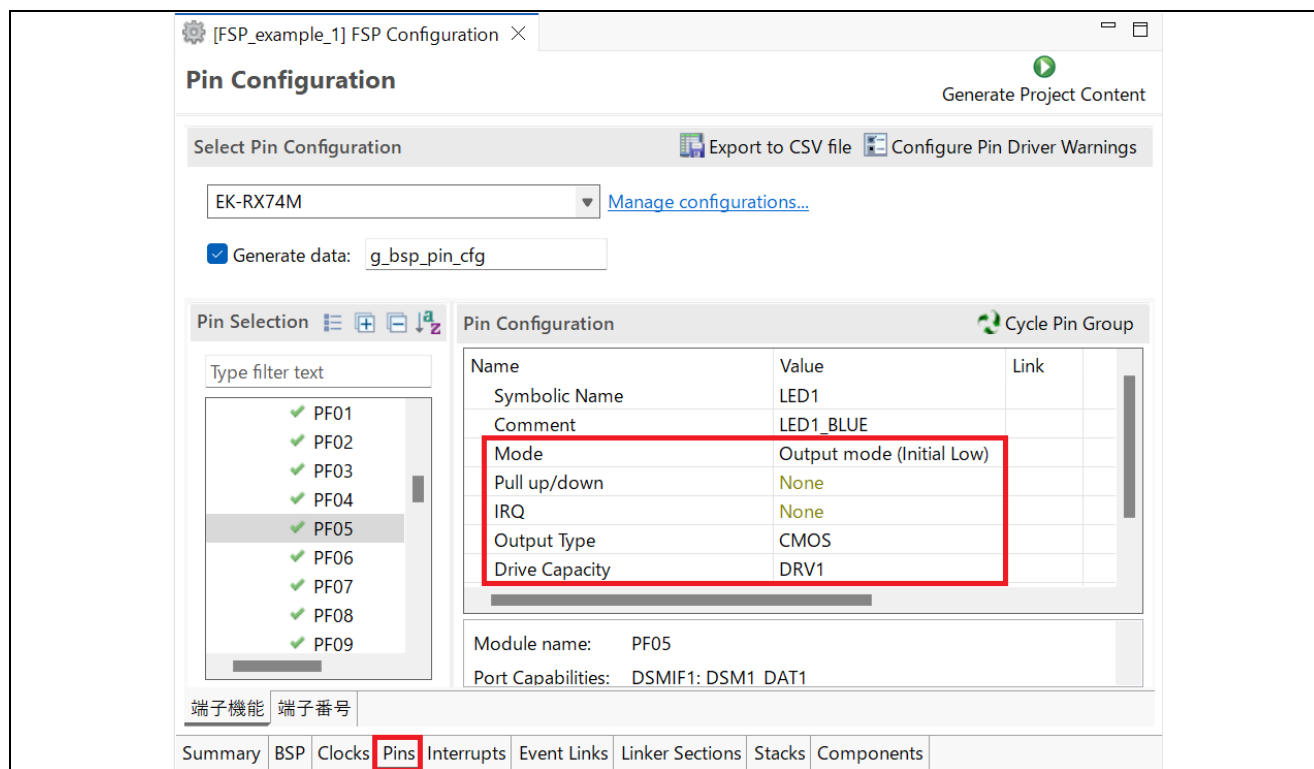


図 7-18 [Pin Configuration]ページ PF05 Mode Output mode

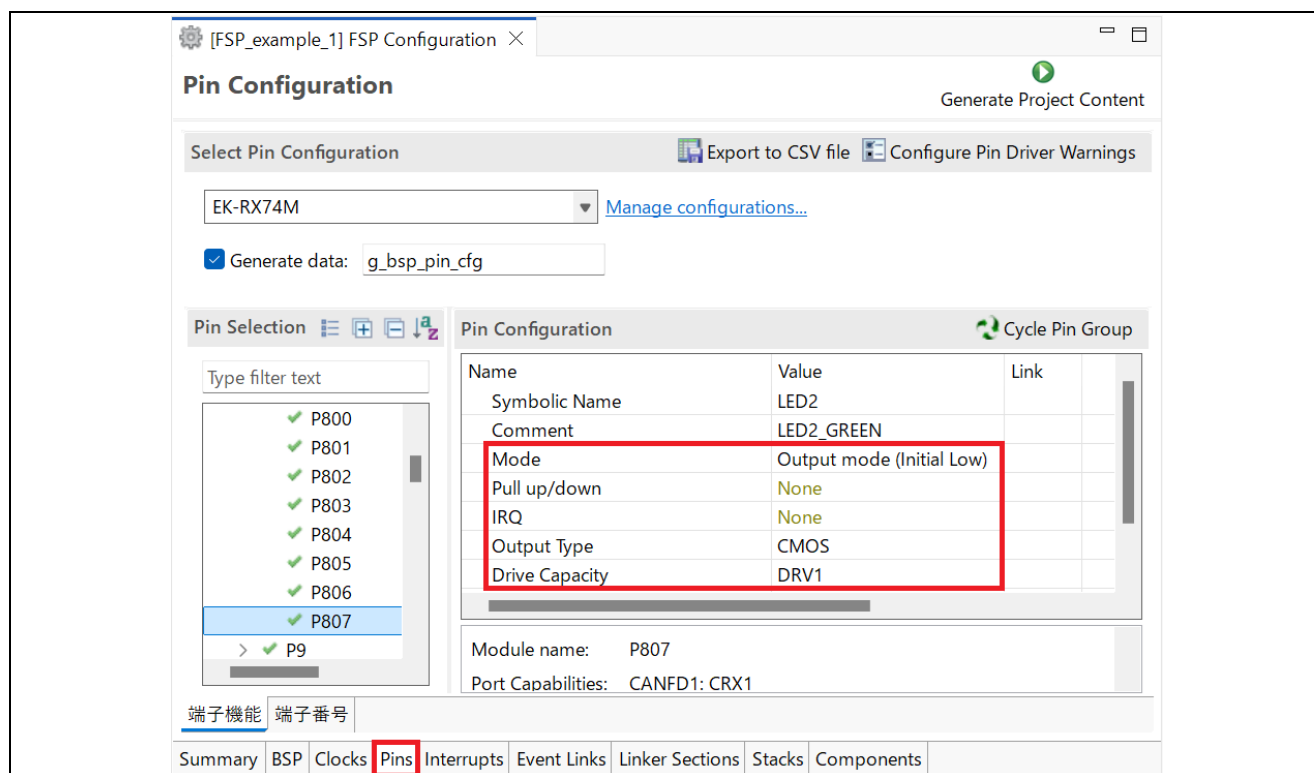


図 7-19 [Pin Configuration]ページ P807 Mode Output mode Output mode

また、SW 端子についても P414 を IRQ 端子として使用するため、g_external_irq で設定を行います。

EK-RX74M では P414 が IRQ30 用の端子として初期設定されています。

プロパティ × 問題 スマート・ブラウザー		
g_external_irq0 External IRQ (r_icu)		
Settings	プロパティ	値
	▼ Common	
	Parameter Checking	Default (BSP)
	▼ Module g_external_irq0 External IRQ (r_icu)	
	Name	g_external_irq0
	Channel	30
	Trigger	Falling
	Digital Filtering	Enabled
	Filter Source	PCLK filter
	Filter Mode	3 consecutive levels match
	Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
	Callback	irq30_callback
	Pin Interrupt Priority	Priority 15
	▼ Pins	
	IRQ30	P414
	IRQ30-DS	None

図 7-20 ICU モジュール(r_icu)のプロパティ Pins

ICU モジュールで使用する IRQ 端子を変更したい場合は端子の変更ボタンから[Pin Configuration]ページに切り替わるので

g_external_irq0 External IRQ (r_icu)		
Settings	プロパティ	値
	Trigger	Falling
	Digital Filtering	Enabled
	Filter Source	PCLK filter
	Filter Mode	3 consecutive levels match
	Digital Filtering Sample Clock (Only valid when Digital Filtering is Enabled)	PCLK / 64
	Callback	irq30_callback
	Pin Interrupt Priority	Priority 15
	▼ Pins	
	IRQ30	P414
	IRQ30-DS	None

図 7-21 ICU モジュール(r_icu)のプロパティ Pins 変更

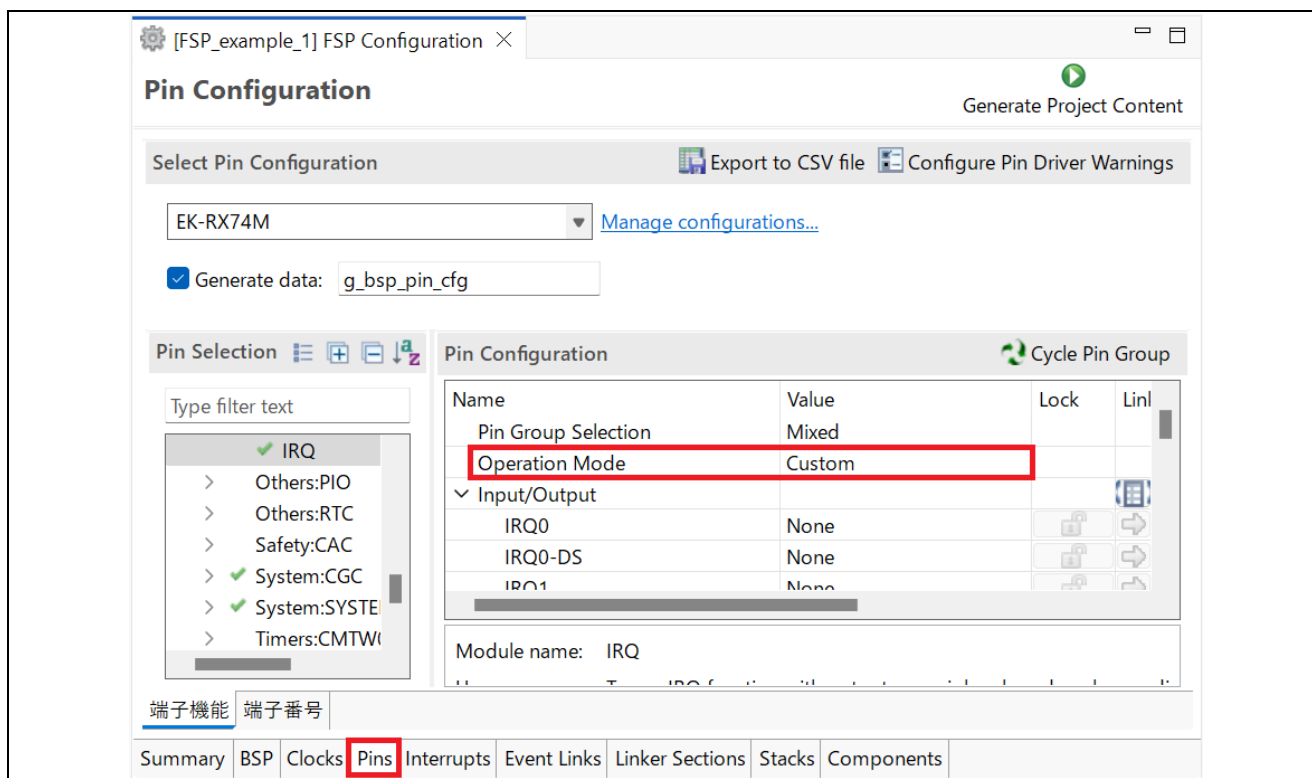


図 7-22 [Pin Configuration]ページ IRQ 有効

Operation Mode を"Custom"に変更し、割り当て可能な任意の端子を割り当ててください。

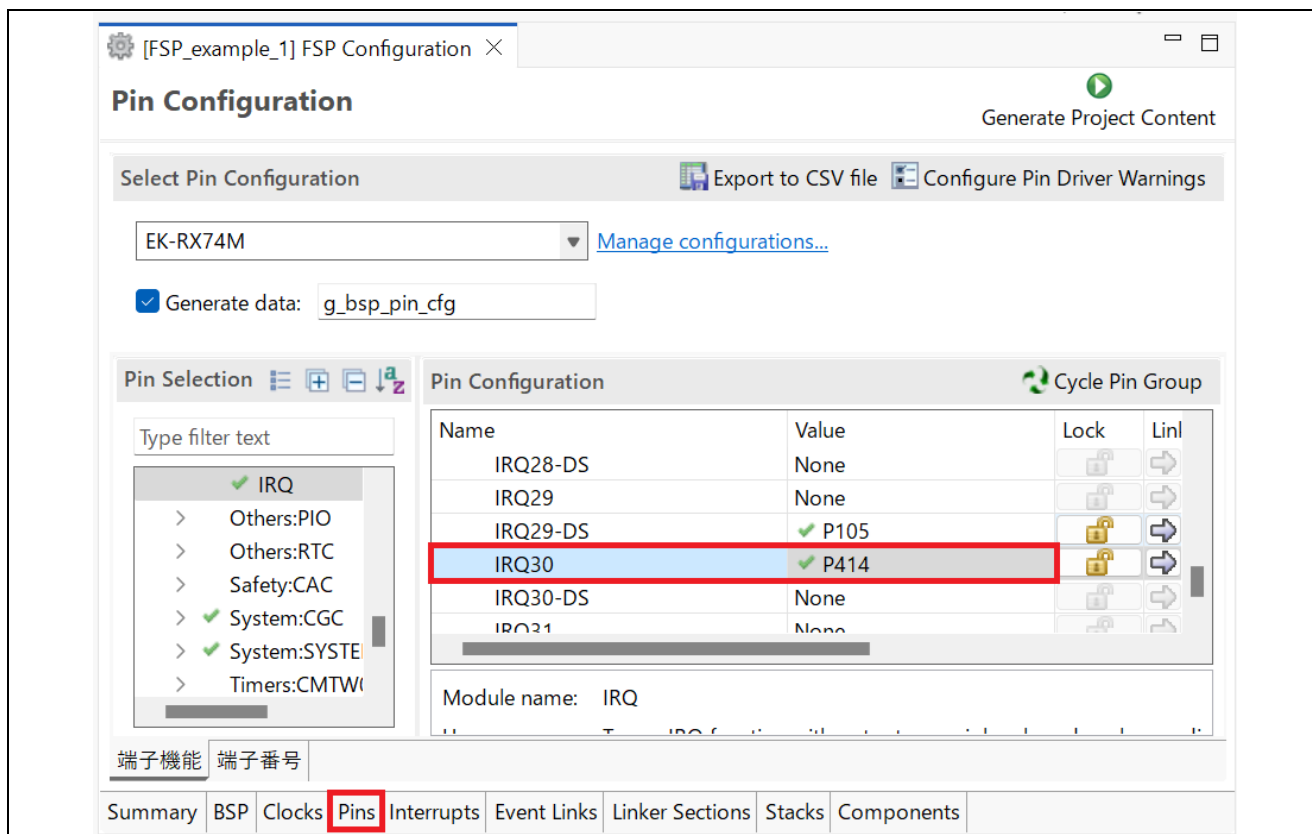


図 7-23 [Pin Configuration]ページ IRQ 有効 IRQ30 P414 割り当て

7.1.7 移行後のプロジェクトのフォルダ・ファイル構成と変更対象

移行後のプロジェクトのフォルダ・ファイル構成と変更対象を以下に示します。hal_entry.c および移植された rskrx671.h が変更対象です。iodefine.h は新規の Executable プロジェクトで生成されたファイルを追加します。

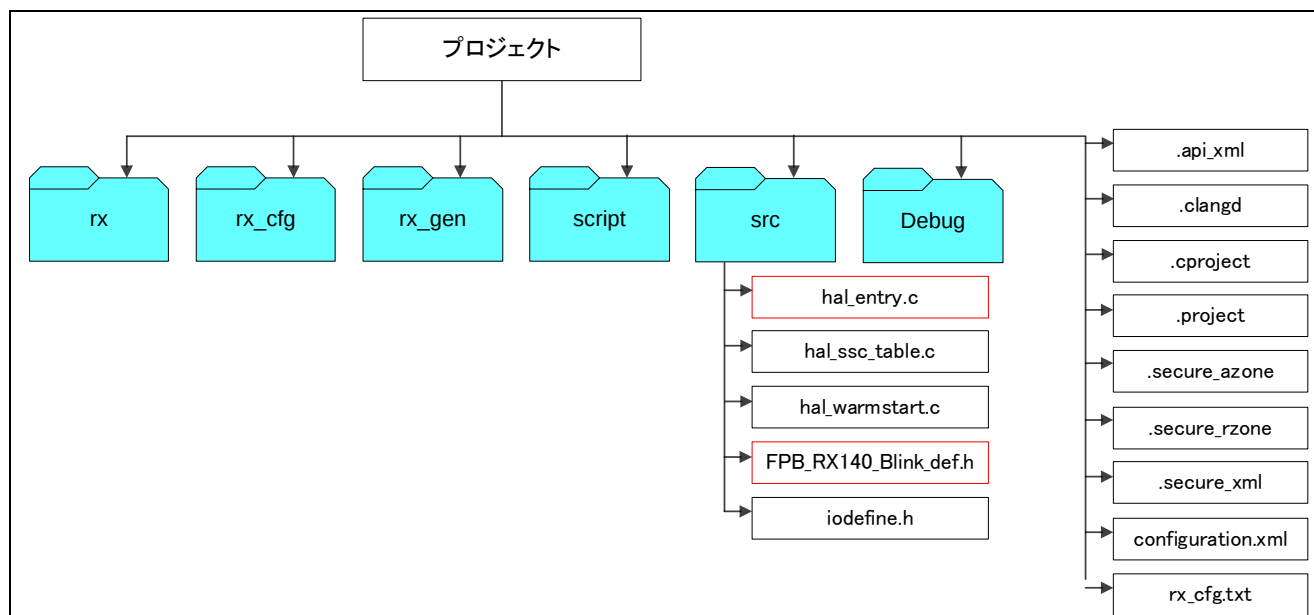


図 7-24 移行後プロジェクトのフォルダ・ファイル構成と変更対象

7.1.8 移行のポイント

5.7 ソースコードの移植に則り、main.c の main 関数本体の処理と CMT、ICU のユーザ処理をコールバック関数として hal_entry.c に移植します。スタートアップで行われる初期化処理はモジュールに置き換えるため、移植しません。FPB_RX140_Blink_def.h を移植します。

FPB_RX140_Blink_def.h が iodefine.h の定義を使用した定義を含むので 5.8 iodefine.h の追加に則り iodefine.h を追加します。デバイスが変わるため、使用している定義を一部変更しています。また、FSP が提供するソースコードの定義と競合する定義の競合を回避するため、一部定義名を変更しています。

5.9 FSP モジュールの処理への置き換えを参考にコード生成の処理を FSP の処理へ置き換えます。

7.1.9 変更対象のファイルの置き換え後の処理

変更した最終的な hal_entry.c および FPB_RX140_Blink_def.h は以下ようになります。

hal_entry.c （赤字が最終的に追加した部分となります。）

```
#include "hal_data.h"
#include "FPB_RX140_Blink_def.h"

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err = FSP_SUCCESS;

    LED1_PODR = LED_OFF;
    LED2_PODR = LED_OFF;

    if (SW1_PUSH == SW1_PIDR) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/
    {
        /* Configure the external interrupt. */
        fsp_err_t err = R_ICU_ExternalIrqOpen(&g_external_irq0_ctrl,
                                              &g_external_irq0_cfg);

        assert(FSP_SUCCESS == err);

        /* Enable the external interrupt. */
        /* Enable not required when used with ELC or DMAC. */
        err = R_ICU_ExternalIrqEnable(&g_external_irq0_ctrl);
        assert(FSP_SUCCESS == err);
    }
    else
    {
        /* Create a timer tick with callback at a rate of 2Hz. */
        /* Initializes the module for CMT0. */
        err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg);
    }
}
```

```

    /* Handle any errors. This function should be defined by the user. */
    assert(FSP_SUCCESS == err);

    /* Start the timer. */
    (void) R_CMT_Start(&g_timer0_ctrl);
}
while(1)
{
    ; /* Blink LED1, LED2 Alternately by Interrupt */
}

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

void cmt0_callback(timer_callback_args_t *p_args)
{
    if (TIMER_EVENT_CYCLE_END == p_args->event)
    {
        /* Add application code to be called periodically here. */
        LED1_PODR = LED2_PODR;
        LED2_PODR = ~LED2_PODR;
    }
}

void irq30_callback(external_irq_callback_args_t *p_args)
{
    (void) p_args;

    LED1_PODR = LED2_PODR;
    LED2_PODR = ~LED2_PODR;
}

```

FPB_RX140_Blink_def.h (赤字が最終的に変更した部分となります。)

```
#include "iodefine.h"
```

```
/******
```

Macro definitions

```

*****/

#ifndef FPB_RX140_BLINK_DEF_H_
#define FPB_RX140_BLINK_DEF_H_

/* General Values */
#define LED_ON          (1)
#define LED_OFF         (0)
#define SW1_PUSH        (0)
#define SW1_RELEASE     (1)

/* Switches */
#define SW1_PIDR        (PORT5.PIDR.BIT.PIDR00)

/* LED port settings */
#define LED1_PODR        (PORTF.PODR.BIT.PODR06)
#define LED2_PODR        (PORT8.PODR.BIT.PODR07)
(中略)
#endif /* FPB_RX140_BLINK_DEF_H_ */

```

7.1.10 本移行例におけるコード生成の処理の FSP モジュールへの処理の置き換え

移行前のコード生成の処理と移行し置き換え後の FSP モジュールの処理の比較を以下に示します。

7.1.10.1 モジュールの初期化処理の比較

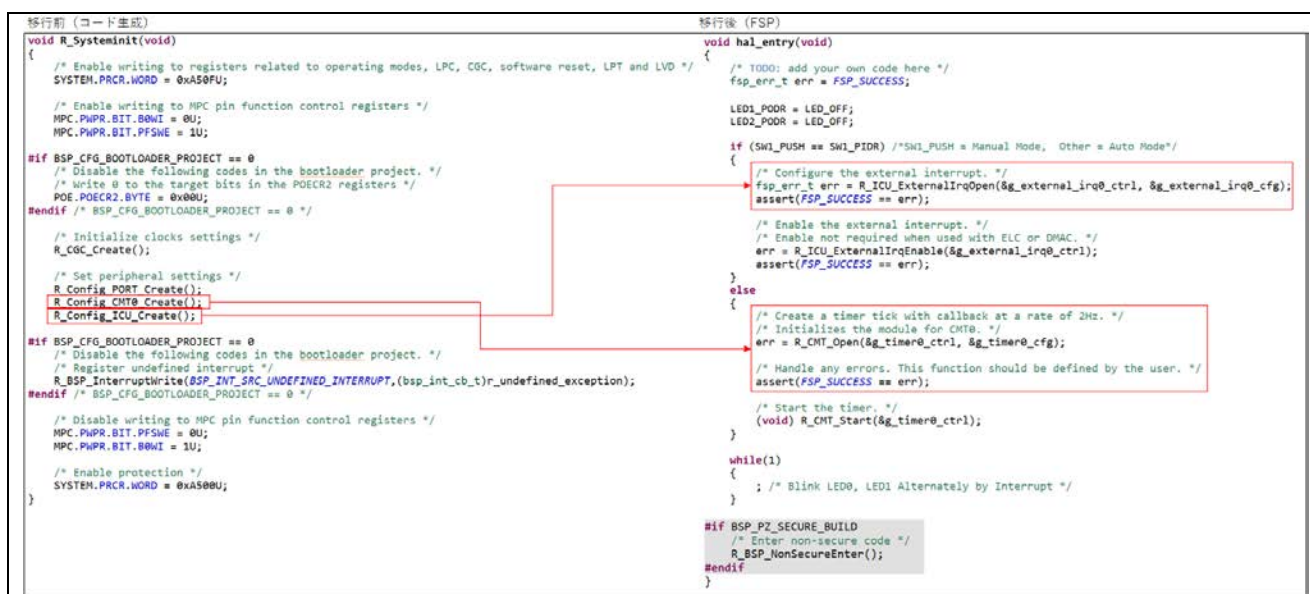


図 7-25 初期化処理

7.1.10.2 メイン処理の比較

移行前 (コード生成)	移行後 (FSP)
<pre> void main(void) { LED1 = LED_OFF; LED2 = LED_OFF; if (SW1_PUSH == SW1) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/ { R_Config_ICU_IRQ0_Start(); } else { R_Config_CMT0_Start(); } while(1) { /* Blink LED0, LED1 Alternately by Interrupt */ } } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; LED1_PODR = LED_OFF; LED2_PODR = LED_OFF; if (SW1_PUSH == SW1_PDIR) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/ { /* Configure the external interrupt. */ fsp_err_t err = R_ICU_ExternalIrqOpen(&g_external_irq0_ctrl, &g_external_irq0_cfg); assert(FSP_SUCCESS == err); /* Enable the external interrupt. */ /* Enable not required when used with ELC or DMAC. */ err = R_ICU_ExternalIrqEnable(&g_external_irq0_ctrl); assert(FSP_SUCCESS == err); } else { /* Create a timer tick with callback at a rate of 2Hz. */ /* Initializes the module for CMT0. */ err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg); /* Handle any errors. This function should be defined by the user. */ assert(FSP_SUCCESS == err); /* Start the timer. */ (void) R_CMT_Start(&g_timer0_ctrl); } while(1) { /* Blink LED0, LED1 Alternately by Interrupt */ } } #if BSP_P2_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif </pre>

図 7-26 メイン処理

7.1.10.3 コールバック関数の比較

移行前 (コード生成)	移行後 (FSP)
<pre> #if FAST_INTERRUPT_VECTOR == VECT_CMT0_CMI0 #pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0),fint) #else #pragma interrupt r_Config_CMT0_cmi0_interrupt(vect=VECT(CMT0,CMI0)) #endif static void r_Config_CMT0_cmi0_interrupt(void) { /* Start user code for r_Config_CMT0_cmi0_interrupt. Do not edit comment generated here */ LED1 = LED2; LED2 = ~LED2; /* End user code. Do not edit comment generated here */ } #if FAST_INTERRUPT_VECTOR == VECT_ICU_IRQ0 #pragma interrupt r_Config_ICU_irq0_interrupt(vect=VECT(ICU,IRQ0),fint) #else #pragma interrupt r_Config_ICU_irq0_interrupt(vect=VECT(ICU,IRQ0)) #endif static void r_Config_ICU_irq0_interrupt(void) { /* Start user code for r_Config_ICU_irq0_interrupt. Do not edit comment generated here */ LED1 = LED2; LED2 = ~LED2; /* End user code. Do not edit comment generated here */ } </pre>	<pre> void cmt0_callback(timer_callback_args_t *p_args) { if (TIMER_EVENT_CYCLE_END == p_args->event) { /* Add application code to be called periodically here. */ LED1_PODR = LED2_PODR; LED2_PODR = ~LED2_PODR; } } void irq30_callback(external_irq_callback_args_t *p_args) { (void) p_args; LED1_PODR = LED2_PODR; LED2_PODR = ~LED2_PODR; } </pre>

図 7-27 コールバック関数

7.2 SCI コード生成のサンプルプログラムを使用した移行例

コード生成の SCI（調歩同期式モード）の機能を使用したサンプルプログラム（以下移行元プロジェクト）を FSP プロジェクトに移行する手順について説明します。

7.2.1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

移行元プロジェクトのフォルダ・ファイル構成と移行対象を以下に示します。

main.c、Config_SCI2_user.c が移行対象です。

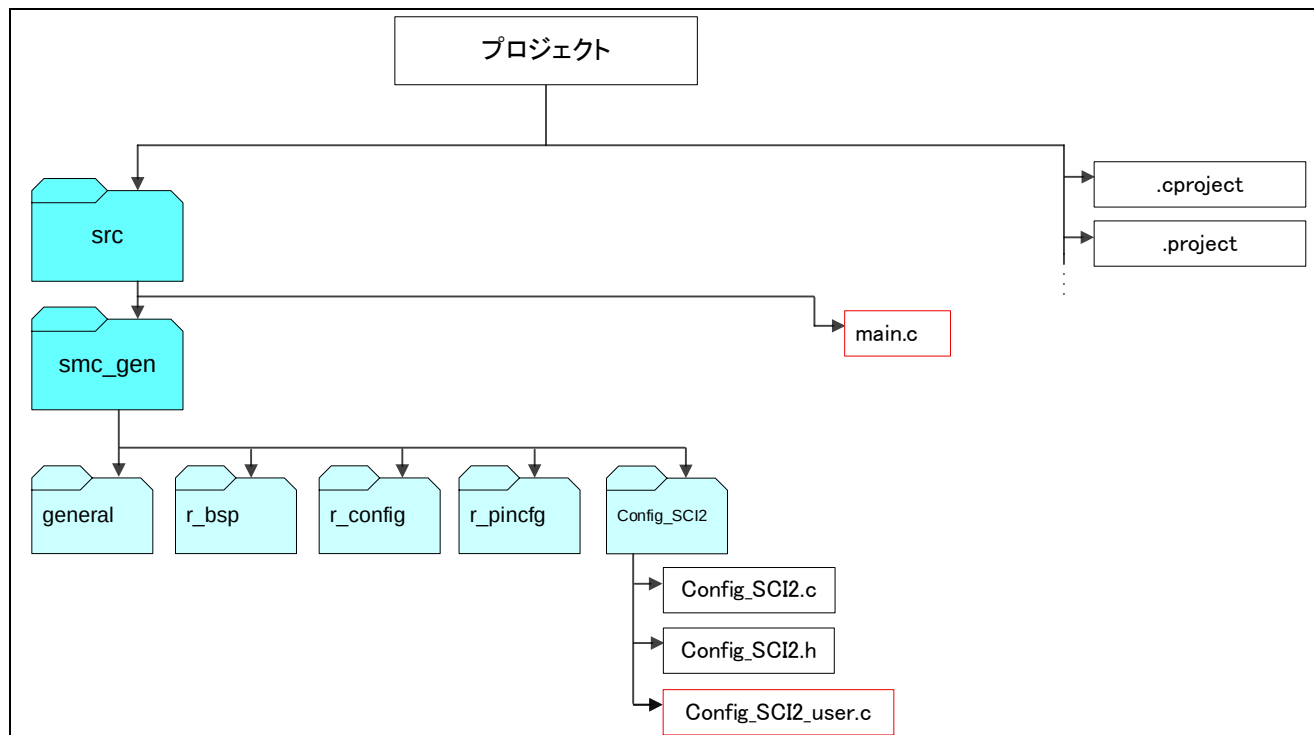


図 7-28 移行元プロジェクトのフォルダ・ファイル構成と移行対象

7.2.2 移行元のプロジェクトの処理概要

main.c は以下のとおりです。

src¥main.c

```

/*****
Includes    <System Includes> , "Project Includes"
*****/

#include "r_smc_entry.h"

/*****
Private global variables and functions
*****/

void main(void);

/*****
* Function Name: main

```

```

* Description : This function implements main function.
* Arguments   : None
* Return Value : None
*****/
void main(void)
{
    /* Set up SCI2 receive buffer */
    R_Config_SCI2_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI2 operations */
    R_Config_SCI2_Start();

    while (1U)
    {
        /* Check the received character */
        if (true == g_tx_flag)
        {
            g_tx_flag = false;

            /* Transmit the data */
            if (MD_OK == R_Config_SCI2_Serial_Send((uint8_t *)&g_tx_char, 1))
            {
                while (false == g_tx_done)
                {
                    /* Wait */
                }
            }
        }
    }
}

```

SCI の初期化は以下の関数でスタートアップ時に実行されます。

src¥smc_gen¥Config_SCI2¥Config_SCI2.c

```

void R_Config_SCI2_Create(void)
{
    /* Cancel SCI stop state */
    MSTP(SCI2) = 0U;

    /* Set interrupt priority */
    IPR(SCI2, RXI2) = _0F_SCI_PRIORITY_LEVEL15;
}

```

```
IPR(SCI2, TXI2) = _0F_SCI_PRIORITY_LEVEL15;

/* Clear the control register */
SCI2.SCR.BYTE = 0x00U;

/* Set clock enable */
SCI2.SCR.BYTE = _00_SCI_INTERNAL_SCK_UNUSED;

/* Clear the SIMR1.IICM, SPMR.CKPH, and CKPOL bit, and set SPMR */
SCI2.SIMR1.BIT.IICM = 0U;
SCI2.SPMR.BYTE = _00_SCI_RTS | _00_SCI_CLOCK_NOT_INVERTED |
_00_SCI_CLOCK_NOT_DELAYED;

/* Set control registers */
SCI2.SMR.BYTE = _01_SCI_CLOCK_PCLK_4 | _00_SCI_MULTI_PROCESSOR_DISABLE |
                _00_SCI_STOP_1 | _00_SCI_PARITY_DISABLE |
                _00_SCI_DATA_LENGTH_8 | _00_SCI_ASYNCHRONOUS_OR_I2C_MODE;
SCI2.SCMR.BYTE = _00_SCI_SERIAL_MODE | _00_SCI_DATA_INVERT_NONE |
                _00_SCI_DATA_LSB_FIRST | _10_SCI_DATA_LENGTH_8_OR_7 |
                _62_SCI_SCMR_DEFAULT;
SCI2.SEMR.BYTE = _00_SCI_BIT_MODULATION_DISABLE | _08_SCI_6_BASE_CLOCK |
                _00_SCI_NOISE_FILTER_DISABLE | _00_SCI_LOW_LEVEL_START_BIT;

/* Set bit rate */
SCI2.BRR = 0x40U;

/* Set RXD2 pin */
MPC.P52PFS.BYTE = 0x0AU;
PORT5.PMR.BYTE |= 0x04U;

/* Set TXD2 pin */
MPC.P50PFS.BYTE = 0x0AU;
PORT5.PODR.BYTE |= 0x01U;
PORT5.PDR.BYTE |= 0x01U;

R_Config_SCI2_Create_UserInit();
}
```

その他に関連する処理としてコード生成のユーザ処理で以下の処理が実装されています。

src¥smc_gen¥Config_SCI2¥Config_SCI2_user.c

(略)

```

/* Start user code for global. Do not edit comment generated here */
/* Global used to receive a character from the PC terminal */
uint8_t g_rx_char;

/* Global used to transmit a character to the PC terminal */
uint8_t g_tx_char;

/* Flag used to control transmission to PC terminal */
volatile uint8_t g_tx_flag = false;

/* Flag used locally to detect transmission complete */
volatile uint8_t g_tx_done;
/* End user code. Do not edit comment generated here */
(中略)
void R_Config_SCI2_Create_UserInit(void)
{
    /* Start user code for user init. Do not edit comment generated here */
    /* End user code. Do not edit comment generated here */
}
(中略)
/*****
* Function Name: r_Config_SCI2_callback_transmitend
* Description  : This function is a callback function when SCI2 finishes
*                transmission
* Arguments    : None
* Return Value : None
*****/

static void r_Config_SCI2_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI2_callback_transmitend. Do not edit
       comment generated here */
    g_tx_done = true;
    /* End user code. Do not edit comment generated here */
}

/*****
* Function Name: r_Config_SCI2_callback_receiveend
* Description  : This function is a callback function when SCI2 finishes
*                reception
*****/

```

```

* Arguments      : None
* Return Value   : None
*****/
static void r_Config_SCI2_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI2_callback_transmitend. Do not edit
       comment generated here */
    g_tx_char = g_rx_char;
    g_tx_flag = true;

    /* Set up SCI2 receive buffer */
    R_Config_SCI2_Serial_Receive((uint8_t *)&g_rx_char, 1);
    /* End user code. Do not edit comment generated here */
}

/*****
* Function Name: r_Config_SCI2_callback_receiveerror
* Description   : This function is a callback function when SCI2 reception
encounters error
* Arguments     : None
* Return Value  : None
*****/
static void r_Config_SCI2_callback_receiveerror(void)
{
    /* Start user code for r_Config_SCI2_callback_transmitend. Do not edit
       comment generated here */
    /* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */

```

移行元のプログラムでは PC のターミナルからデータを受信するとターミナルに受信したデータを送信します。

7.2.3 移行先のプロジェクトの作成

5 FSP 以前のプロジェクトから FSP プロジェクトに移行する手順で示した手順に則り、移行するプロジェクトを作成します。5.1 FSP プロジェクトの新規作成～5.4 モジュールの移行 (FIT→FSP) を参考に新規に作成した FSP プロジェクトをベースにします。

7.2.4 SCI モジュールの追加

以下の方法でプロジェクトに SCI モジュール 1 チャンネル分を追加してください。

[New Stack] ボタンから[Connectivity] → [UART (r_sci_b_uart)]を選択し実行します。

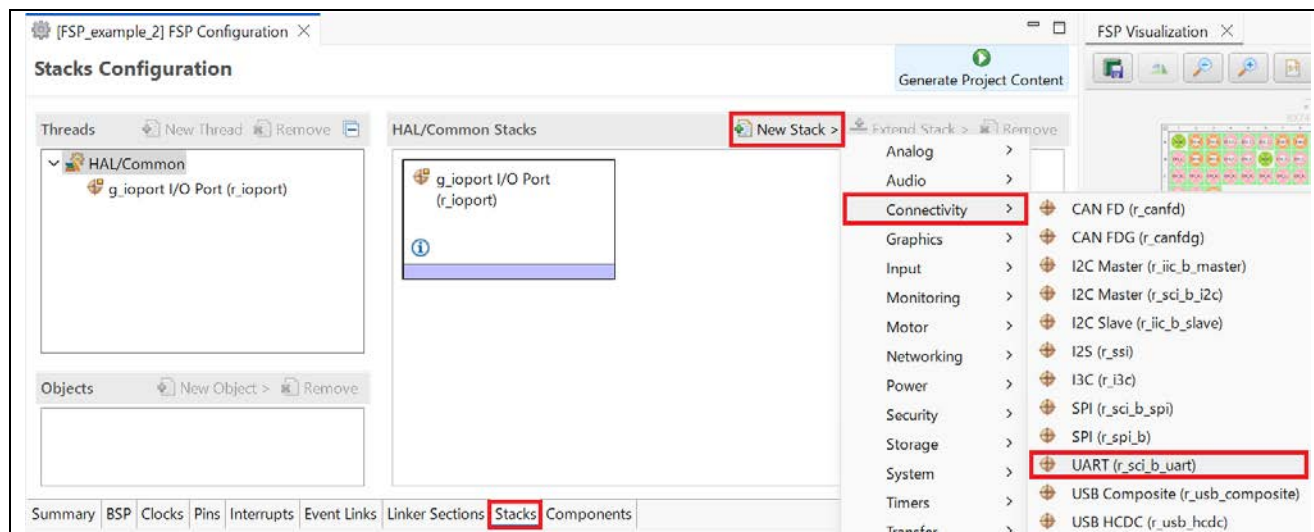


図 7-29 [Stacks Configuration]ページ UART(SCI)の選択

SCI 用の"g_uart0(r_sci_b_uart)"が[Stacks Configuration]ページに追加されます。

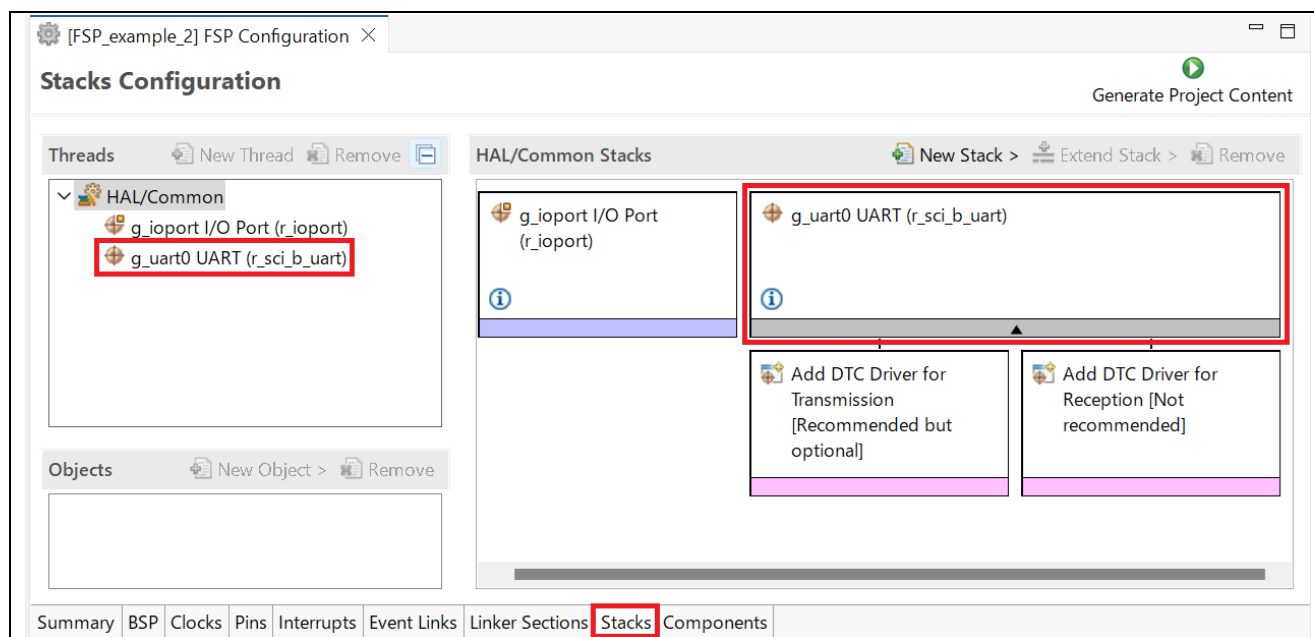


図 7-30 [Stacks Configuration]ページ UART(SCI)モジュール(r_sci_b_uart)の追加

コード生成の SCI の設定に合わせて g_uart0 のプロパティを設定します。

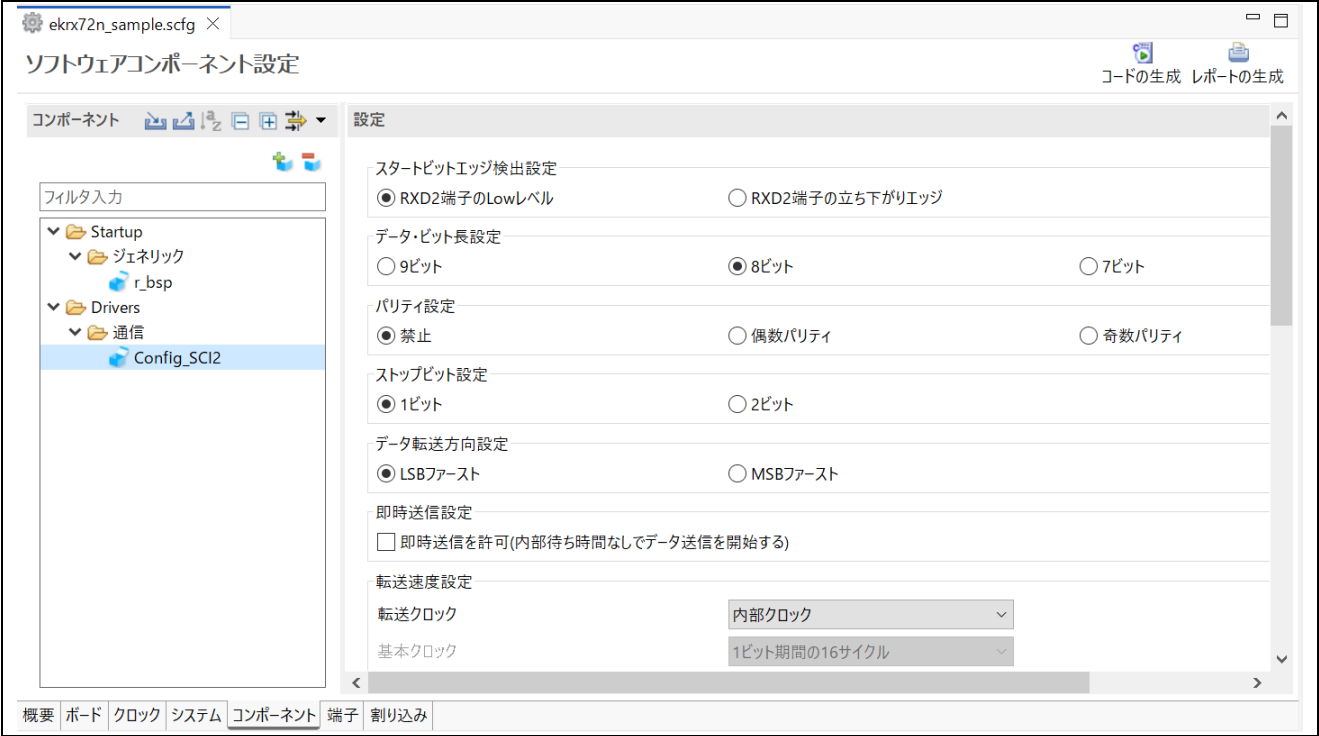


図 7-31 [コンポーネント] SCI コード生成の設定

g_uart0 のプロパティは以下のように設定します。

表 7.2 移行時の設定項目と設定値(1)

No	SCI コンポーネントでの設定項目および設定値	g_uart0 のプロパティでの設定項目および設定値
1	スタートビットエッジ検出設定 RXD2 端子の Low レベルを選択。	Extra>Start bit detection を Low Level に設定。
2	データ・ビット長設定 8 ビットを選択。	General>Data Bits を 8bits に設定。
3	パリティ設定 禁止を選択。	General>Parity を None に設定。
4	ストップビット設定 1 ビットを選択。	General>Stop Bits を 1bit に設定。
5	データ転送方向設定 LSB ファーストを選択。	なし
6	転送速度設定 転送クロックは内部クロックを選択。 基本クロックは 1 ビット期間の 16 サイクルを選択。 ビットレートは 38400 bps を選択。 ビットレートモジュレーション機能を有効はチェックしない。 SCK2 端子機能は[SCK2 を使用しない]を選択。	Extra>Clock Source を Internal Clock に設定。 なし Baud>Baud Rate を 38400 に設定。 Baud>Baud Rate Modulation を Disabled に設定。 なし
7	ノイズフィルタ設定 ノイズフィルタ有効をチェックしない。	Extra>Noise Filter を Disable に設定。

表 7.3 移行時の設定項目と設定値(2)

No	SCI コンポーネントでの設定項目および設定値	g_uart0 のプロパティでの設定項目および設定値
8	ハードウェアフロー制御設定 禁止を選択。	Common>Flow Control Support を Disable に設定。
9	データ一致検出機能 データ一致検出機能有効をチェックしない。	なし
10	データ処理設定 送信データ処理は[割り込みサービスルーチンで処理する]を選択。 受信データ処理は[割り込みサービスルーチンで処理する]を選択。	なし
11	割り込み設定 TXI2 優先順位はレベル 15(最高)を選択。 RXI2 優先順位はレベル 15(最高)を選択。 受信エラー割り込み許可(ERI2)にチェック。 TEI2,ERI2 優先順位(グループ BL0)はレベル 15(最高)を選択。	Interrupts>Transmit Data Empty Interrupt Priority を Priority 15 に設定。
		Interrupts>Receive Interrupt Priority を Priority 15 に設定。
		なし
		Interrupts>Transmit End Interrupt Priority を Priority 15 に設定。
12	多重割り込みの設定 すべての項目をチェックしない。	Interrupts>Error Interrupt Priority を Priority 15 に設定。
		なし
13	コールバック機能設定 送信完了にチェック。 受信完了にチェック。 受信エラーにチェック。	Interrupts>Callback を my_rsci_callback に設定。

表 7.4 移行時の設定項目と設定値(3)

No	SCI コンポーネントでのその他の設定	g_uart0 のプロパティでの設定項目および設定値
1	SCI(SCIF)調歩同期式モード 作業モード：送信/受信	なし
2	リソース：SCI2	General>Channel を 12 に設定。（デバイス・ボード変更により Channel 12 に変更）

上記の設定で設定した g_uart0 のプロパティは以下のとおりです。

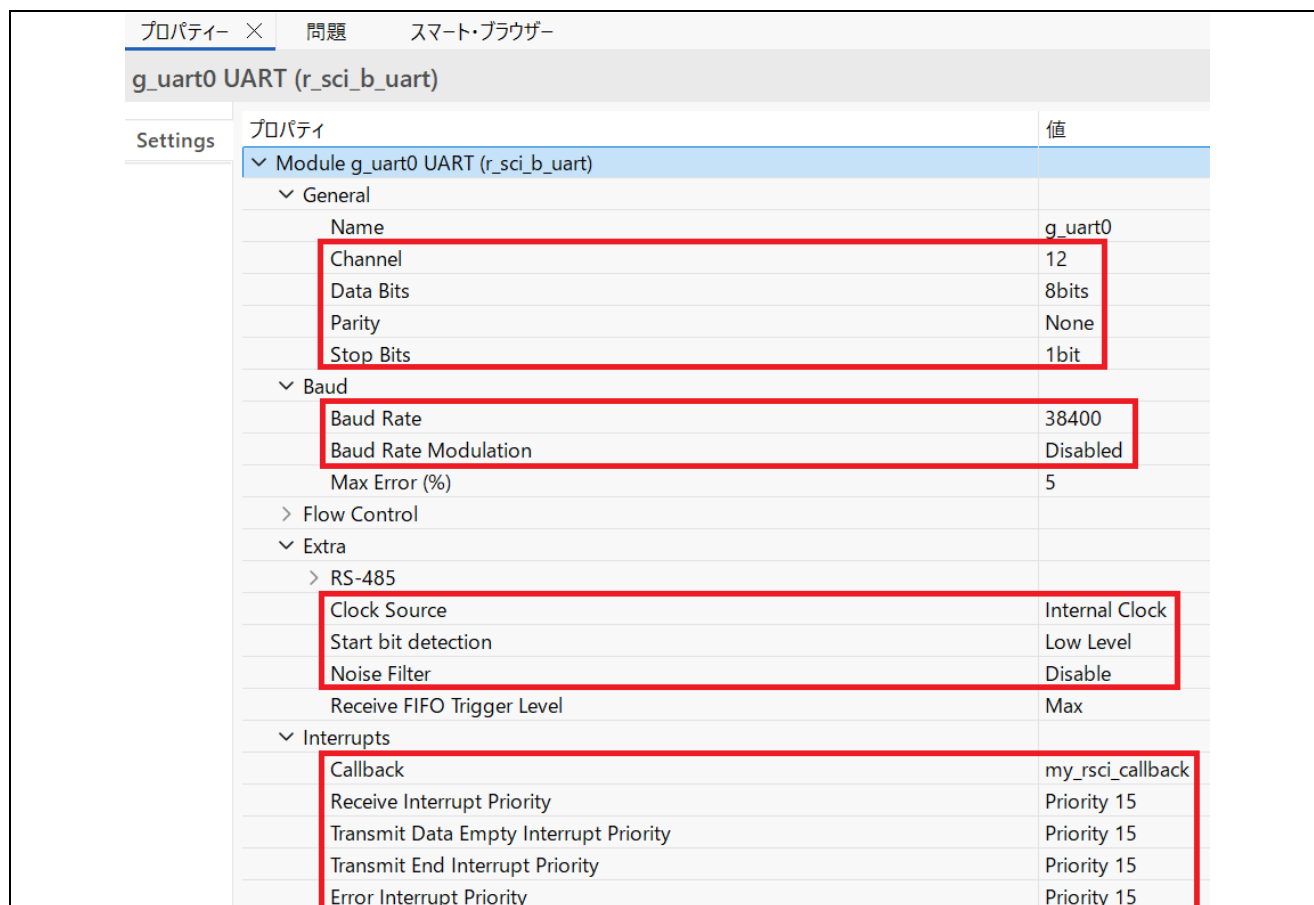


図 7-32 UART(SCI)モジュール(r_sci_b_uart)のプロパティ

コールバック関数と割り込み優先レベルの設定により以下のように RSCI12 の割り込みを使用する情報が登録されます。割り込み関数はモジュールに含まれる割り込み関数名となります。

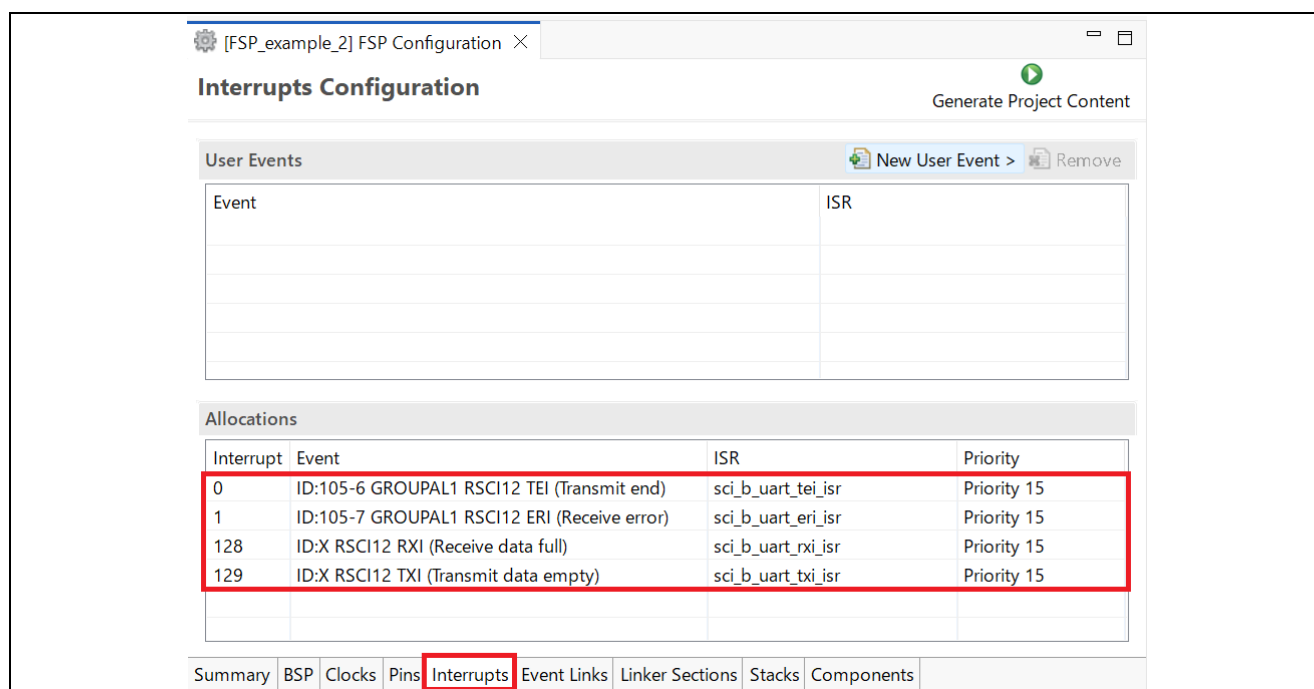


図 7-33 [Interrupts Configuration]ページ UART(SCI)割り込み登録

[Generate Project Content]を実行すると割り込み関数の定義が生成され、vector_data.hに割り込み関数定義の宣言が出力されます。上記の設定で割り込み関数が出力された vector_data.h は以下のようになります。

rx_gen¥vector_data.h

```

1  /* generated vector header file - do not edit */
2  #ifndef VECTOR_DATA_H
3  #define VECTOR_DATA_H
4  #ifdef __cplusplus
5      extern "C" {
6      #endif
7      /* Number of interrupts allocated */
8      #ifndef VECTOR_DATA_IRQ_COUNT
9      #define VECTOR_DATA_IRQ_COUNT    (4)
10     #endif
11     /* ISR prototypes */
12     void sci_b_uart_tei_isr(void);
13     void sci_b_uart_eri_isr(void);
14     void sci_b_uart_rxi_isr(void);
15     void sci_b_uart_txi_isr(void);
16
17     /* Vector table allocations */
18     #define VECTOR_NUMBER_RSCI12_TEI ((IRQn_Type) (6 << 8U | 105)) /*
19     #define RSCI12_TEI_IRQn          ((IRQn_Type) (6 << 8U | 105)) /*
20     #define VECTOR_NUMBER_RSCI12_ERI ((IRQn_Type) (7 << 8U | 105)) /*
21     #define RSCI12_ERI_IRQn         ((IRQn_Type) (7 << 8U | 105)) /*
22     #define VECTOR_NUMBER_RSCI12_RXI ((IRQn_Type) 128) /* ID:X RSCI12
23     #define RSCI12_RXI_IRQn         ((IRQn_Type) 128) /* ID:X RSCI12
24     #define VECTOR_NUMBER_RSCI12_TXI ((IRQn_Type) 129) /* ID:X RSCI12
25     #define RSCI12_TXI_IRQn         ((IRQn_Type) 129) /* ID:X RSCI12

```

図 7-34 vector_data.h UART(SCI)割り込み関数宣言

コールバック関数の宣言は hal_data.h に出力されます。

rx_gen¥hal_data.h

```

1  /* generated HAL header file - do not edit */
2  #ifndef HAL_DATA_H_
3  #define HAL_DATA_H_
4  #include <stdint.h>
5  #include "bsp_api.h"
6  #include "common_data.h"
7  #include "r_sci_b_uart.h"
8  #include "r_uart_api.h"
9  FSP_HEADER
10  /** UART on SCI Instance. */
11  extern const uart_instance_t g_uart0;
12
13  /** Access the UART instance using these structures when calling API functions directly (::p_ap
14  extern sci_b_uart_instance_ctrl_t g_uart0_ctrl;
15  extern const uart_cfg_t g_uart0_cfg;
16  extern const sci_b_uart_extended_cfg_t g_uart0_cfg_extend;
17
18  #ifndef my_rsci_callback
19  void my_rsci_callback(uart_callback_args_t *p_args);
20  #endif
21  void hal_entry(void);
22  void g_hal_init(void);
23  FSP_FOOTER
24  #endif /* HAL_DATA_H_ */
25

```

図 7-35 hal_data.h UART(SCI)コールバック関数宣言

割り込み関数の実体はモジュール内に定義されていますが、

rx¥fsp¥src¥r_sci_b_uart¥r_sci_b_uart.c

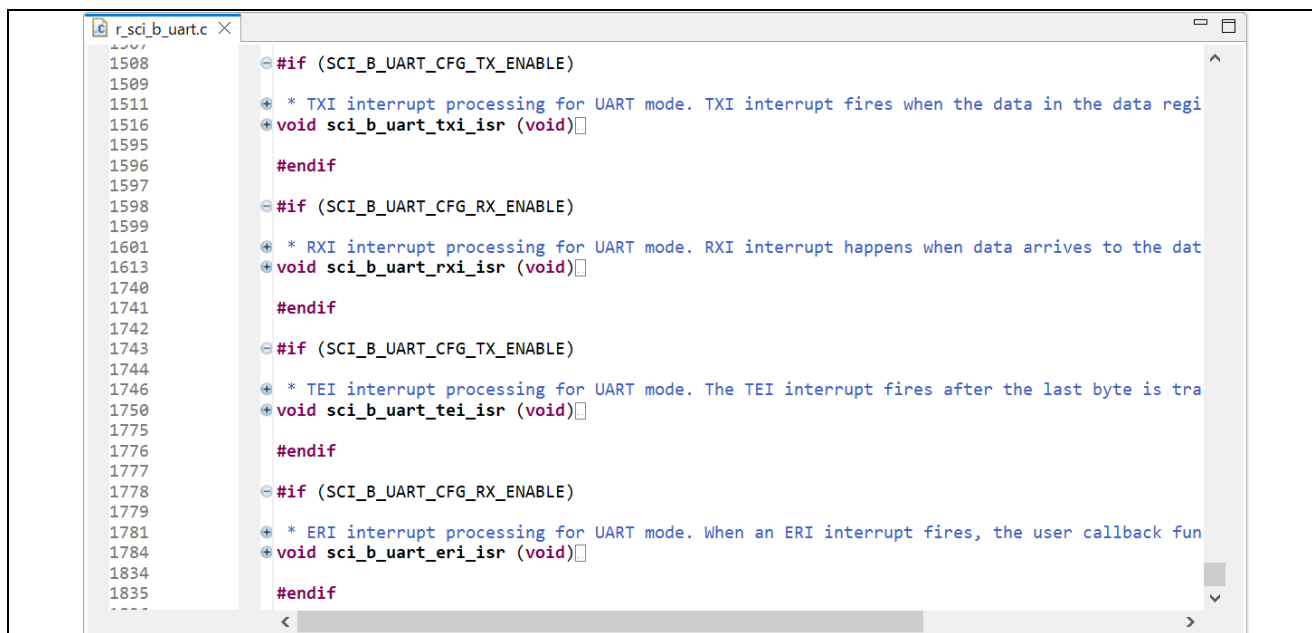


図 7-36 r_sci_b_uart.c 割り込み関数

コールバック関数は宣言のみが出力されるため、コールバック関数の実体はユーザ自身で作成する必要があります。

7.2.5 PORT 設定の追加

移行元プロジェクトでは SCI 端子の端子設定のために Config_Port を使用していました。移行先プロジェクトでは[Pin Configuration]ページで設定することによりスタートアップで呼び出される IOPORT モジュールの初期設定処理を用いて設定するように変更します。

SCI2 として RXD2 に P52、TXD2 に P50 が設定されていましたが、移行先ではポートの割り当てを変更します。



図 7-37 [端子] SCI の設定

g_uart0 のプロパティの Pins から RXD と TXD 端子の割り当てを行います。なおプロジェクト作成時にボードを選択した場合はボード情報によって周辺機能および端子が最初から割り当てられていることがあります。



図 7-38 UART(SCI)モジュール(r_sci_b_uart)のプロパティ Pins

端子の変更ボタンから[Pin Configuration]ページに切り替わるので



図 7-39 UART(SCI)モジュール(r_sci_b_uart)のプロパティ Pins 変更

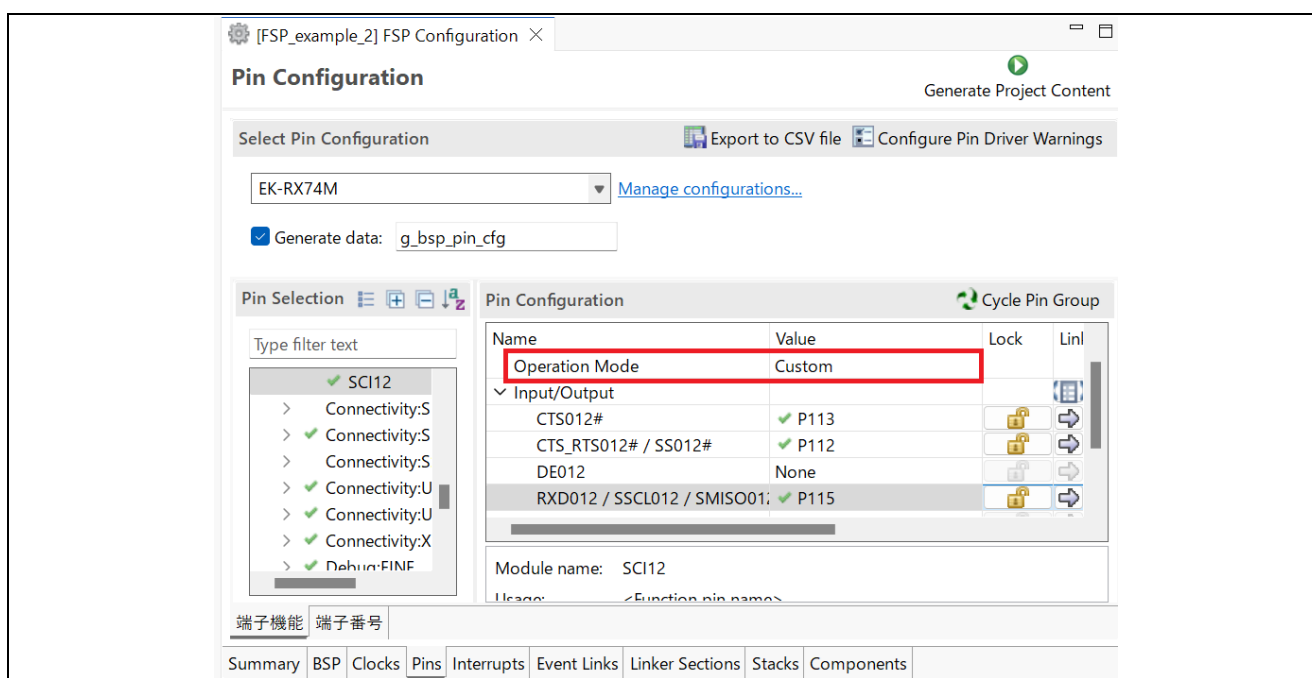


図 7-40 [Pin Configuration]ページ SCI12 Operation Mode 初期設定

EK-RX74M では Operation Mode は"Custom"に初期設定されており、RXD、TXD 端子もボードの仕様に合わせたポートが割り当てられています。Operation Mode を"Asynchronous UART"に変更します。

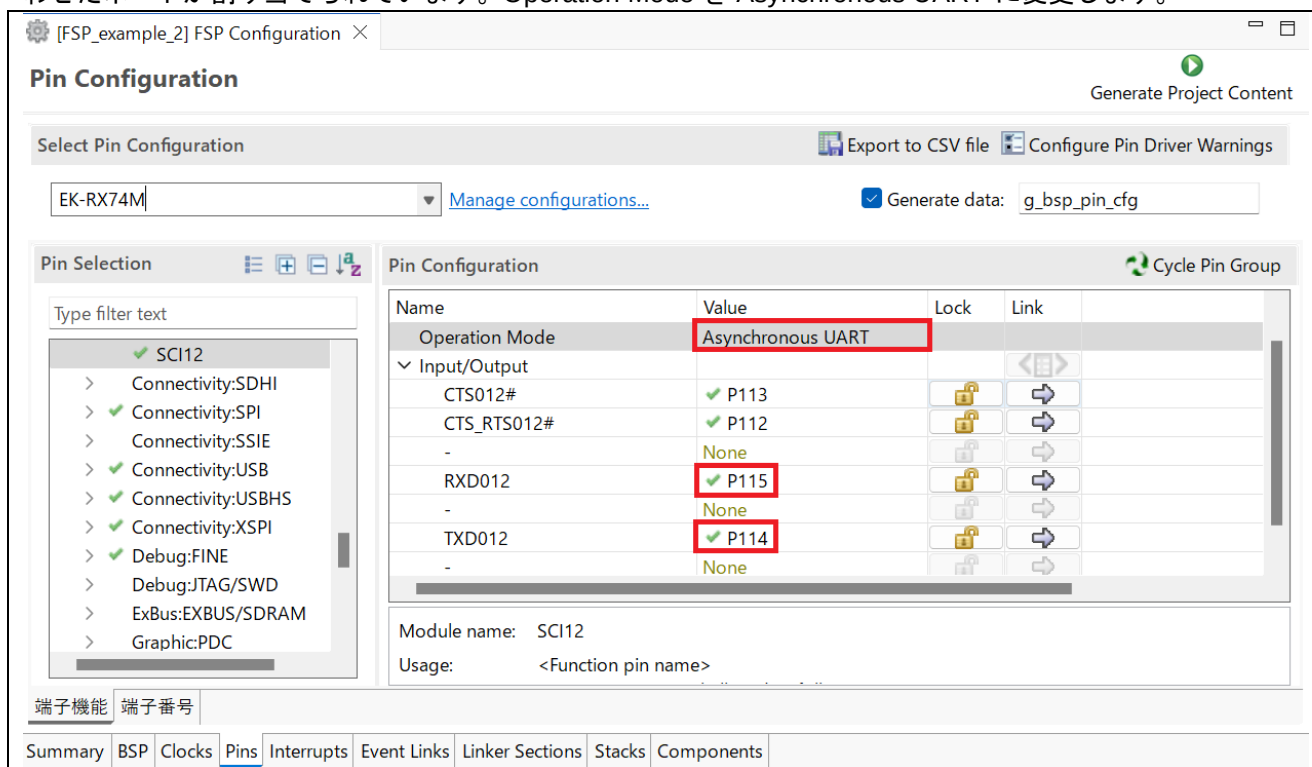


図 7-41 [Pin Configuration]ページ SCI12 Operation Mode UART

7.2.6 移行後のプロジェクトのフォルダ・ファイル構成と変更対象

移行後のプロジェクトのフォルダ・ファイル構成と変更対象を以下に示します。hal_entry.c が変更対象です。

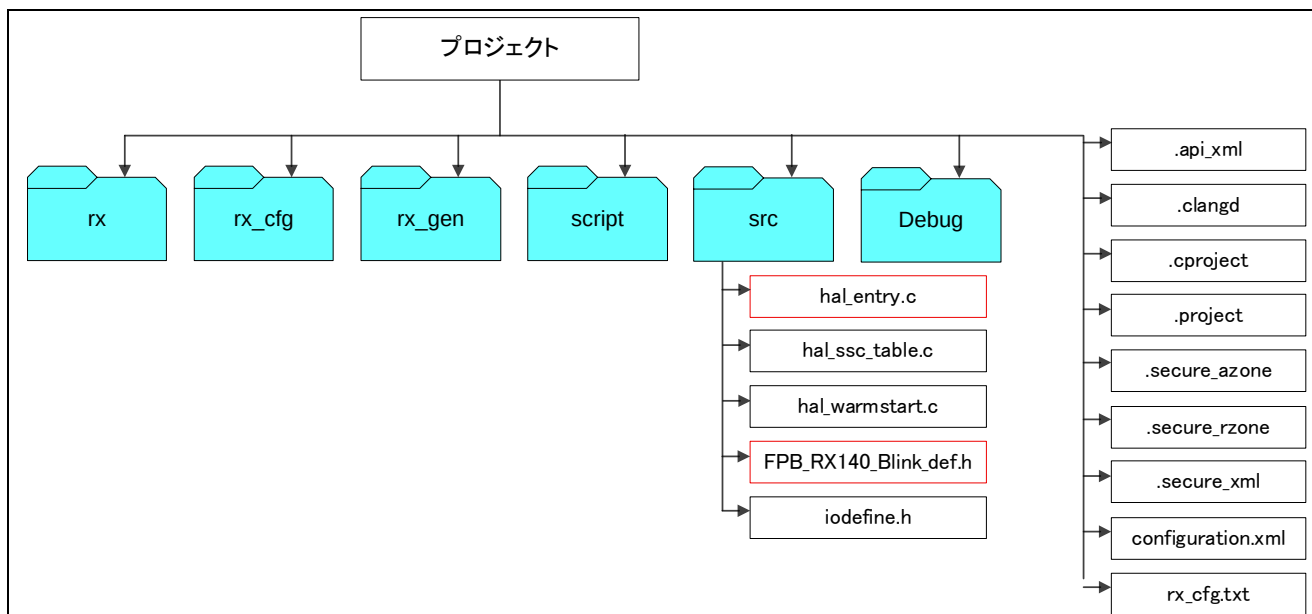


図 7-42 移行後プロジェクトのフォルダ・ファイル構成と変更対象

7.2.7 移行のポイント

5.7 ソースコードの移植に則り、main.c の main 関数本体の処理と SCI のユーザ処理をコールバック関数として hal_entry.c に移植します。スタートアップで行われる初期化処理はモジュールに置き換えるため、移植しません。

5.9 FSP モジュールの処理への置き換えを参考にコード生成の処理を FSP の処理へ置き換えます。

7.2.8 変更対象のファイルの置き換え後の処理

変更した最終的な hal_entry.c は以下のようになります。

hal_entry.c （赤字が最終的に追加した部分となります。）

```
#include "hal_data.h"

/* Global used to receive a character from the PC terminal */
uint8_t g_rx_char;

/* Global used to transmit a character to the PC terminal */
uint8_t g_tx_char;

/* Flag used to control transmission to PC terminal */
volatile uint8_t g_tx_flag = false;

/* Flag used locally to detect transmission complete */
volatile uint8_t g_tx_done;

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    fsp_err_t err = FSP_SUCCESS;

    /* Called the R_SCI_B_UART_Open to open the transfer instance
       with initial configuration. */
    err = g_uart0.p_api->open(&g_uart0_ctrl, &g_uart0_cfg);
    assert(FSP_SUCCESS == err);
```

```
/* Called the R_SCI_B_UART_Read to read data from UART. */
err = g_uart0.p_api->read(&g_uart0_ctrl, &g_rx_char, 1);
assert(FSP_SUCCESS == err);

while (1U)
{
    /* Check the received character */
    if (true == g_tx_flag)
    {
        g_tx_flag = false;

        /* Transmit the data */
        /* Called the R_SCI_B_UART_Write to write data to UART. */
        err = g_uart0.p_api->write(&g_uart0_ctrl, &g_tx_char, 1);
        assert(FSP_SUCCESS == err);

        while (false == g_tx_done)
        {
            /* Wait */
        }
    }
}

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
* Function Name: my_rsci_callback
* Description   : This is a template for an RSCI Async Mode callback function.
* Arguments      : pArgs -
*                  pointer to rsci_cb_args_t structure cast to a void. Structure
*                  contains event and associated data.
* Return Value   : none
*****/
void my_rsci_callback(uart_callback_args_t *p_args)
{
```

```
fsp_err_t err = FSP_SUCCESS;

/* Handle the UART event */
switch (p_args->event)
{
    /* Received a character */
    case UART_EVENT_RX_CHAR:
    {
        break;
    }

    /* Receive complete */
    case UART_EVENT_RX_COMPLETE:
    {
        g_tx_char = g_rx_char;
        g_tx_flag = true;

        /* Called the R_SCI_B_UART_Read to read data from UART. */
        err = g_uart0.p_api->read(&g_uart0_ctrl, &g_rx_char, 1);
        assert(FSP_SUCCESS == err);
        break;
    }

    /* Transmit complete */
    case UART_EVENT_TX_COMPLETE:
    {
        g_tx_done = true;
        break;
    }

    default:
    {
    }
}
}
```

7.2.9 本移行例におけるコード生成の処理の FSP モジュールへの処理の置き換え

移行前のコード生成の処理と移行し置き換え後の FSP モジュールの処理の比較を以下に示します。

7.2.9.1 モジュールの初期化処理の比較

移行前 (コード生成)	移行後 (FSP)
<pre> void R_Systeminit(void) { /* Enable writing to registers related to operating modes, LPC, CGC and software reset */ SYSTEM.PRCR.WORD = 0xA500U; /* Enable writing to MPC pin function control registers */ MPC.PWPR.BIT.B0W1 = 0U; MPC.PWPR.BIT.PFSWE = 1U; #if BSP_CFG_BOOTLOADER_PROJECT == 0 /* Disable the following codes in the bootloader project. */ /* Write 0 to the target bits in the POECR2 registers */ POE3.POECR2.WORD = 0x0000U; #endif /* BSP_CFG_BOOTLOADER_PROJECT == 0 */ /* Initialize clocks settings */ R_CGC_Create(); /* Set peripheral settings */ R_Config_SCI2_Create(); /* Set interrupt settings */ R_Interrupt_Create(); #if BSP_CFG_BOOTLOADER_PROJECT == 0 /* Disable the following codes in the bootloader project. */ /* Register undefined interrupt */ R_BSP_InterruptWrite(BSP_INT_SRC_UNDEFINED_INTERRUPT, (bsp_int_cb_t)r_undefined_exception); #endif /* BSP_CFG_BOOTLOADER_PROJECT == 0 */ /* Register group BL0 interrupt TEI2 (SCI2) */ R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI2_TEI2, (bsp_int_cb_t)r_Config_SCI2_transmitend_interrupt); /* Register group BL0 interrupt ERI2 (SCI2) */ R_BSP_InterruptWrite(BSP_INT_SRC_BL0_SCI2_ERI2, (bsp_int_cb_t)r_Config_SCI2_receiveerror_interrupt); /* Disable writing to MPC pin function control registers */ MPC.PWPR.BIT.PFSWE = 0U; MPC.PWPR.BIT.B0W1 = 1U; /* Enable protection */ SYSTEM.PRCR.WORD = 0xA500U; } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; /* Called the R_SCI_B_UART_Open to open the transfer instance with initial configuration. */ err = g_uart0.p_api->open(&g_uart0_ctrl, &g_uart0_cfg); assert(FSP_SUCCESS == err); /* Called the R_SCI_B_UART_Read to read data from UART. */ err = g_uart0.p_api->read(&g_uart0_ctrl, &g_rx_char, 1); assert(FSP_SUCCESS == err); while (1U) { /* Check the received character */ if (true == g_tx_flag) { g_tx_flag = false; /* Transmit the data */ /* Called the R_SCI_B_UART_Write to write data to UART. */ err = g_uart0.p_api->write(&g_uart0_ctrl, &g_tx_char, 1); assert(FSP_SUCCESS == err); while (false == g_tx_done) { /* Wait */ } } } #if BSP_PZ_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

図 7-43 初期化処理

7.2.9.2 メイン処理の比較

移行前 (コード生成)	移行後 (FSP)
<pre> void main(void) { LED1 = LED_OFF; LED2 = LED_OFF; if (SW1_PUSH == SW1) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/ { R_Config_ICU_IRQ0_Start(); } else { R_Config_CMT0_Start(); } while(1) { /* Blink LED0, LED1 Alternately by Interrupt */ } } </pre>	<pre> void hal_entry(void) { /* TODO: add your own code here */ fsp_err_t err = FSP_SUCCESS; LED1_PODR = LED_OFF; LED2_PODR = LED_OFF; if (SW1_PUSH == SW1_PIDR) /*SW1_PUSH = Manual Mode, Other = Auto Mode*/ { /* Configure the external interrupt. */ fsp_err_t err = R_ICU_ExternalIrqOpen(&g_external_irq0_ctrl, &g_external_irq0_cfg); assert(FSP_SUCCESS == err); /* Enable the external interrupt. */ /* Enable not required when used with EIC or DMAC. */ err = R_ICU_ExternalIrqEnable(&g_external_irq0_ctrl); assert(FSP_SUCCESS == err); } else { /* Create a timer tick with callback at a rate of 2Hz. */ /* Initializes the module for CMT0. */ err = R_CMT_Open(&g_timer0_ctrl, &g_timer0_cfg); /* Handle any errors. This function should be defined by the user. */ assert(FSP_SUCCESS == err); /* Start the timer. */ (void) R_CMT_Start(&g_timer0_ctrl); } while(1) { /* Blink LED0, LED1 Alternately by Interrupt */ } #if BSP_PZ_SECURE_BUILD /* Enter non-secure code */ R_BSP_NonSecureEnter(); #endif } </pre>

図 7-44 メイン処理

7.2.9.3 コールバック関数の比較

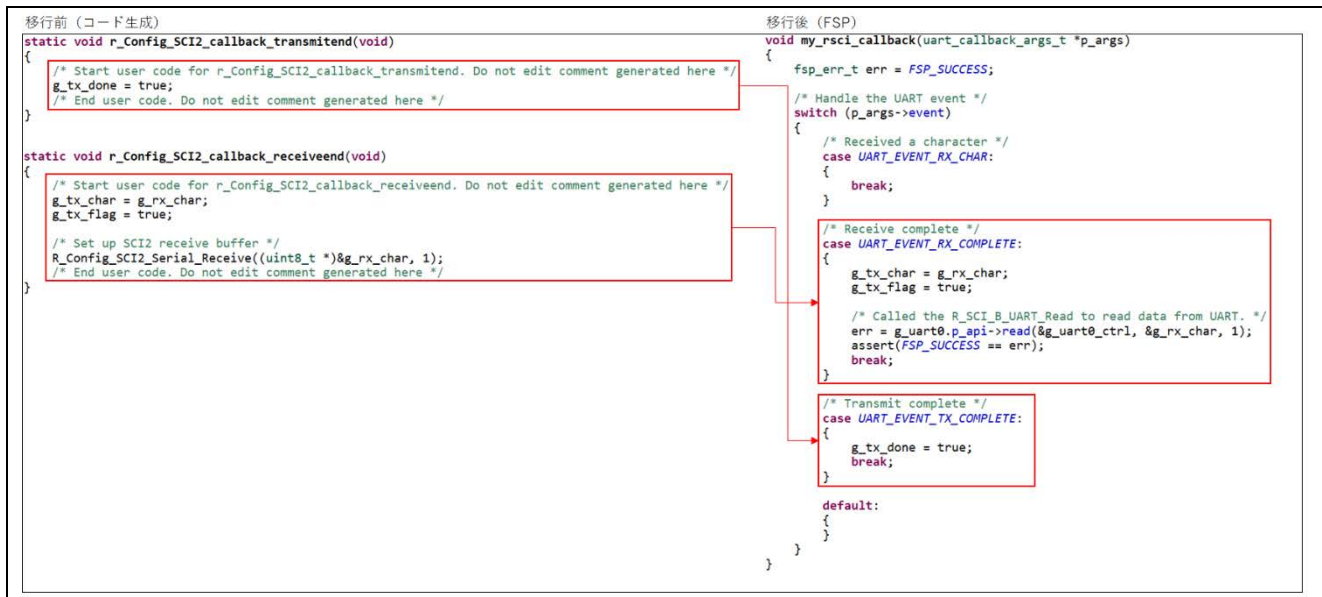


図 7-45 コールバック関数

8. スマート・コンフィグレータを使用しないプロジェクトの FSP プロジェクトへの移行例

[初期設定例のアプリケーションノート](#)をベースに使用したスマート・コンフィグレータを使用しないプロジェクトのサンプルプログラムを例として FSP プロジェクトへ移行する方法について説明します。

8.1 割り込み関数を含まないサンプルプログラムを使用した移行例

割り込みの機能を使用しないサンプルプログラム（以下移行元プロジェクト）を FSP プロジェクトに移行する手順について説明します。

8.1.1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

移行元プロジェクトのフォルダ・ファイル構成と移行対象を以下に示します。

main.c、led_define.h、led_control.c が移行対象です。

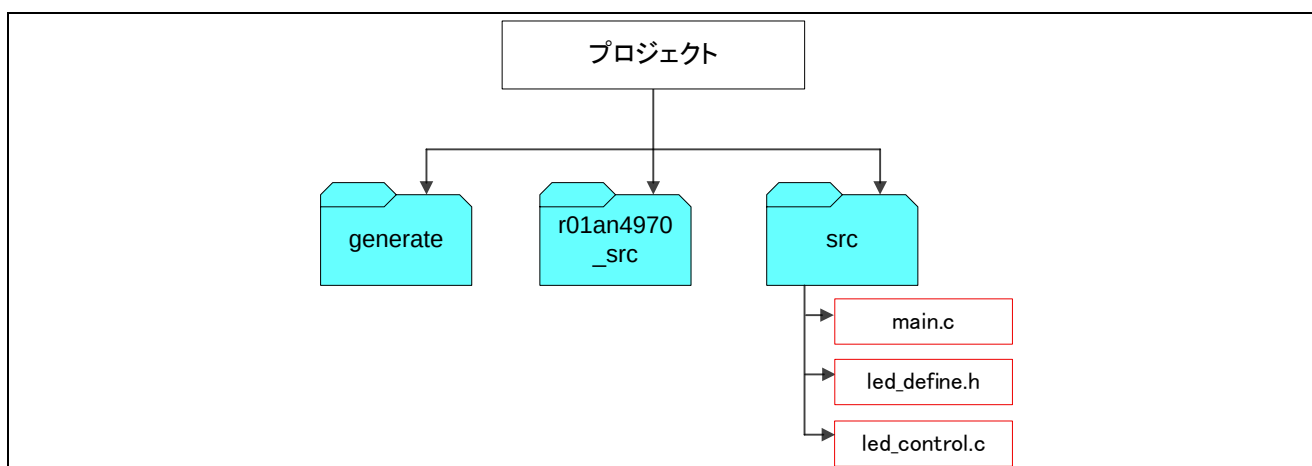


図 8-1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

8.1.2 移行元のプロジェクトの処理概要

main.c は以下のとおりです。

src¥main.c

```
#include <stdint.h>
#include "led_define.h"

static void led_blinking(void);

int main(void)
{
    /* ---- Stopping the peripherals which start operations ---- */
    R_INIT_StopModule();

    /* ---- Initialization of the non-existent ports ---- */
    R_INIT_Port_Initialize();
}
```

```
/* ---- Initialization of the clock ---- */
R_INIT_Clock();

/* ---- Initialization of the ROM Cache ---- */
R_INIT_ROM_Cache();

/* Initialize for application */
led_port_init();

while(1)
{
    // TODO: add application code here
    led_blinking();
}

return 0;
}

/*****
* Function Name: led_blinking
* Description   : Blink the User LED.
* Arguments    : none
* Return Value  : none
*****/
static void led_blinking(void)
{
    uint32_t volatile i;

    led_output(LED_ON);

    /* wait */
    for(i = 0; i <= 100000000; i++)
    {

    }

    led_output(LED_OFF);

    /* wait */
    for(i = 0; i <= 100000000; i++)
```

```
{
}
}
```

src\led_define.h

```
#ifndef LED_DEFINE_H_
#define LED_DEFINE_H_

/*****
Macro definitions
*****/

#define LED_ON          (0U)
#define LED_OFF         (1U)

/*****
Functions Prototypes
*****/

extern void led_port_init(void);
extern void led_output(uint8_t level);

#endif /* LED_DEFINE_H_ */
```

led_control.c

```
/*****
Includes <System Includes> , "Project Includes"
*****/

#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Macro definitions
*****/

#define PORT_OUTPUT      (1U)
#define USER_LED        (PORT4.PODR.BIT.B0)

/*****
Exported global variables and functions (to be accessed by other files)
*****/

void led_port_init(void);
```

```
void led_output(uint8_t level);

/*****
 * Function Name: led_port_init
 * Description   : Initialize the User LED port.
 * Arguments    : none
 * Return Value  : none
 *****/
void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORT4.PDR.BIT.B0 = PORT_OUTPUT;
    PORT4.PMR.BIT.B0 = 0U;
}

/*****
 * Function Name: led_output
 * Description   : Turns the User LED on or off.
 * Arguments    : level : output level
 * Return Value  : none
 *****/
void led_output(uint8_t level)
{
    USER_LED = level;
}
```

移行元のプログラムではLEDを点滅します。

8.1.3 移行先のプロジェクトの作成

5 FSP 以前のプロジェクトから FSP プロジェクトに移行する手順で示した手順に則り、移行するプロジェクトを作成します。5.1 FSP プロジェクトの新規作成～5.4 モジュールの移行 (FIT→FSP) を参考に新規に作成した FSP プロジェクトをベースにします。

8.1.4 移行後のプロジェクトのフォルダ・ファイル構成と変更対象

移行後のプロジェクトのフォルダ・ファイル構成と変更対象を以下に示します。hal_entry.c、led_control.c が変更対象です。

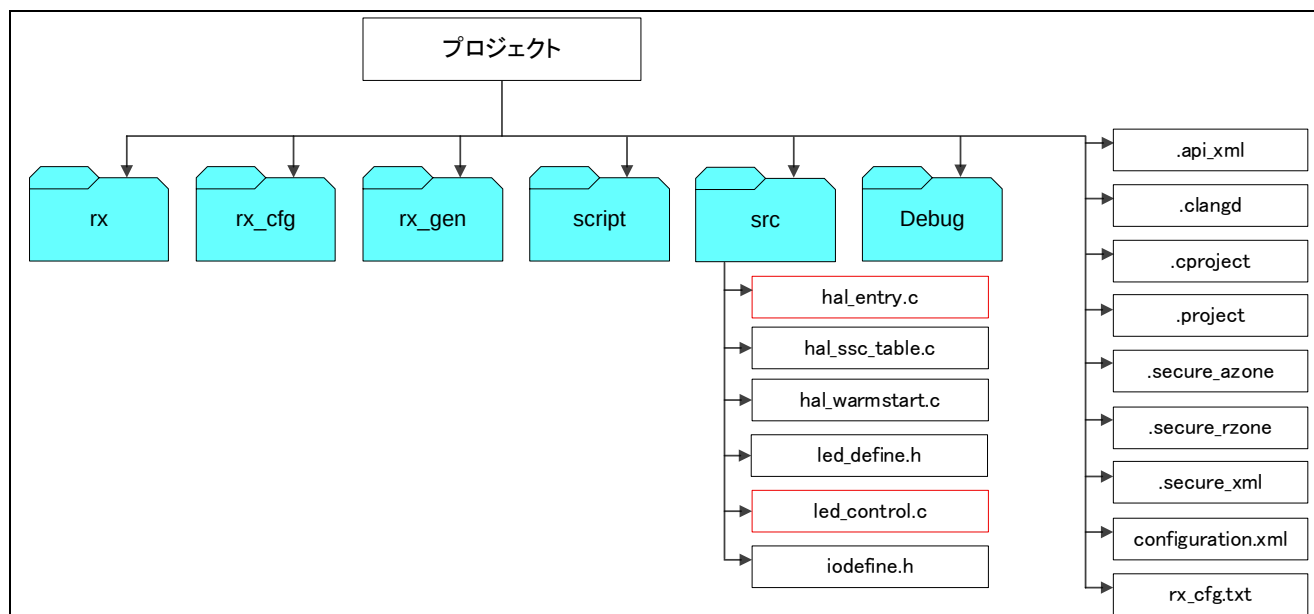


図 8-2 移行後プロジェクトのフォルダ・ファイル構成と変更対象

8.1.5 移行のポイント

5.7 ソースコードの移植に則り、main.c の main 関数本体の処理と関連する処理を hal_entry.c に移植します。[初期設定例のアプリケーションノート](#)が提供する main 関数内の処理では

クロックの初期設定を R_INIT_Clock()で行ういくつか初期設定を行う関数が提供されますが、FSP プロジェクトでは main 関数に到達した時点でこれらのクロック設定を含む初期設定が完了しています。R_INIT_Clock()含むこれらの関数で再設定を行うと関連する設定が競合することになるため、初期設定例が提供する main 処理部分は移行しません。led_define.h、led_control.c を移植します。led_control.c が iodef.h の定義を使用した定義を含むので 5.8 iodef.h の追加に則り iodef.h を追加します。デバイスが変わるため、移行前は PORT40 を使用していましたが、移行後は PORTF06 に変更しています。

5.9 FSP モジュールの処理への置き換えを参考に FIT モジュールの処理を FSP の処理へ置き換えます。

8.1.6 変更対象のファイルの置き換え後の処理

変更した最終的な hal_entry.c および led_control.c は以下のようになります。

hal_entry.c （赤字が最終的に追加した部分となります。）

```
#include "hal_data.h"
#include "led_define.h"

FSP_CPP_HEADER
FSP_CPP_FOOTER

static void led_blinking(void);

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
```

```
* RTOS is used.
*****/

void hal_entry(void)
{
    /* TODO: add your own code here */
    /* Initialize for application */
    led_port_init();

    while(1)
    {
        /* TODO: add application code here */
        led_blinking();
    }

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
* Function Name: led_blinking
* Description   : Blink the User LED.
* Arguments    : none
* Return Value  : none
*****/
static void led_blinking(void)
{
    uint32_t volatile i;

    led_output(LED_ON);

    /* wait */
    for(i = 0; i <= 10000000; i++)
    {

    }

    led_output(LED_OFF);
}
```

```

/* wait */
for(i = 0; i <= 10000000; i++)
{

}
}

```

led_control.c (赤字が変更した部分となります。)

```

/*****
Includes  <System Includes> , "Project Includes"
*****/

#include <stdint.h>
#include "iodef.h"
#include "led_define.h"

/*****
Macro definitions
*****/

#define PORT_OUTPUT      (1U)
#define USER_LED        (PORTF.PODR.BIT.PODR06)

/*****
Exported global variables and functions (to be accessed by other files)
*****/

void led_port_init(void);
void led_output(uint8_t level);

/*****
* Function Name: led_port_init
* Description  : Initialize the User LED port.
* Arguments    : none
* Return Value : none
*****/

void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORTF.PDR.BIT.PDR06 = PORT_OUTPUT;
    PFS.PF06PFS.BIT.PMR = 0U;
}

```

```
/******  
* Function Name: led_output  
* Description  : Turns the User LED on or off.  
* Arguments    : level : output level  
* Return Value : none  
*****/  
void led_output(uint8_t level)  
{  
    USER_LED = level;  
}
```

8.2 割り込み関数を含むサンプルプログラムを使用した移行例

割り込みの機能を使用したサンプルプログラム（以下移行元プロジェクト）を FSP プロジェクトに移行する手順について説明します。

8.2.1 移行元プロジェクトのフォルダ・ファイル構成と移行対象

移行元プロジェクトのフォルダ・ファイル構成と移行対象を以下に示します。

main.c、led_define.h、led_control.c、cmt_define.h、cmt_control.c が移行対象です。

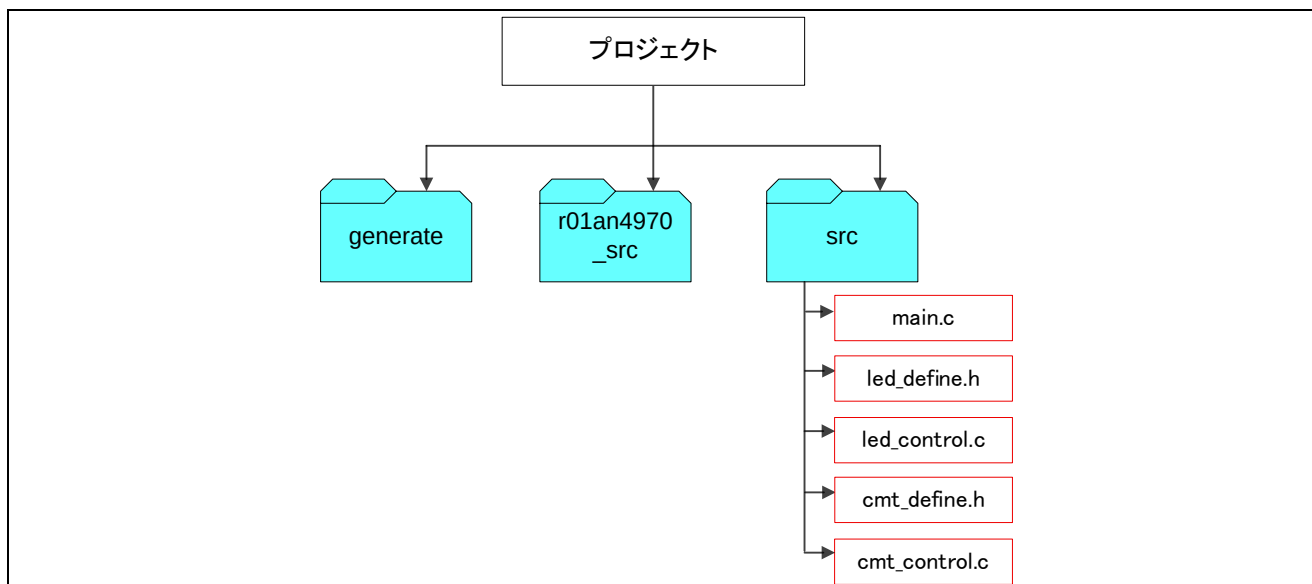


図 8-3 移行元プロジェクトのフォルダ・ファイル構成と移行対象

8.2.2 移行元のプロジェクトの処理概要

main.c は以下のとおりです。

src¥main.c

```
#include <stdint.h>
#include "led_define.h"
#include "cmt_define.h"

int main(void)
{
    /* ---- Stopping the peripherals which start operations ---- */
    R_INIT_StopModule();

    /* ---- Initialization of the non-existent ports ---- */
    R_INIT_Port_Initialize();

    /* ---- Initialization of the clock ---- */
    R_INIT_Clock();

    /* ---- Initialization of the ROM Cache ---- */
}
```

```

R_INIT_ROM_Cache();

/* Initialize for application */
led_port_init();
cmt_initialize();
cmt_start();

while(1)
{
    // TODO: add application code here
}

return 0;
}

```

led_define.h

```

#ifndef LED_DEFINE_H_
#define LED_DEFINE_H_

/*****
Macro definitions
*****/

#define LED_ON          (0U)
#define LED_OFF         (1U)

/*****
Functions Prototypes
*****/

extern void led_port_init(void);
extern void led_output(uint8_t level);
extern void led_output_inversion(void);

#endif /* LED_DEFINE_H_ */

```

led_control.c

```

/*****
Includes <System Includes> , "Project Includes"
*****/

#include <stdint.h>
#include "iodef.h"

```

```
#include "led_define.h"

/*****
Macro definitions
*****/

#define PORT_OUTPUT      (1U)
#define USER_LED        (PORT4.PODR.BIT.B0)

/*****
Exported global variables and functions (to be accessed by other files)
*****/

void led_port_init(void);
void led_output(uint8_t level);
void led_output_inversion(void);

/*****
* Function Name: led_port_init
* Description   : Initialize the User LED port.
* Arguments    : none
* Return Value  : none
*****/

void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORT4.PDR.BIT.B0 = PORT_OUTPUT;
    PORT4.PMR.BIT.B0 = 0U;
}

/*****
* Function Name: led_output
* Description   : Turns the User LED on or off.
* Arguments    : level : output level
* Return Value  : none
*****/

void led_output(uint8_t level)
{
    USER_LED = level;
}
```

```

/*****
 * Function Name: led_output_inversion
 * Description  : Inverting the LED output.
 * Arguments    : none
 * Return Value : none
 *****/
void led_output_inversion(void)
{
    if(USER_LED == LED_OFF)
    {
        led_output(LED_ON);
    }
    else
    {
        led_output(LED_OFF);
    }
}

```

cmt_define.h

```

#ifndef CMT_DEFINE_H_
#define CMT_DEFINE_H_

/*****
Functions Prototypes
 *****/

extern void cmt_initialize(void);
extern void cmt_start(void);
extern void cmt_cb_isr(void);

#endif /* CMT_DEFINE_H_ */

```

cmt_control.c

```

/*****
Includes    <System Includes> , "Project Includes"
 *****/

#include <stdint.h>
#include "idefine.h"
#include "led_define.h"

/*****

```

```

Exported global variables and functions (to be accessed by other files)
*****/

void cmt_initialize(void);
void cmt_start(void);
void cmt_cb_isr(void);

/*****
* Function Name: cmt_initialize
* Description  : Initialize the cmt.
* Arguments    : none
* Return Value : none
*****/
void cmt_initialize(void)
{
    /* initialize cmt */
    /*** 周辺モジュール設定 ***/
    SYSTEM.PRCR.WORD = 0xA502U;
    SYSTEM.MSTPCRA.BIT.MSTPA15 = 0U;
    SYSTEM.PRCR.WORD = 0xA500U;

    IR(CMT0, CMI0) = 0U;
    IPR(CMT0, CMI0) = 0x0FU;

    /* The CMT interrupt interval is set to 0.5 seconds(2 Hz). */
    CMT0.CMCR.WORD = 0x0043U; /* PCLKB:60MHz/512 = 117.1875kHz */
    CMT0.CMCOR = 58593U - 1U;
}

/*****
* Function Name: cmt_start
* Description  : Start counting CMT.
* Arguments    : none
* Return Value : none
*****/
void cmt_start(void)
{
    CMT.CMSTR0.BIT.STR0 = 1U;
    IEN(CMT0, CMI0) = 1U;
}

```

```
/* *****  
 * Function Name: cmt_cb_isr  
 * Description  : Callback for CMT interrupt.  
 * Arguments    : none  
 * Return Value : none  
 * *****/  
  
void cmt_cb_isr(void)  
{  
    led_output_inversion();  
}
```

inthandler.c

```
#include "interrupt_handlers.h"  
#include "cmt_define.h"  
  
// CMT0 CMI0  
void INT_Excep_CMT0_CMI0(void)  
{  
    cmt_cb_isr();  
}
```

移行元のプログラムでは CMT を使用し 2Hz 周期で LED を点滅します。

8.2.3 移行先のプロジェクトの作成

5 FSP 以前のプロジェクトから FSP プロジェクトに移行する手順で示した手順に則り、移行するプロジェクトを作成します。5.1 FSP プロジェクトの新規作成～5.4 モジュールの移行（FIT→FSP）を参考に新規に作成した FSP プロジェクトをベースにします。

8.2.4 割り込み設定の追加

移行元プロジェクトでは LED の周期点滅処理のために CMT 割り込みを使用していました。移行先プロジェクトでは 5.5.2 直接追加による割り込み設定を参考に[Interrupts Configuration]ページで CMT 割り込みを設定するように変更します。

[Interrupts Configuration]ページの[New User Event]から[CMT0]を選択します。

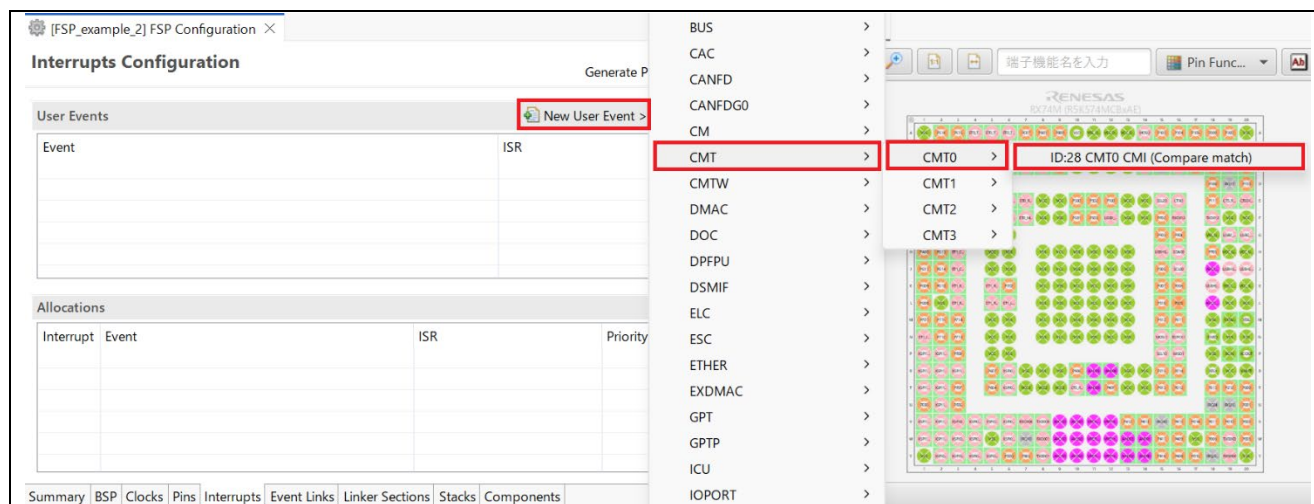


図 8-4 [Interrupts Configuration]ページ CMT0 の割り込み要因の選択

割り込み関数名を入力します。ここでは"cmt0_isr"としています。[OK]ボタンをクリックすると

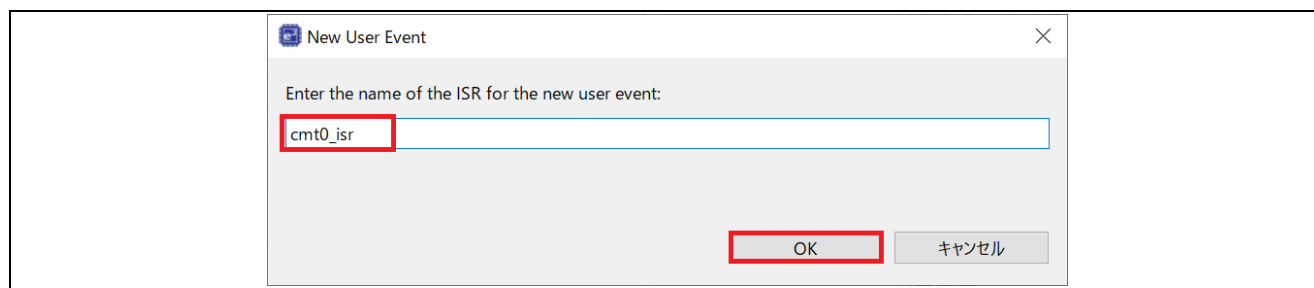


図 8-5 [Interrupts Configuration]ページ CMT0 の割り込み関数名入力

選択した CMT0 の割り込み情報が登録されます。

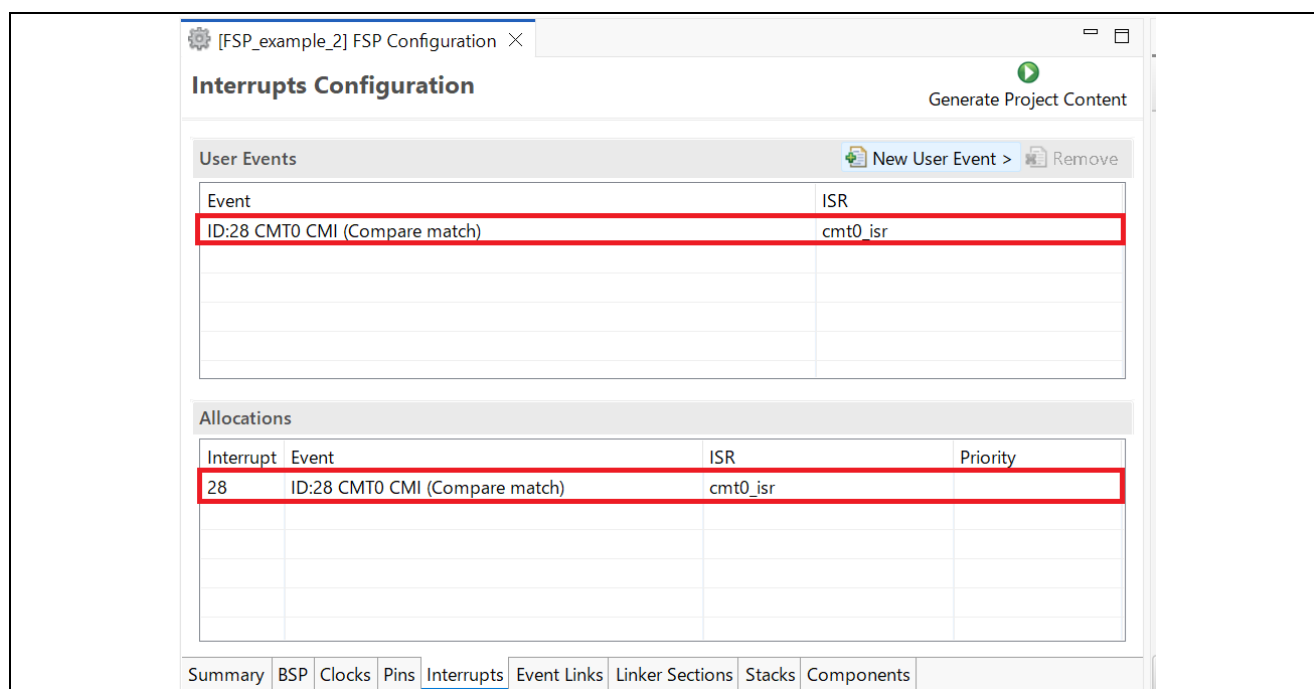


図 8-6 [Interrupts Configuration]ページ CMT0 割り込み追加

登録を確認後、[Generate Project Content]を実行してください。CMT0 の割り込み関数の宣言が生成されます。

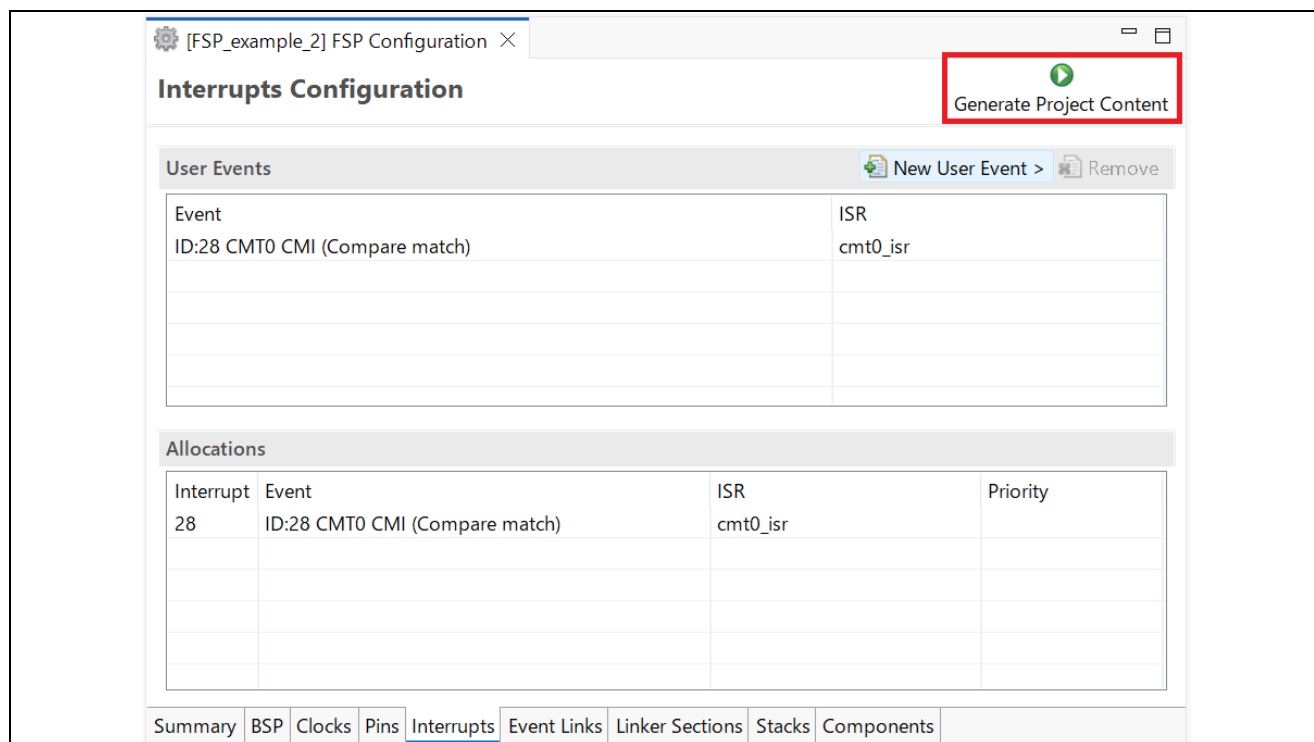


図 8-7 [Interrupts Configuration]ページ CMT0 割り込み生成

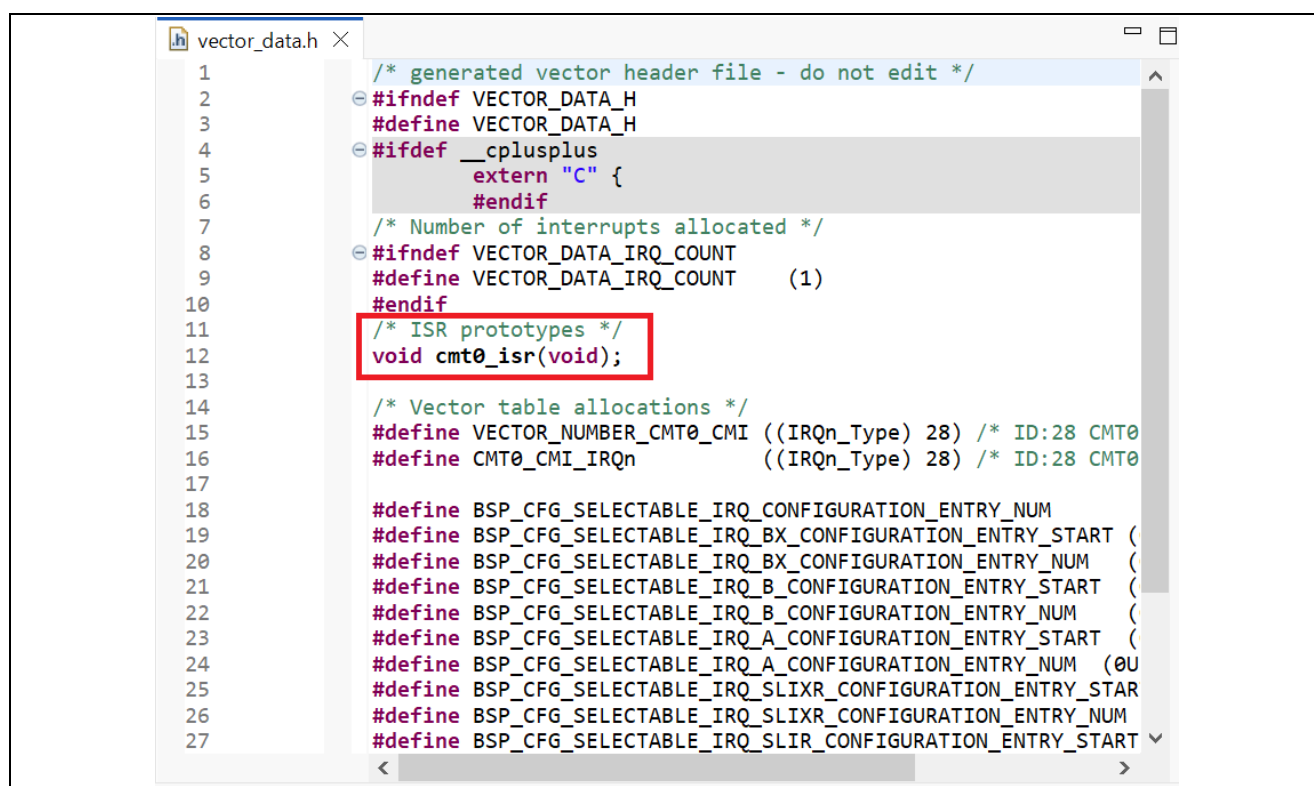


図 8-8 vector_data.h CMT0 割り込み関数の宣言

生成されるのは割り込み関数の宣言のみであるため、割り込み関数の実体は後述で定義を追加します。

8.2.5 移行後のプロジェクトのフォルダ・ファイル構成と変更対象

移行後のプロジェクトのフォルダ・ファイル構成と変更対象を以下に示します。hal_entry.c、led_control.c、cmt_control.c が変更対象です。

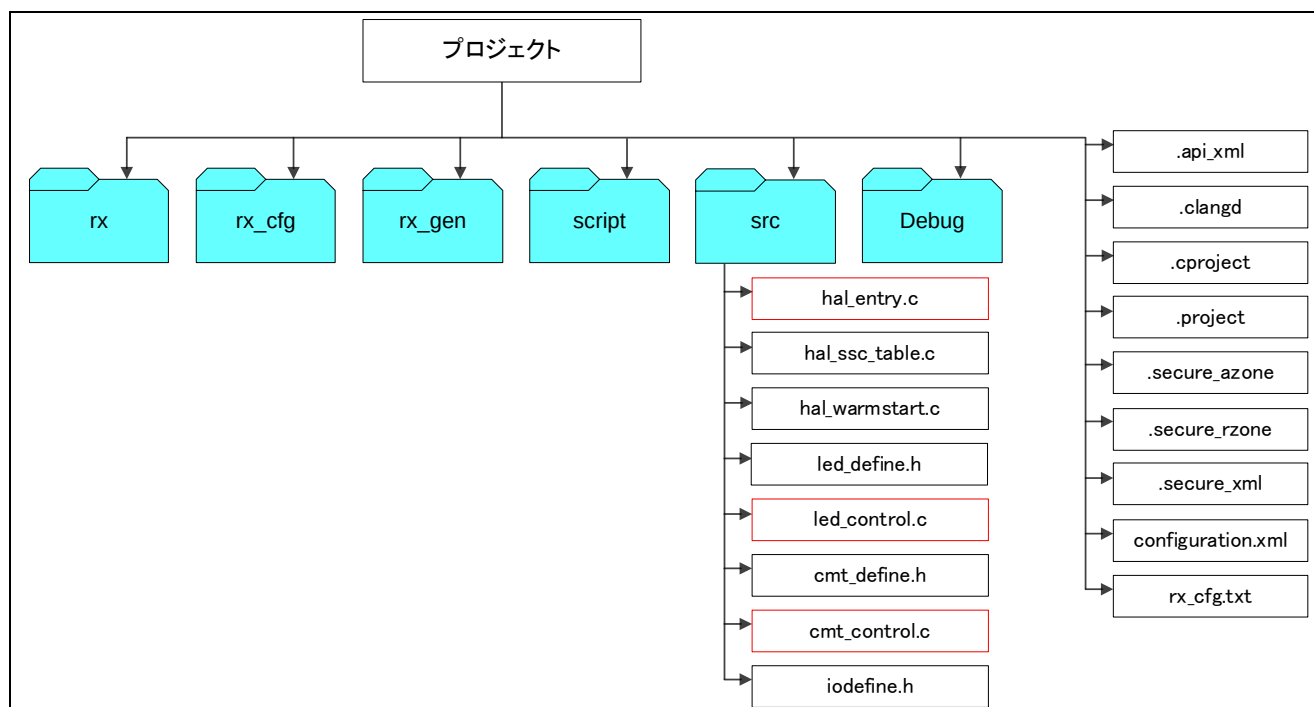


図 8-9 移行後プロジェクトのフォルダ・ファイル構成と変更対象

8.2.6 移行のポイント

5.7 ソースコードの移植に則り、main.c の main 関数本体の処理を hal_entry.c に移植します。[初期設定例のアプリケーションノート](#)が提供する main 関数内の処理ではクロックの初期設定を R_INIT_Clock()で行う他いくつか初期設定を行う関数が提供されますが、FSP プロジェクトでは main 関数に到達した時点でこれらのクロック設定を含む初期設定が完了しています。R_INIT_Clock()含むこれらの関数で再設定を行うと関連する設定が競合することになるため、初期設定例が提供する main 処理部分は移行しません。

CMT の割り込み関数を hal_entry.c に追加します。

led_define.h、led_control.c、cmt_define.h、cmt_control.c を移植します。led_control.c および cmt_control.c が iodef.h の定義を使用した定義を含むので 5.8 iodef.h の追加に則り iodef.h を追加します。デバイスが変わるため、移行前は PORT40 を使用していましたが、移行後は PORTF06 に変更しています。

5.9 FSP モジュールの処理への置き換えを参考に FIT モジュールの処理を FSP の処理へ置き換えます。

8.2.7 変更対象のファイルの置き換え後の処理

変更した最終的な hal_entry.c、led_control.c および cmt_control.c は以下のようになります。

hal_entry.c （赤字が最終的に追加した部分となります。）

```

#include "hal_data.h"
#include "led_define.h"
#include "cmt_define.h"

```

```

FSP_CPP_HEADER
FSP_CPP_FOOTER

/*****
 * main() is generated by the RX Configuration editor and is used to generate
 * threads if an RTOS is used. This function is called by main() when no
 * RTOS is used.
 *****/
void hal_entry(void)
{
    /* TODO: add your own code here */
    /* Initialize for application */
    led_port_init();
    cmt_initialize();
    cmt_start();

    while(1)
    {
        // TODO: add application code here
    }

#ifdef BSP_PZ_SECURE_BUILD
    /* Enter non-secure code */
    R_BSP_NonSecureEnter();
#endif
}

/*****
 * Function Name: cmt0_isr
 * Description   : Interrupt function for cmt0.
 * Arguments     : none
 * Return Value  : none
 *****/
void cmt0_isr(void)
{
    cmt_cb_isr();
}

```

led_control.c (赤字が変更した部分となります。)

```

/*****

```

```

Includes <System Includes> , "Project Includes"
*****/

#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Macro definitions
*****/

#define PORT_OUTPUT          (1U)
#define USER_LED            (PORTF.PODR.BIT.PODR06)

/*****
Exported global variables and functions (to be accessed by other files)
*****/

void led_port_init(void);
void led_output(uint8_t level);
void led_output_inversion(uint8_t level);

/*****
* Function Name: led_port_init
* Description   : Initialize the User LED port.
* Arguments    : none
* Return Value  : none
*****/
void led_port_init(void)
{
    /* initialize port */
    USER_LED = LED_OFF;
    PORTF.PDR.BIT.PDR06 = PORT_OUTPUT;
    PFS.PF06PFS.BIT.PMR = 0U;
}

/*****
* Function Name: led_output
* Description   : Turns the User LED on or off.
* Arguments    : level : output level
* Return Value  : none
*****/
void led_output(uint8_t level)

```

```

{
    USER_LED = level;
}

/*****
 * Function Name: led_output_inversion
 * Description  : Inverting the LED output.
 * Arguments    : none
 * Return Value : none
 *****/
void led_output_inversion(void)
{
    if(USER_LED == LED_OFF)
    {
        led_output(LED_ON);
    }
    else
    {
        led_output(LED_OFF);
    }
}

```

cmt_control.c (赤字が変更した部分となります。)

```

/*****
Includes  <System Includes> , "Project Includes"
 *****/

#include <stdint.h>
#include "iodefine.h"
#include "led_define.h"

/*****
Exported global variables and functions (to be accessed by other files)
 *****/

void cmt_initialize(void);
void cmt_start(void);
void cmt_cb_isr(void);

/*****
 * Function Name: cmt_initialize
 * Description  : Initialize the cmt.
 * Arguments    : none
 *****/

```

```

* Return Value : none
*****/
void cmt_initialize(void)
{
    /* initialize cmt */
    /*** 周辺モジュール設定 ***/
    SYSC_NS.PRCR_NS.WORD = 0xA502U;
    MSTPREG.MSTPCR.F.BIT.MSTP.F15 = 0U;
    SYSC_NS.PRCR_NS.WORD = 0xA500U;

    IR(CMT0, CMI0) = 0U;
    IPR(CMT0, CMI0) = 0x0FU;

    /* The CMT interrupt interval is set to 0.5 seconds(2 Hz). */
    CMT0.CMC.R.WORD = 0x0043U; /* PCLKB:60MHz/512 = 117.1875kHz */
    CMT0.CMCOR = 58593U - 1U;
}

/*****
* Function Name: cmt_start
* Description   : Start counting CMT.
* Arguments     : none
* Return Value  : none
*****/
void cmt_start(void)
{
    CMT.CMSTR0.BIT.STR0 = 1U;
    IEN(CMT0, CMI0) = 1U;
}

/*****
* Function Name: cmt_cb_isr
* Description   : Callback for CMT interrupt.
* Arguments     : none
* Return Value  : none
*****/
void cmt_cb_isr(void)
{
    led_output_inversion();
}

```

9. FSP プロジェクトへの移行時の注意点

本章では、既存の RX スマート・コンフィグレータや FIT ソフトウェアモジュールを使用したプロジェクト、またはこれらルネサス製ソフトウェアを使用しないプロジェクト（移行元プロジェクト）から FSP を使用したプロジェクト（移行先プロジェクト）に置き換える際の注意点について説明します。

9.1 BSP の組み込みと競合

FSP プロジェクトではプロジェクト生成時にプロジェクトの基盤となる BSP が組み込まれます。

BSP はリセット解除後から実行されるスタートアップ処理を含みます。FSP プロジェクトへの移行にあたり、スタートアップ処理や割り込み処理、端子設定等 FSP プロジェクトが提供する機能を使用する前提となります。移行元のプロジェクトからのスタートアップ処理や割り込み処理をそのまま用いると FSP プロジェクトの BSP の処理と競合が発生し、正常に動作しなくなりますのでご注意ください。

9.2 IO レジスタ定義ファイルの違いと競合

既存プロジェクトでは IO レジスタ定義は `iodef.h` のみで定義されていました。FSP プロジェクトにおける IO レジスタ定義ファイルは `renesas.h` から連なる複数のヘッダファイルで定義され構成されています。（図 2-32 bsp フォルダのファイル構成を参照。）

FSP プロジェクトでは基本的には IO レジスタにアクセスする処理は FSP の BSP が保持する `renesas.h` を参照することを推奨します。

`iodef.h` の IO レジスタ定義と `renesas.h` の IO レジスタ定義が異なるので IO レジスタにアクセスする記述例方法が以下のように変わります。

レジスタアクセス時の記述例（`iodef.h`）

```
CMT.CMSTR0.WORD = 0x0001;
WDT.WDTRR = 0x00;
```

レジスタアクセス時の記述例（`renesas.h`）

```
R_CMT->UNT->CMSTR0 = 0x0000;
R_WDT->WDTRR = 0x00;
```

既存プロジェクトの `iodef.h` を用いた IO レジスタアクセス処理を FSP プロジェクトへ移行する場合には、`iodef.h` を FSP プロジェクトに組み込む方法もあります。その場合、移行した FSP プロジェクトでは IO レジスタ定義ファイル（`iodef.h`）と FSP プロジェクトにおける IO レジスタ定義ファイル（`renesas.h`）の 2 つの定義ファイルを FSP プロジェクト内に保持することになります。

`iodef.h` におけるレジスタ定義と `renesas.h` におけるレジスタ定義は異なるため、プロジェクト内で同時に各ファイルの定義を用いた記述を行ったとしてもそれらが競合することはありません。以下に `iodef.h` から抜粋したいくつかの定義と `renesas.h` および関連するヘッダファイルから抜粋した定義を示します。

`iodef.h` の定義

```
(略)
#define CMT (*(volatile struct st_cmt __evenaccess *)0x88000)
#define CMT0 (*(volatile struct st_cmt0 __evenaccess *)0x88002)
```

(中略)

```
#define WDT (*(volatile struct st_wdt __evenaccess *)0x88020)
```

(中略)

```
typedef struct st_cmt {  
    union {  
        unsigned short WORD;  
        struct {  
            unsigned short :14;  
            unsigned short STR1:1;  
            unsigned short STR0:1;  
        } BIT;  
    } CMSTR0;  
    char wk0[14];  
    union {  
        unsigned short WORD;  
        struct {  
            unsigned short :14;  
            unsigned short STR3:1;  
            unsigned short STR2:1;  
        } BIT;  
    } CMSTR1;  
} st_cmt_t;
```

(中略)

```
typedef struct st_wdt {  
    unsigned char WDTRR;  
    char wk0[1];  
    union {  
        unsigned short WORD;  
        struct {  
            unsigned short :2;  
            unsigned short RPSS:2;  
            unsigned short :2;  
            unsigned short RPES:2;  
            unsigned short CKS:4;  
            unsigned short :2;  
            unsigned short TOPS:2;  
        } BIT;  
    } WDTCR;  
    union {  
        unsigned short WORD;
```

```

        struct {
            unsigned short REFEF:1;
            unsigned short UNDFE:1;
            unsigned short CNTVAL:14;
        } BIT;
    } WDTSR;
    union {
        unsigned char BYTE;
        struct {
            unsigned char RSTIRQS:1;
            unsigned char :7;
        } BIT;
    } WDTRCR;
} st_wdt_t;
(略)

```

renesas.h の定義

```

(略)
#if BSP_MCU_GROUP_RX74M
#include "R5K574M.h"
#elif BSP_MCU_GROUP_RX74N
#include "R5K574N.h"
#else
#if __has_include("renesas_internal.h")
#include "renesas_internal.h"
#else
#warning "Unsupported MCU"
#endif
#endif
(略)

```

R5F574M.h の定義

```

(略)
#include "R5F574M/r_cmt_reg.h"
(中略)
#include "R5F574M/r_wdt_reg.h"
(中略)
#define R_CMT_BASE (0x87826000UL + BASE_NS_OFFSET)
(中略)
#define R_WDT_BASE (0x87802600UL + BASE_NS_OFFSET)

```

(中略)

```
#define R_CMT                ((R_CMT_Type *) R_CMT_BASE)
```

(中略)

```
#define R_WDT                ((R_WDT_Type *) R_WDT_BASE)
```

(略)

r_cmt_reg.h の定義

(略)

```
typedef struct
```

{

```
    union
```

{

```
        __IOM uint16_t CMSTR0;          /*!< (@ 0x00000000) Compare Match Timer
                                         Start Register */
```

```
        struct
```

{

```
#if defined(__BIG) && defined(__GNUC__)
```

```
    uint16_t          : 14;
```

```
    __IOM uint16_t STR1 : 1; /*!< [1..1] Count Start 0 */
```

```
    __IOM uint16_t STR0 : 1; /*!< [0..0] Count Start 1 */
```

```
#else
```

```
    __IOM uint16_t STR0 : 1; /*!< [0..0] Count Start 1 */
```

```
    __IOM uint16_t STR1 : 1; /*!< [1..1] Count Start 0 */
```

```
    uint16_t          : 14;
```

```
#endif
```

```
    } CMSTR0_b;
```

};

```
    __IOM R_CMT_UNT_CM_Type CM[2];      /*!< (@ 0x00000002) 2 Timer Start Register
                                         Pairs */
```

```
    __IM uint16_t      RESERVED;
```

```
} R_CMT_UNT_Type;                      /*!< Size = 16 (0x10) */
```

(略)

r_wdt_reg.h の定義

(略)

```
typedef struct                                /*!< (@0x87802600) R_WDT Structure */
```

{

```
    union
```

{

```

__IOM uint8_t WDTRR; /*!< (@ 0x00000000) WDT Refresh Register */

struct
{
    __IOM uint8_t WDTRR : 8;
} WDTRR_b;
};
__IM uint8_t RESERVED;
(中略)
union
{
    __IOM uint8_t WDTRCR; /*!< (@ 0x00000006) WDT Reset Control */

    struct
    {
#if defined(__BIG) && defined(__GNUC__)
        __IOM uint8_t RSTIRQS : 1; /*!< [7..7] Reset Interrupt Request Select
        */
        uint8_t          : 7;
#else
        uint8_t          : 7;
        __IOM uint8_t RSTIRQS : 1; /*!< [7..7] Reset Interrupt Request Select
        */
#endif
    } WDTRCR_b;
};
__IM uint8_t RESERVED1;
__IM uint16_t RESERVED2;
} R_WDT_Type; /*!< Size = 10 (0xa) */
(略)

```

9.3 同じ周辺機能が搭載されていないデバイスでの移行

移行先の RX デバイスが同じ周辺機能を搭載していない場合は、デバイス・ドライバの置き換えができません。置き換えができない場合は、移行先の RX デバイス用のプログラムを新規に開発する必要があります。

10. 参考ドキュメント

- RX Flexible Software Package Documentation
- Firmware Integration Technology ユーザーズマニュアル(R01AN1833)
- RX ファミリ CMT モジュール Firmware Integration Technology(R01AN1856)
- RX ファミリ RSCI モジュール Firmware Integration Technology(R01AN5759)
- スマート・コンフィグレータ ユーザーズマニュアル RX API リファレンス編(R20UT4360)
- RX72N グループ 初期設定例 アプリケーションノート(R01AN4970)
- RX140 グループ Fast Prototyping Board for RX140 LED 点灯制御プログラムアプリケーションノート (R01AN7266)

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Oct.09.26	－	初版

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策

CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。

2. 電源投入時の処置

電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. 電源オフ時における入力信号

当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。

4. 未使用端子の処理

未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。

5. クロックについて

リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

6. 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、 V_{IL} (Max.) から V_{IH} (Min.) までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、 V_{IL} (Max.) から V_{IH} (Min.) までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。

7. リザーブアドレス（予約領域）のアクセス禁止

リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

8. 製品間の相違について

型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ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。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た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しします。
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。