Renesas e2 studio

R20AN0480JJ0100
Rev.1.00
2017.11.01

コード生成を使用したプロジェクトをスマート･コンフィグレータを使用したプロジェクトに移行する方法

要旨

本アプリケーションノートでは、コード生成を使用したプロジェクトを、スマート・コンフィグレータを使用したプロジェクトに移行する方法について説明します。

動作確認デバイス

- RX64M グループ

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

関連ドキュメント

Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド(R20AN0451)

e^2 studio 統合開発環境 ユーザーズマニュアル 入門ガイド(R20UT2858)

RSK+RX64M コード生成支援ツールチュートリアルマニュアル(R20UT2930)

RX64M Renesas Starter Kit Sample Code for CubeSuite+ (R01AN2219)

目次

1. 概要

1.1 本ドキュメントの目的

本アプリケーションノートでは、コード生成を使用したプロジェクトを、設定方法や、生成される関数名が異なるスマート・コンフィグレータを使用したプロジェクトに移行する方法について、実際にサンプルソースコードを用いて具体的に説明します。

e² studio の使い方については、「e² studio 統合開発環境 ユーザーズマニュアル 入門ガイド」を参照してください。

1.2 動作環境

表 1.1 動作環境

対象デバイス	RX64M グループ
エミュレータ	E1
IDE	e² studio v.6.0.0 以降
ツールチェーン	RX ファミリ用ルネサス C/C++コンパイラパッケージ
ツールチェーンバージョン	CC-RX V2.07.00

2.　コード生成を使用したプロジェクトをスマート・コンフィグレータを使用したプロジェクトに移行する方法

図 2.1 に、コード生成を使用したプロジェクトを、スマート・コンフィグレータを使用したプロジェクトに移行する手順を示します。

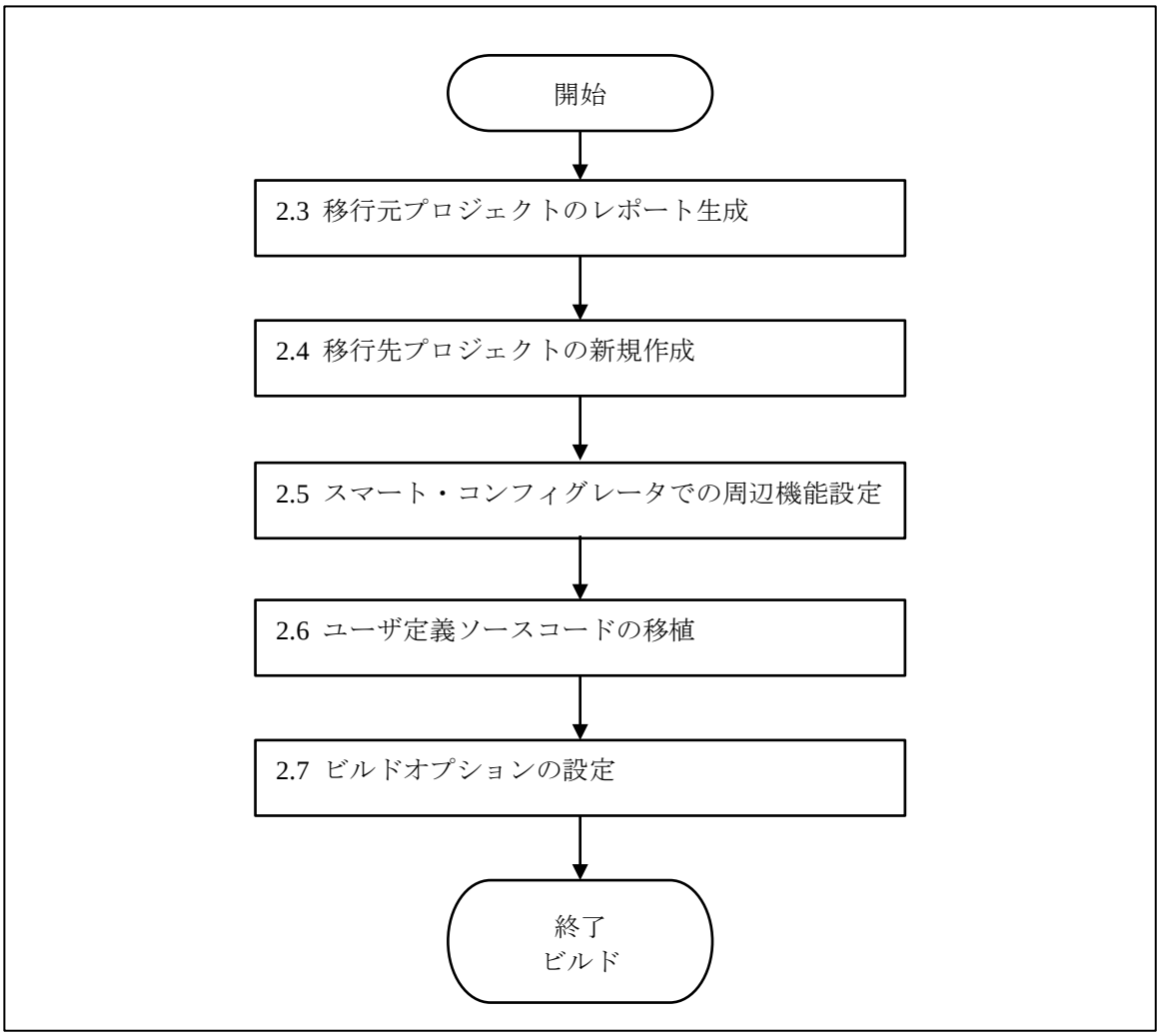


図 2.1　コード生成を使用したプロジェクトを
スマート・コンフィグレータを使用したプロジェクトに移行する手順

2.1 本アプリケーションノートで使用するプロジェクト

本アプリケーションノートでは、以下の 2 つのプロジェクトを使用します。

移行元プロジェクトとして、ルネサスマイクロコントローラ用の評価ツールである、RSK+RX64M のプロジェクトを使用します。移行先プロジェクトは新規作成します。

表 2.1 本アプリケーションノートで使用するプロジェクト

プロジェクト名	プロジェクトの概要
RSK+RX64M_Tutorial	移行元プロジェクトとして使用する、コード生成を使用した、RSK+RX64M のプロジェクト。周辺機能設定のレポート生成や、ユーザ作成ソースコードのコピー元として使用。
RSK_RX64M_Tutorial_SC	スマート･コンフィグレータを使用する設定で新規作成する移行先プロジェクト。移行元プロジェクトの周辺機能設定やユーザ作成ソースコードを、手順に従ってスマート・コンフィグレータ用に修正して反映する。

2.2　移行元プロジェクトのダウンロード

本アプリケーションノートで移行元プロジェクトとして使用する RSK+RX64M のプロジェクトは、ルネサスエレクトロニクスのホームページからダウンロードできます。
注：ダウンロードするためには、My Renesas への登録が必要です。

(1) ルネサスエレクトロニクスホームページのトップページ(https://www.renesas.com/ja-jp/)から、「製品情報」－「ボード&キット」を選択し、「Renesas Starter Kits」の「関連製品をもっと見る」を選択します。

図 2.2 移行元プロジェクトのダウンロード(1)

(2) 表示された Renesas Starter Kits の一覧から、「Renesas Starter Kit+ for RX64M」を選択します。

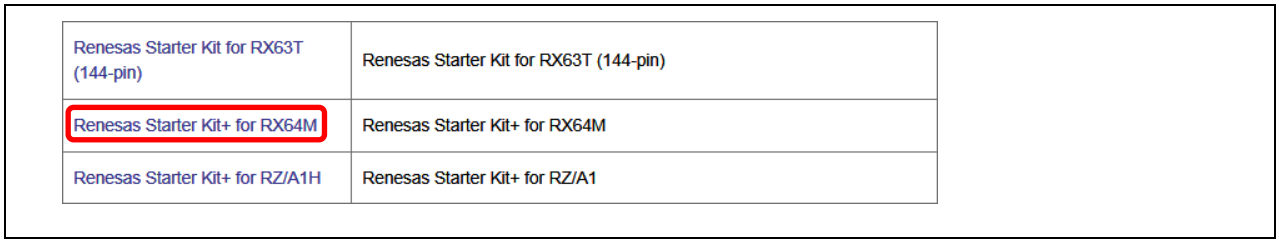


図 2.3 移行元プロジェクトのダウンロード(2)

(3)　「ダウンロード」タブの製品名称一覧から「RX64M Renesas Starter Kit Sample Code for CubeSuite+」を選択し、ダウンロードページ下部の「ダウンロード」ボタンを選択して、ダウンロードします。

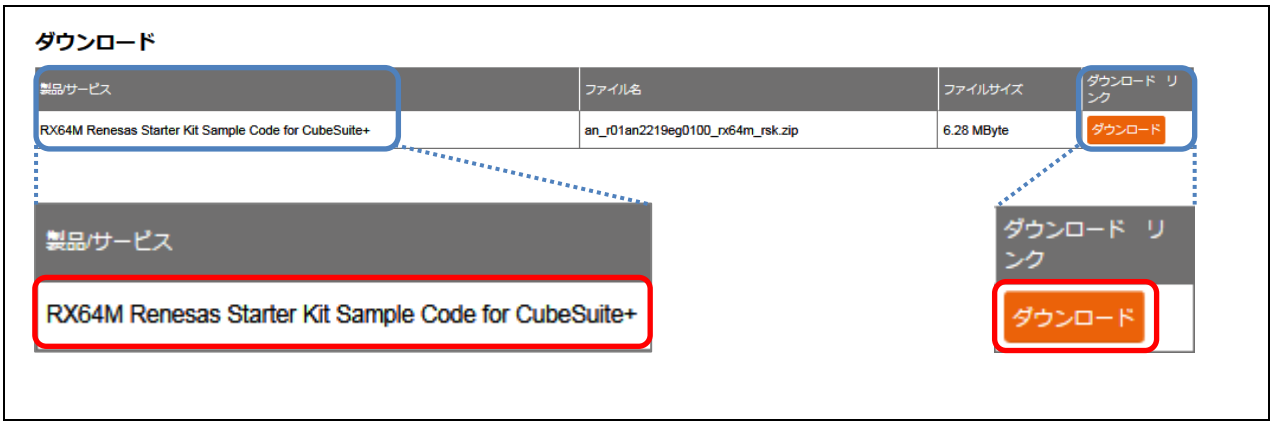


図 2.4 移行元プロジェクトのダウンロード(3)

2.3　移行元プロジェクトのレポート生成

移行元プロジェクトでコード生成のレポート生成機能を使い、周辺機能の設定一覧のレポートを出力します。このレポートを参照しながら、移行先プロジェクトでスマート・コンフィグレータの周辺機能設定を行います。

2.3.1　レポート生成の手順

【CS+の場合】

(1)　CS+ を起動し、コード生成を使用した移行元プロジェクト RSK+RX64M_Tutorial を開きます。プロジェクト・ツリーの[コード生成]を展開し、[周辺機能]をダブルクリックします。

(2)　[ファイル]メニューより、[コード生成レポートを保存(S)]を選択し、レポートを生成します。

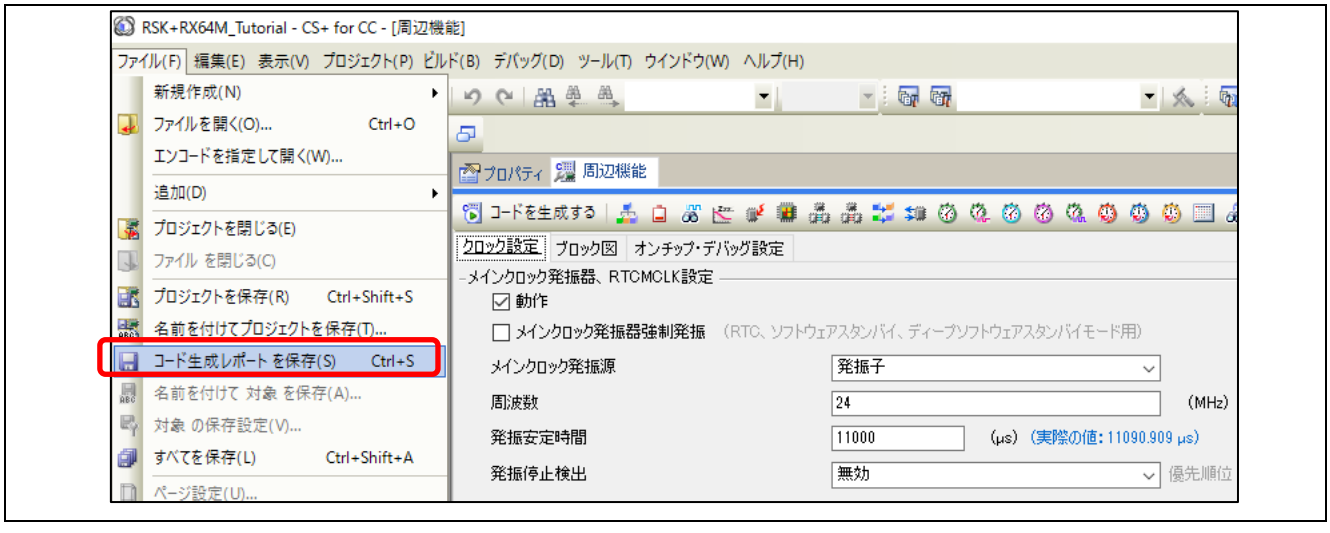


図 2.5 CS+のレポート生成の手順

(3)　レポートの生成が完了すると、プロジェクトフォルダ直下に、Function.html、Macro.html の 2 本のファイルが出力されます。

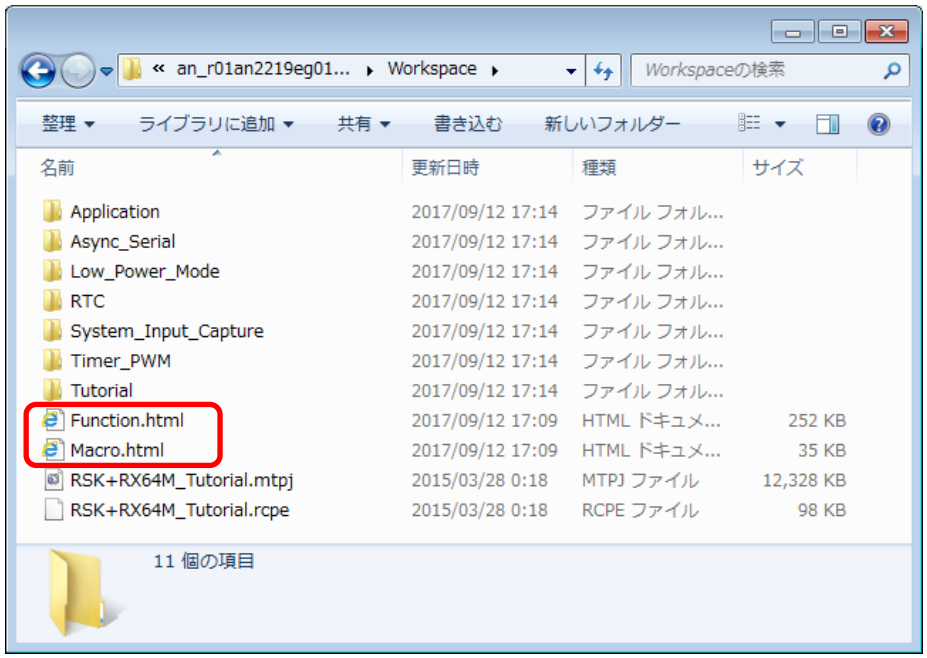


図 2.6　CS+のコード生成のレポート機能が出力するレポート

【e² studio の場合】

(1)　e² studio を起動し、コード生成を使用した移行元プロジェクト RSK+RX64M_Tutorial を開きます。プロジェクト・ツリーの[コード生成]を展開し、[周辺機能]をダブルクリックします。

(2)　[レポートを生成する]ボタンを押し、レポートを生成します。

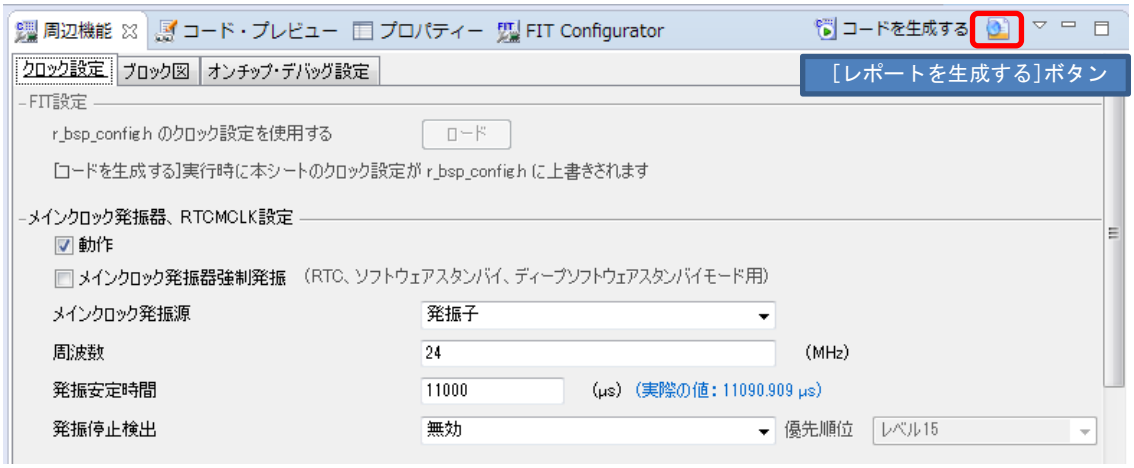


図 2.7 e² studio のレポート生成の手順

(3)　レポートの生成が完了すると、プロジェクトフォルダ直下の doc フォルダ直下に、Function.html、Macro.html の 2 本のファイルが出力されます。

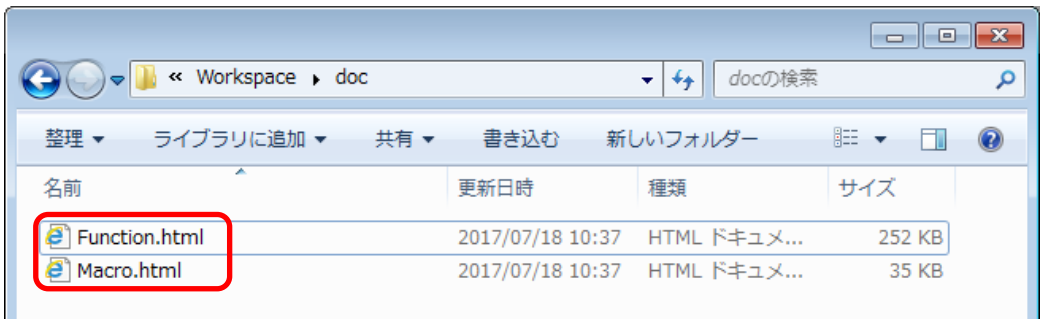


図 2.8　e^2 studio のコード生成のレポート機能が出力するレポート

表 2.2 コード生成のレポート機能が出力するレポート

ファイル名	概要
Function.html	コード生成が生成する API 関数の一覧
Macro.html	コード生成で設定した周辺機能の項目一覧

2.4 移行先プロジェクトの新規作成

移行先プロジェクトとなる、スマート・コンフィグレータを使用する C プロジェクトを新規作成します。プロジェクト作成方法については、「Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド」の「2 プロジェクトの作成」を参照してください。

2.5 スマート・コンフィグレータでの周辺機能設定

2.5.1 コード生成とスマート・コンフィグレータの周辺機能の対応

RSK+RX64M のプロジェクトで設定が必要な周辺機能について、コード生成とスマート・コンフィグレータの対応を表 2.3 に示します。

表 2.3 コード生成とスマート・コンフィグレータの周辺機能の対応(1)

コード生成			スマート・コンフィグレータ			
周辺機能	設定項目		タブ	周辺機能	設定項目	
割り込み機能	IRQ2 設定	―	コンポーネント	割り込みコントローラ	IRQ2 設定	―
			端子	端子機能	ポート	IRQ2
	IRQ5 設定	―	コンポーネント	割り込みコントローラ	IRQ5 設定	―
			端子	端子機能	ポート	IRQ5
	グループ BL0 設定	―	割り込み	GROUPBL0	―	―
コンペアマッチタイマ	CMT0	―	コンポーネント	コンペアマッチタイマ	CMT0	―
	CMT1	―			CMT1	―
	CMT2	―			CMT2	―
12 ビット A/D コンバータ	シングルスキャンモード	アナログ入力チャネル設定	コンポーネント	シングルスキャンモード S12AD	基本設定	アナログ入力チャネル設定
		変換開始トリガ設定				変換開始トリガ設定
		ADTRGn#端子選択	端子	端子機能	12 ビット A/D コンバータ	―

表 2.4 コード生成とスマート・コンフィグレータの周辺機能の対応(2)

コード生成			スマート・コンフィグレータ			
周辺機能	設定項目		タブ	周辺機能	設定項目	
シリアルコミュニケーションインタフェース	簡易 SPI モード	マスタ送信	コンポーネント	SPI クロック同期式モード	ー	マスタ送信
		データ転送方向設定			ー	データ転送方向設定
		転送速度設定			ー	転送速度設定
		端子設定	端子	端子機能	シリアルコミュニケーションインタフェース	ー
	調歩同期式	送信/受信	コンポーネント	SCI(SCIF)調歩同期式モード	ー	送信/受信
		スタートビット検出設定			ー	スタートビット検出設定
		転送速度設定			ー	転送速度設定
ポート設定	Port0	P03	コンポーネント	ポート	PORT0	P03
		P05				P05
	Port2	P26			PORT2	P26
		P27				P27
	Port4	P45			PORT4	P45
		P46				P46
		P47				P47

「2.3 移行元プロジェクトのレポート生成」で出力したレポートを参照しながら、「2.4 移行先プロジェクトの新規作成」で作成したプロジェクトで、スマート・コンフィグレータの設定を行います。

ここでは、クロック発生回路、コンペアマッチタイマ、シリアルコミュニケーションインタフェースの設定について説明します。その他の周辺機能については、同様の手順で設定を行ってください。

2.5.2　クロック発生回路の設定

クロック発生回路の設定を行います。

(1)　「2.3 移行元プロジェクトのレポート生成」で出力したレポートの Macro.html ファイルを開き、「クロック発生回路」の設定箇所を表示します。

MCU name: RX64M_4MB(4MB)
Chip name: R5F564MLCxFC

Peripheral function	Macro	SubMacro	Setting	Status
クロック発生回路				使用する
	CGC			使用する
			メインクロック発振器強制発振	使用しない
			メインクロック発振源	発振子
			メインクロック発振源 周波数	24(MHz)
			発振安定時間	11000(μs),(実際の値：11090.909 μs)
			発振停止検出	無効
			PLL 動作	使用する
			PLLクロックソース選択	メインクロック発振器
			入力分周比	x 1
			周波数逓倍率	x 10.0
			周波数	240 (MHz)
			SubCLK 動作	使用しない

図 2.9　クロック発生回路のコード生成レポート

(2)　「2.4 移行先プロジェクトの新規作成」で作成したプロジェクトで、スマート･コンフィグレータの設定画面を開き、「クロック」タブを選択します。

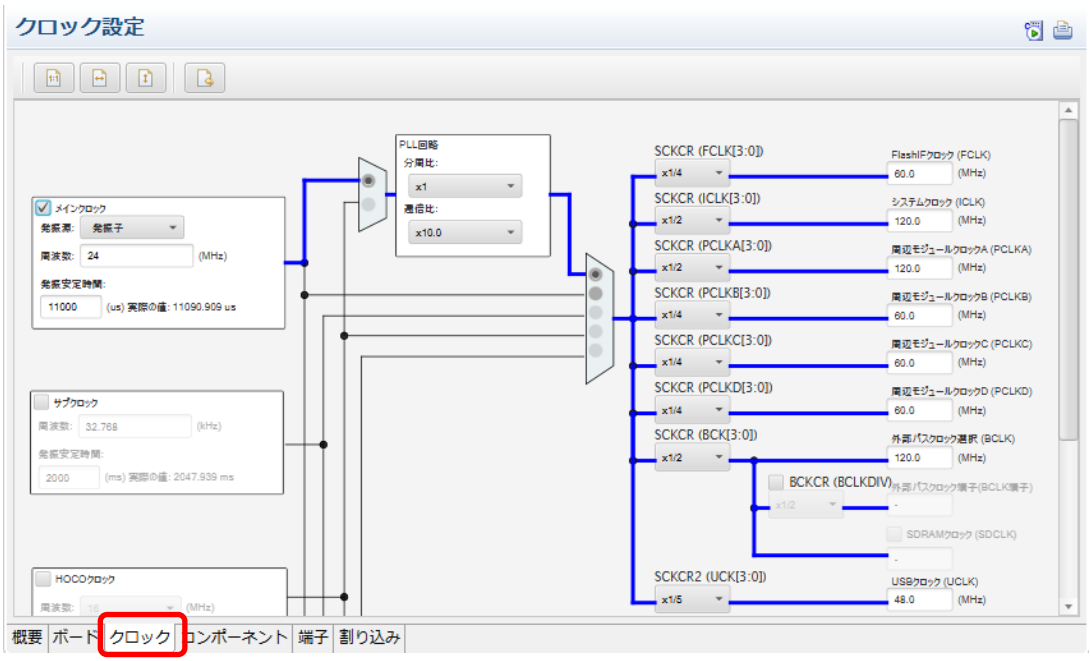


図 2.10　スマート・コンフィグレータのクロック設定画面

(3)　レポートの Macro.html の「Setting」の列の設定項目と「Status」の列の設定内容を、スマート・コンフィグレータの設定に反映していきます。

Setting	Status
	使用する
	使用する
メインクロック発振器強制発振	使用しない
メインクロック発振源	発振子 (1)
メインクロック発振源 周波数	24(MHz) (2)
発振安定時間	11000(μs),(実際の値: 11090.909 μs) (3)
発振停止検出	無効
PLL 動作	使用する
PLLクロックソース選択	メインクロック発振器 (4)
入力分周比	x 1 (5)
周波数逓倍率	x 10.0 (6)
周波数	240 (MHz)
SubCLK 動作	使用しない

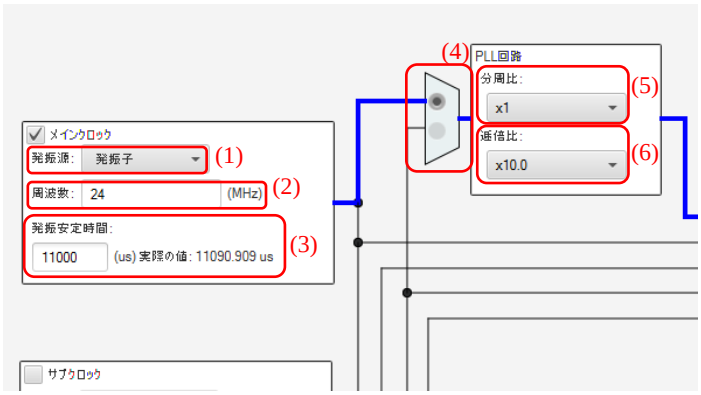


図 2.11　スマート・コンフィグレータのクロック設定(1)

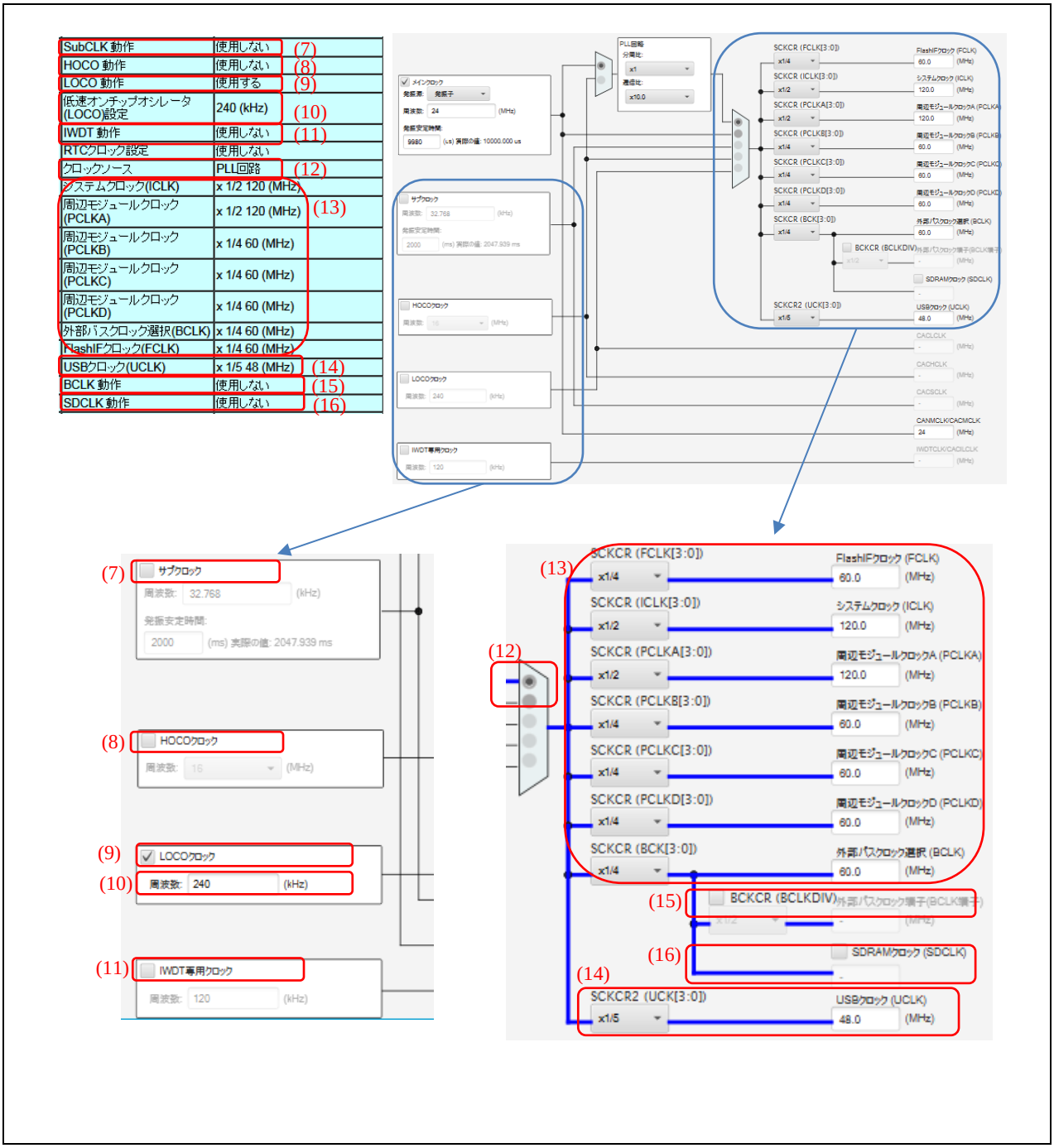


図 2.12　スマート・コンフィグレータのクロック設定(2)

表 2.5 クロック発生回路の設定(1)

	コード生成		スマート・コンフィグレータ	
	設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
(1)	メインクロック発振源	発振子	[メインクロック] 発振源	発振子
(2)	メインクロック発振源 周波数	24(MHz)	[メインクロック] 周波数	24(MHz)
(3)	発振安定時間	11000(us) (実際の値：11090.909us)	[メインクロック] 発振安定時間	11000(us) (実際の値：11090.909us)
(4)	PLL クロックソース選択	メインクロック発振器	[メインクロック]が PLL クロックソースとして選択されていることを確認	
(5)	入力分周比	× 1	[PLL 回路] 分周比	× 1
(6)	周波数逓倍率	× 10.0	[PLL 回路] 逓倍比	× 10.0
(7)	SubCLK 動作	使用しない	サブクロック	チェックなし
(8)	HOCO 動作	使用しない	HOCO クロック	チェックなし
(9)	LOCO 動作	使用する	LOCO クロック	チェックあり
(10)	低速オンチップオシレータ(LOCO)設定	240 (kHz)	周波数	240 (kHz)
(11)	IWDT 動作	使用しない	IWDT 専用クロック	チェックなし
(12)	クロックソース	PLL 回路	[PLL 回路]がクロックソースとして選択されていることを確認	

表 2.6 クロック発生回路の設定(2)

	コード生成		スマート・コンフィグレータ	
	設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
(13)	システムクロック (ICLK)	× 1/2 120 (MHz)	SCKCR (ICLK[3:0])	×1/2
			システムクロック (ICLK)	120.0 (MHz)
	周辺モジュールクロック (PCLKA)	× 1/2 120 (MHz)	SCKCR (PCLKA[3:0])	×1/2
			周辺モジュールクロック A (PCLKA)	120.0 (MHz)
	周辺モジュールクロック (PCLKB)	× 1/4 60 (MHz)	SCKCR (PCLKB[3:0])	×1/4
			周辺モジュールクロック B (PCLKB)	60.0 (MHz)
	周辺モジュールクロック (PCLKC)	× 1/4 60 (MHz)	SCKCR (PCLKC[3:0])	×1/4
			周辺モジュールクロック C (PCLKC)	60.0 (MHz)
	周辺モジュールクロック (PCLKD)	× 1/4 60 (MHz)	SCKCR (PCLKD[3:0])	×1/4
			周辺モジュールクロック D (PCLKD)	60.0 (MHz)
	外部バスクロック選択 (BCLK)	× 1/4 60 (MHz)	SCKCR (BCK[3:0])	×1/4
			外部バスクロック選択 (BCLK)	60.0 (MHz)
	FlashIF クロック (FCLK)	× 1/4 60 (MHz)	SCKCR (FCLK[3:0])	×1/4
			FlashIF クロック (FCLK)	60.0 (MHz)
(14)	USB クロック (UCLK)	× 1/5 48 (MHz)	SCKCR2 (UCK[3:0])	×1/5
			USB クロック (UCLK)	48.0 (MHz)
(15)	BCLK 動作	使用しない	BCKCR (BCLKDIV)	チェックなし
(16)	SDCLK 動作	使用しない	SDRAM クロック (SDCLK)	チェックなし

2.5.3 コンペアマッチタイマの設定

コンペアマッチタイマの設定を行います。

(1) 「Renesas e² studio スマート・コンフィグレータ ユーザーガイド」の「3.3.1 プロジェクトへのソフトウェアコンポーネント追加」－「(1) コード生成コンポーネント追加方法」を参照し、コンペアマッチタイマのコンポーネントをプロジェクトに追加します。
このとき、[選択したコンポーネントのコンフィグレーションを追加します]ダイアログボックスで、以下の通り、リソースに対するコンフィグレーション名にデフォルト名を使用してください。

表 2.7 コンペアマッチタイマのリソースとコンフィグレーション名の対応

コンポーネントのタイプ	コンポーネント	リソース	コンフィグレーション名
コード生成	コンペアマッチタイマ	CMT0	Config_CMT0 (デフォルト)
		CMT1	Config_CMT1 (デフォルト)
		CMT2	Config_CMT2 (デフォルト)

(2) 「2.3 移行元プロジェクトのレポート生成」で出力したレポートの Macro.html ファイルで、「コンペアマッチタイマ」の設定箇所を表示します。

コンペアマッチタイマ				使用する
	CMT0			使用する
			コンペアマッチタイマ動作設定	使用する
			クロック設定	PCLK/8
			インターバル時間設定	1ms,(実際の値:1)
			コンペアマッチ割り込みを許可(CMI0)	使用する
			優先順位	レベル10
	CMT1			使用する
			コンペアマッチタイマ動作設定	使用する
			クロック設定	PCLK/32
			インターバル時間設定	20ms,(実際の値:20)
			コンペアマッチ割り込みを許可(CMI1)	使用する
			優先順位	レベル10
	CMT2			使用する
			コンペアマッチタイマ動作設定	使用する
			クロック設定	PCLK/512
			インターバル時間設定	200ms,(実際の値:200.004267)
			コンペアマッチ割り込みを許可(CMI2)	使用する
			優先順位	レベル10

図 2.13 コンペアマッチタイマのコード生成レポート

(3)　(1)で作成したコンペアマッチタイマ CMT0 の設定画面を開きます。

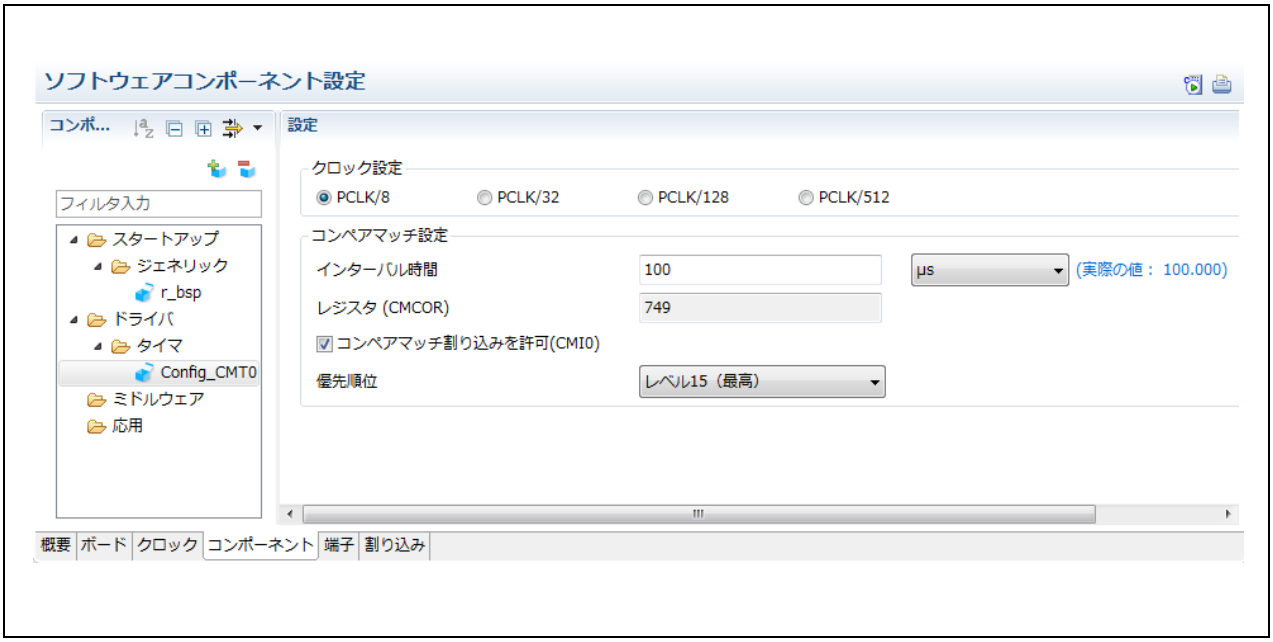


図 2.14　スマート・コンフィグレータのコンペアマッチタイマ(CMT0)の設定画面

(4)　Macro.html のコンペアマッチタイマの設定内容を、スマート・コンフィグレータの CMT0 の設定に反映していきます。

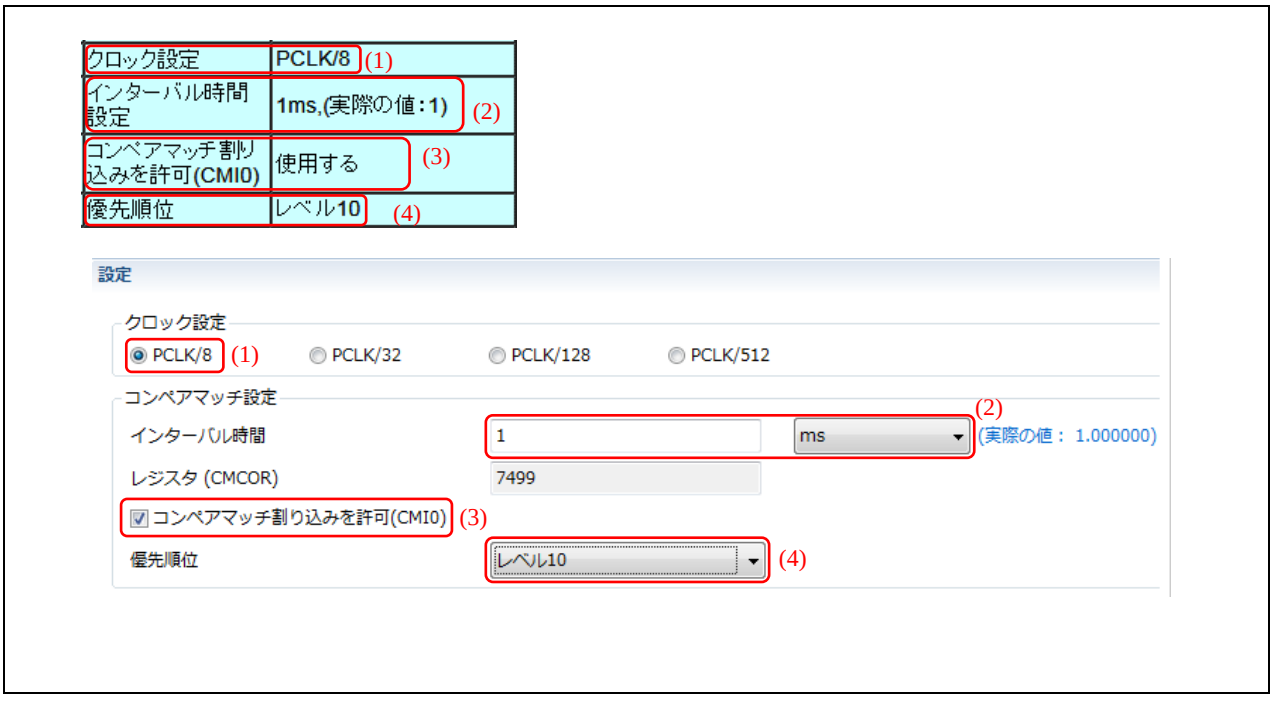


図 2.15　スマート・コンフィグレータのコンペアマッチタイマ(CMT0)の設定

表 2.8 コンペアマッチタイマ(CMT0)の設定

	コード生成		スマート・コンフィグレータ	
	設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
(1)	クロック設定	PCLK/8	クロック設定	PCLK/8
(2)	インターバル時間設定	1 ms (実際の値：1)	[コンペアマッチ設定] インターバル時間	1 ms (実際の値：1.000000)
(3)	コンペアマッチ割り込みを許可(CMI0)	使用する	[コンペアマッチ設定] コンペアマッチ割り込みを許可(CMI0)	チェックあり
(4)	優先順位	レベル 10	[コンペアマッチ設定] 優先順位	レベル 10

(5)　同様に、CMT1、CMT2 についてもコンポーネントを追加し、設定を行います。

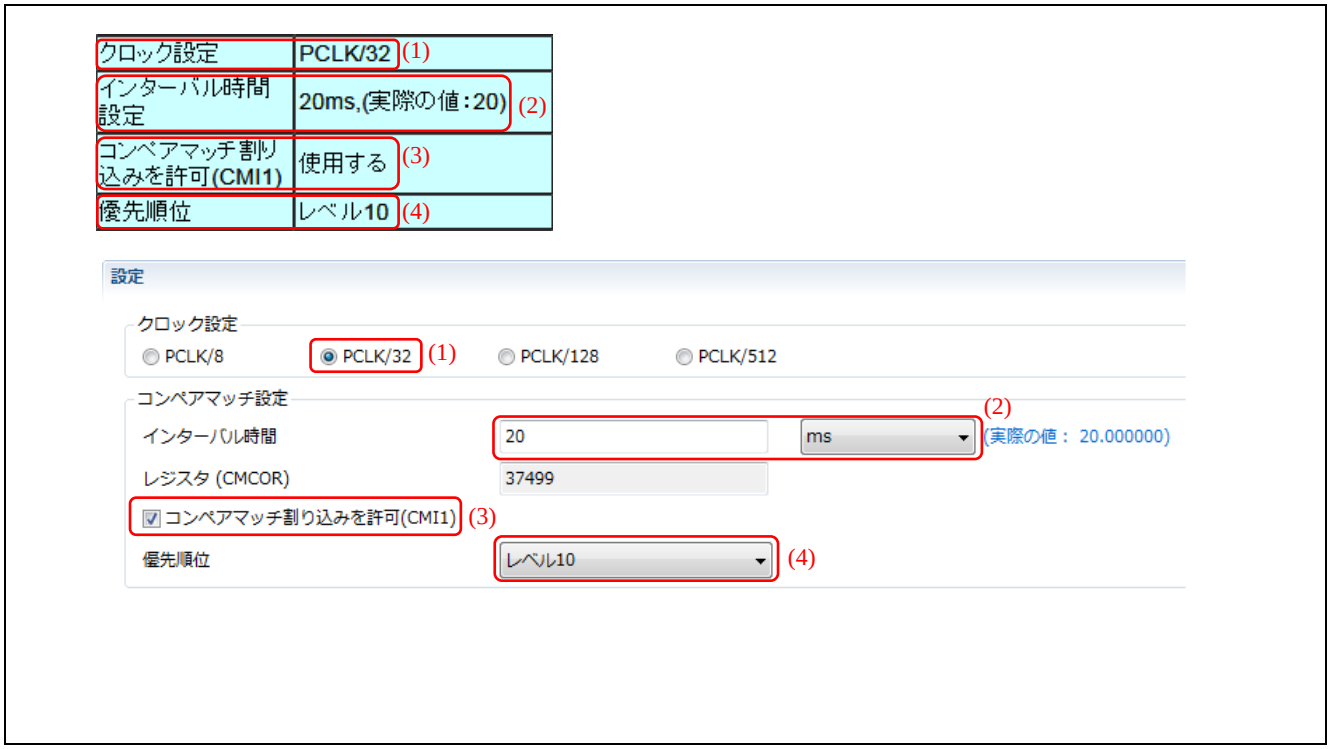


図 2.16　スマート・コンフィグレータのコンペアマッチタイマ(CMT1)の設定

表 2.9 コンペアマッチタイマ(CMT1)の設定

	コード生成		スマート・コンフィグレータ	
	設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
(1)	クロック設定	PCLK/32	クロック設定	PCLK/32
(2)	インターバル時間設定	20 ms (実際の値：20)	[コンペアマッチ設定] インターバル時間	20 ms (実際の値：20.000000)
(3)	コンペアマッチ割り込みを許可(CMI1)	使用する	[コンペアマッチ設定] コンペアマッチ割り込みを許可(CMI1)	チェックあり
(4)	優先順位	レベル 10	[コンペアマッチ設定] 優先順位	レベル 10

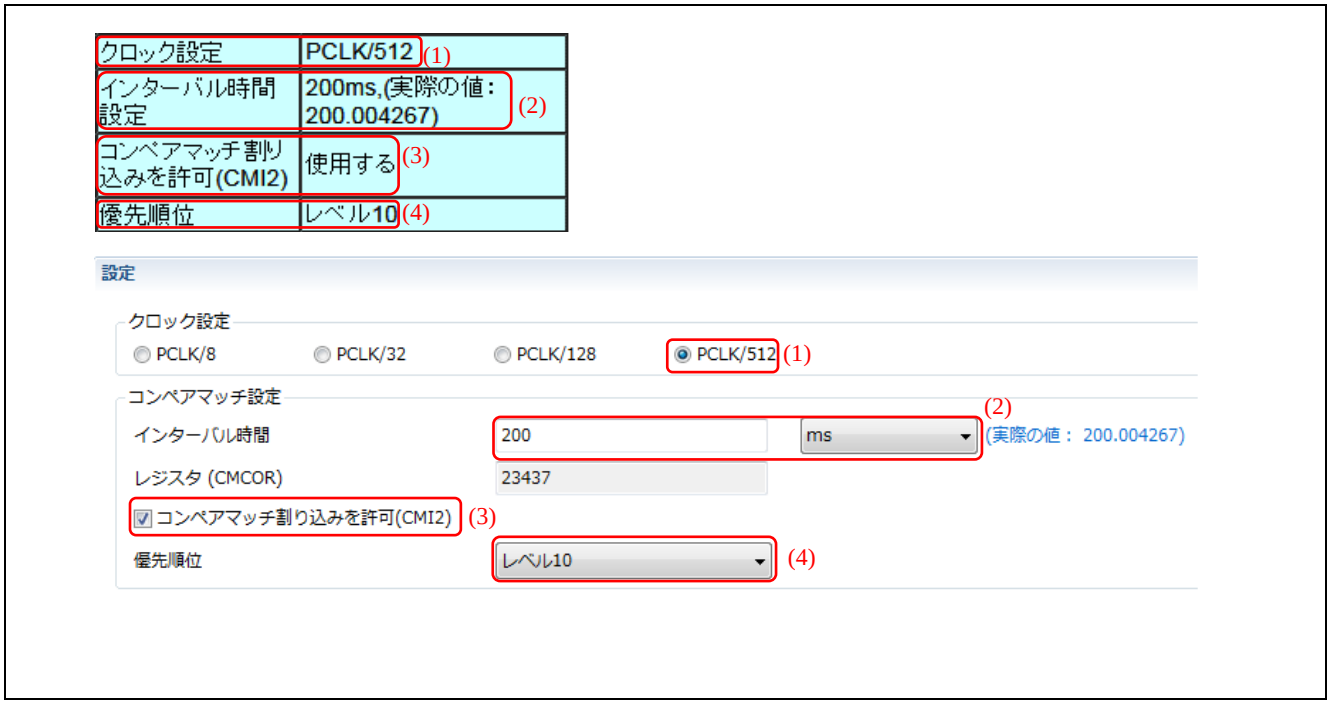


図 2.17　スマート・コンフィグレータのコンペアマッチタイマ(CMT2)の設定

表 2.10 コンペアマッチタイマ(CMT2)の設定

	コード生成		スマート・コンフィグレータ	
	設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
(1)	クロック設定	PCLK/512	クロック設定	PCLK/512
(2)	インターバル時間設定	200 ms (実際の値：200.004267)	[コンペアマッチ設定] インターバル時間	200 ms (実際の値：20.004267)
(3)	コンペアマッチ割り込みを許可(CMI2)	使用する	[コンペアマッチ設定] コンペアマッチ割り込みを許可(CMI2)	チェックあり
(4)	優先順位	レベル 10	[コンペアマッチ設定] 優先順位	レベル 10

2.5.4 シリアルコミュニケーションインタフェースの設定

シリアルコミュニケーションインタフェースの設定を行います。

(1) 「Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド」の「3.3.1 プロジェクトへのソフトウェアコンポーネント追加」－「(1) コード生成コンポーネント追加方法」を参照し、シリアルコミュニケーションインタフェースのコンポーネントをプロジェクトに追加します。
SCI6 を簡易 SPI モード、SCI7 を SCI 調歩同期式モードとして使用しますので、コンポーネント、リソース、動作/作業モードを以下の通り設定し、デフォルトのコンフィグレーション名を使用してください。

表 2.11 シリアルコミュニケーションインタフェースのリソースとコンフィグレーション名の対応

コンポーネントのタイプ	コンポーネント	リソース	コンフィグレーション名	動作/作業モード
コード生成	SPI クロック同期式モード	SCI6	Config_SCI6 (デフォルト)	マスタ送信機能
	SCI(SCIF)調歩同期式モード	SCI7	Config_SCI7 (デフォルト)	送信/受信

(2)　「2.3 移行元プロジェクトのレポート生成」で出力したレポートの Macro.html ファイルで、「シリアルコミュニケーションインタフェース」の設定箇所を表示します。

シリアルコミュニケーションインタフェース				使用する
	SCI6			使用する
			機能設定	簡易SPIバス (マスタ送信)
			SMOSI6	P00
		SimpleSPIMode_Master_Transmit6		使用する
			データ転送方向設定	MSBファースト
			受信データ･レベル設定	標準
			転送クロック	内部クロック
			ビットレート	1500000 (bps)
			ビットレートモジュレーション機能有効	使用しない
			SCK6端子機能	クロック出力
			SCK6	P02
			クロック遅延	クロック遅れなし
			クロック極性反転あり	使用しない
			送信データ処理	割り込みサービスルーチンで処理する
			TXI6優先順位	レベル15
			TEI6, ERI6 優先順位 (グループBL0)	レベル15
			送信完了	使用する
	SCI7			使用する
			機能設定	調歩同期式 (送信/受信)
			TXD7	P90
			RXD7	P92
		AsynchronousMode_TransmitReceive7		使用する
			スタートビット検出設定	RXD7端子の立ち下がりエッジ
			データ･ビット長設定	8ビット
			パリティ設定	パリティなし
			ストップビット設定	1ビット
			データ転送方向設定	LSBファースト
			転送クロック	内部クロック
			ビットレート	19200 (bps)
			ビットレートモジュレーション機能有効	使用する
			SCK7端子機能	SCK7を使用しない
			ノイズ除去機能を使用する	使用しない
			ハードウェアフロー制御設定	禁止
			送信データ処理	割り込みサービスルーチンで処理する
			受信データ処理	割り込みサービスルーチンで処理する
			エラー割り込み許可(ERI7)	使用する
			TXI7優先順位	レベル15
			RXI7優先順位	レベル15
			TEI7,ERI7優先順位 (グループBL0)	レベル15
			送信完了	使用する
			受信完了	使用する
			受信エラー	使用する

図 2.18　シリアルコミュニケーションインタフェースのコード生成レポート

(3)　(1)で作成したシリアルコミュニケーションインタフェース SCI6 の設定画面を開きます。

図 2.19 スマート・コンフィグレータのシリアルコミュニケーションインタフェース(SCI6)の設定画面

(4)　Macro.html のシリアルコミュニケーションインタフェースの設定内容を、スマート・コンフィグレータの SCI6 の設定に反映していきます。
SMOSI6、SCK6 の端子設定は、「端子」タブで行います。

	機能設定	簡易SPIバス (マスタ送信)
	SMOSI6	P00
SimpleSPIMode_Master_Transmit6		使用する
	データ転送方向設定	MSBファースト (1)
	受信データ･レベル設定	標準 (2)
	転送クロック	内部クロック (3)
	ビットレート	1500000 (bps) (4)
	ビットレートモジュレーション機能有効	使用しない (5)
	SCK6端子機能	クロック出力
	SCK6	P02
	クロック遅延	クロック遅れなし (6)
	クロック極性反転あり	使用しない (7)
	送信データ処理	割り込みサービスルーチンで処理する (8)
	TXI6優先順位	レベル15 (9)
	TEI6, ERI6 優先順位 (グループBL0)	レベル15 (10)
	送信完了	使用する (11)

図 2.20 スマート・コンフィグレータのシリアルコミュニケーションインタフェース(SCI6)の設定

表 2.12 シリアルコミュニケーションインタフェース(SCI6)の設定

	コード生成		スマート・コンフィグレータ	
	設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
(1)	データ転送方向設定	MSB ファースト	データ転送方向設定	MSB ファースト
(2)	受信データ・レベル設定	標準	受信データ・レベル設定	標準
(3)	転送クロック	内部クロック	[転送速度設定] 転送クロック	内部クロック (SCK6 端子はクロック出力端子となります)
(4)	ビットレート	1500000 (bps)	[転送速度設定] ビットレート	1500 (kbps)
(5)	ビットレートモジュレーション機能有効	使用しない	[転送速度設定] ビットレートモジュレーション機能有効	チェックなし
(6)	クロック遅延	クロック遅れなし	[クロック設定] クロック遅れあり	チェックなし
(7)	クロック極性反転あり	使用しない	[クロック設定] クロック極性反転あり	チェックなし
(8)	送信データ処理	割り込みサービスルーチンで処理する	[データ処理設定] 送信データ処理	割り込みサービスルーチンで処理する
(9)	TXI6 優先順位	レベル 15	[割り込み設定] TXI6 優先順位	レベル 15 (最高)
(10)	TEI6, ERI6 優先順位 (グループ BL0)	レベル 15	[割り込み設定] TEI6 優先順位 (グループ BL0)	レベル 15 (最高)
(11)	送信完了	使用する	[コールバック機能設定] 送信完了	チェックあり

(5)　(1)で作成したシリアルコミュニケーションインタフェース SCI7 の設定画面を開きます。

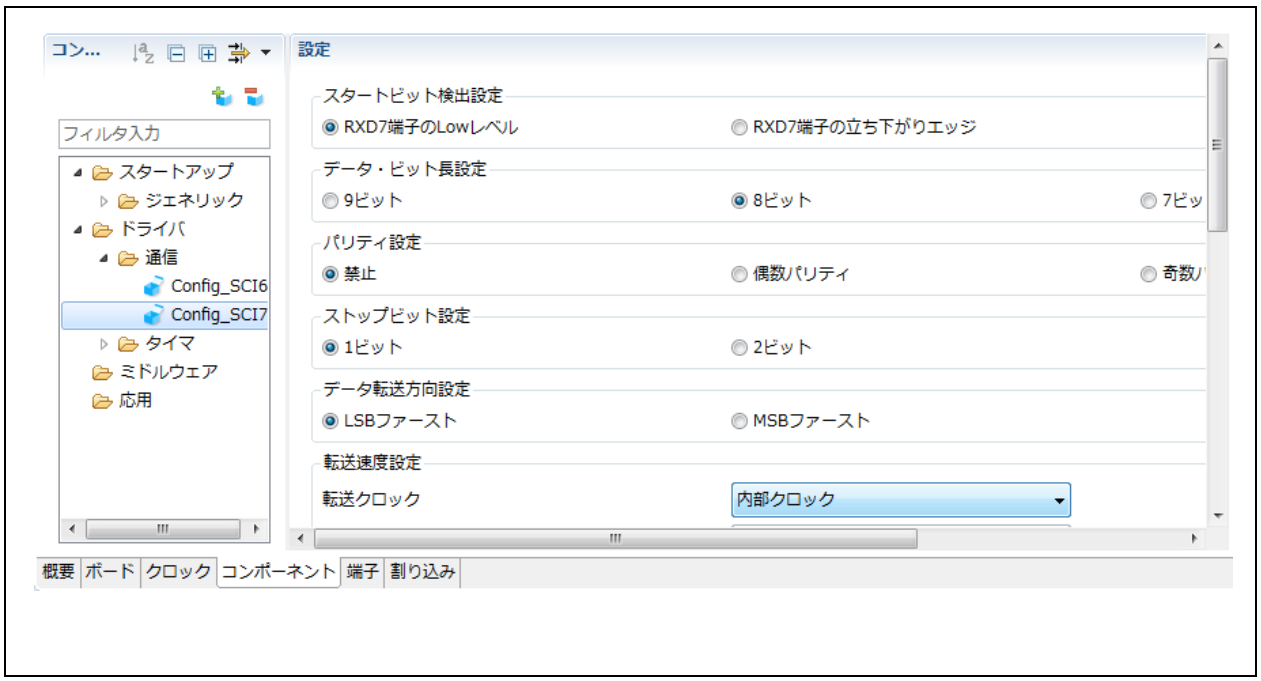


図 2.21 スマート・コンフィグレータのシリアルコミュニケーションインタフェース(SCI7)の設定画面

(6)　Macro.html のシリアルコミュニケーションインタフェースの設定内容を、スマート・コンフィグレータの SCI7 の設定に反映していきます。TXD7、RXD7 の端子設定は、「端子」タブで行います。

	TXD7	P90
	RXD7	P92
AsynchronousMode_TransmitReceive7		使用する
	スタートビット検出設定	RXD7端子の立ち下がりエッジ (1)
	データ･ビット長設定	8ビット (2)
	パリティ設定	パリティなし (3)
	ストップビット設定	1ビット (4)
	データ転送方向設定	LSBファースト (5)
	転送クロック	内部クロック (6)
	ビットレート	19200 (bps) (7)
	ビットレートモジュレーション機能有効	使用する (8)
	SCK7端子機能	SCK7を使用しない (9)
	ノイズ除去機能を使用する	使用しない (10)

図 2.22 スマート・コンフィグレータのシリアルコミュニケーションインタフェース(SCI7)の設定(1)

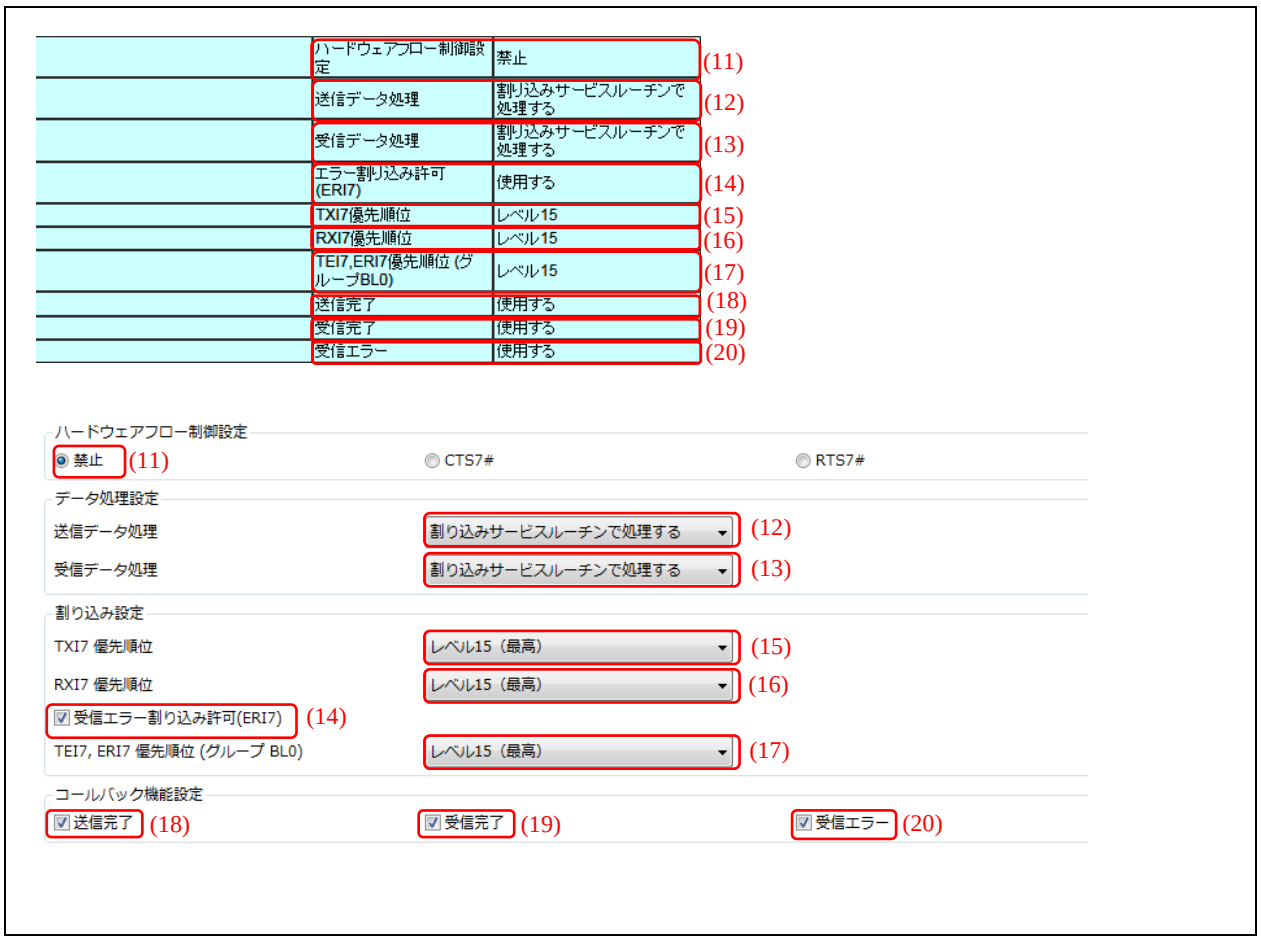


図 2.23 スマート・コンフィグレータのシリアルコミュニケーションインタフェース(SCI7)の設定(2)

表 2.13 シリアルコミュニケーションインタフェース(SCI7)の設定

	コード生成		スマート・コンフィグレータ	
	設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
(1)	スタートビット検出設定	RXD7 端子の立ち下がりエッジ	スタートビット検出設定	RXD7 端子の立ち下がりエッジ
(2)	データ・ビット長設定	8 ビット	データ・ビット長設定	8 ビット
(3)	パリティ設定	パリティなし	パリティ設定	禁止
(4)	ストップビット設定	1 ビット	ストップビット設定	1 ビット
(5)	データ転送方向設定	LSB ファースト	データ転送方向設定	LSB ファースト
(6)	転送クロック	内部クロック	[転送速度設定] 転送クロック	内部クロック
(7)	ビットレート	19200 (bps)	[転送速度設定] ビットレート	19200 (bps)
(8)	ビットレートモジュレーション機能有効	使用する	[転送速度設定] ビットレートモジュレーション機能有効	チェックあり
(9)	SCK7 端子機能	SCK7 を使用しない	[転送速度設定] SCK7 端子機能	SCK7 を使用しない
(10)	ノイズ除去機能を使用する	使用しない	[ノイズフィルタ設定] ノイズ除去機能を使用する	チェックなし
(11)	ハードウェアフロー制御設定	禁止	ハードウェアフロー制御設定	禁止
(12)	送信データ処理	割り込みサービスルーチンで処理する	[データ処理設定] 送信データ処理	割り込みサービスルーチンで処理する
(13)	受信データ処理	割り込みサービスルーチンで処理する	[データ処理設定] 受信データ処理	割り込みサービスルーチンで処理する
(14)	エラー割り込み許可(ERI7)	使用する	[割り込み設定] 受信エラー割り込み許可(ERI7)	チェックあり
(15)	TXI7 優先順位	レベル 15	[割り込み設定] TXI7 優先順位	レベル 15 (最高)
(16)	RXI7 優先順位	レベル 15	[割り込み設定] RXI7 優先順位	レベル 15 (最高)
(17)	TEI7,ERI7 優先順位 (グループ BL0)	レベル 15	[割り込み設定] TEI7,ERI7 優先順位 (グループ BL0)	レベル 15 (最高)
(18)	送信完了	使用する	[コールバック機能設定] 送信完了	チェックあり
(19)	受信完了	使用する	[コールバック機能設定] 受信完了	チェックあり
(20)	受信エラー	使用する	[コールバック機能設定] 受信エラー	チェックあり

(7)　端子の設定を行います。「端子」タブを選択し、「端子機能」タブを選択します。左のペインで「シリアルコミュニケーションインタフェース」の「SCI6」または「SCI7」を選択すると、右の端子機能ペインに使用される端子の一覧が表示されます。

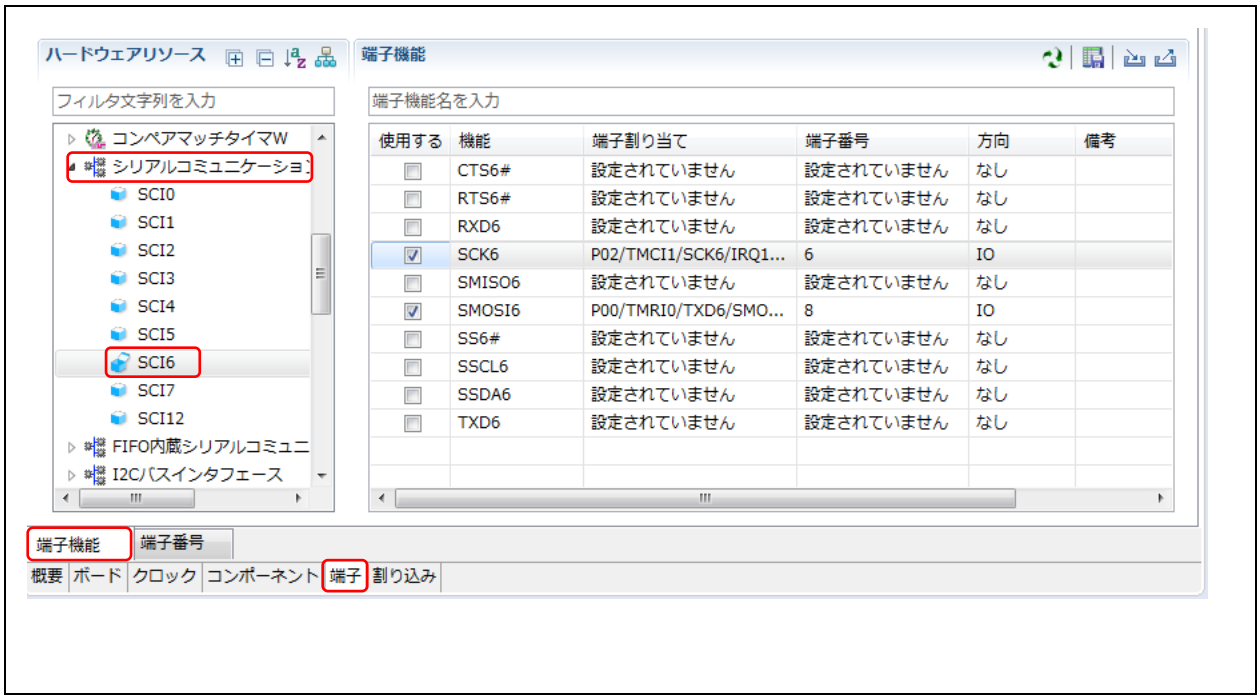


図 2.24 スマート・コンフィグレータのシリアルコミュニケーションインタフェースの端子設定(1)

(8)　SMOSI6、SCK6 の端子を割り当てます。

SMOSI6	P00
	使用する
データ転送方向設定	MSBファースト
受信データ･レベル設定	標準
転送クロック	内部クロック
ビットレート	1500000 (bps)
ビットレートモジュレーション機能有効	使用しない
SCK6端子機能	クロック出力
SCK6	P02

端子機能

端子機能名を入力

使用する	機能	端子割り当て	端子番号	方向	備考
☐	CTS6#	設定されていません	設定されていません	なし	
☐	RTS6#	設定されていません	設定されていません	なし	
☐	RXD6	設定されていません	設定されていません	なし	
☑	SCK6	P02/TMCI1/SCK6/IRQ1...	6	IO	
☐	SMISO6	設定されていません	設定されていません	なし	
☑	SMOSI6	P00/TMRI0/TXD6/SMO...	8	IO	
☐	SS6#	設定されていません	設定されていません	なし	
☐	SSCL6	設定されていません	設定されていません	なし	
☐	SSDA6	設定されていません	設定されていません	なし	
☐	TXD6	設定されていません	設定されていません	なし	

TXD7	P90
RXD7	P92

端子機能

端子機能名を入力

使用する	機能	端子割り当て	端子番号	方向	備考
☐	CTS7#	設定されていません	設定されていません	なし	
☐	RTS7#	設定されていません	設定されていません	なし	
☑	RXD7	P92/A18/D18/POE4#/E...	160	I	
☐	SCK7	設定されていません	設定されていません	なし	
☐	SMISO7	設定されていません	設定されていません	なし	
☐	SMOSI7	設定されていません	設定されていません	なし	
☐	SS7#	設定されていません	設定されていません	なし	
☐	SSCL7	設定されていません	設定されていません	なし	
☐	SSDA7	設定されていません	設定されていません	なし	
☑	TXD7	P90/A16/D16/ET1_RX_...	163	O	

図 2.25 スマート・コンフィグレータのシリアルコミュニケーションインタフェースの端子設定(2)

表 2.14 シリアルコミュニケーションインタフェース(SCI6、SCI7)の端子設定

コード生成		スマート・コンフィグレータ	
設定項目 (Macro.html の「Macro」又は「Setting」)	設定 (Macro.html の「Status」)	設定項目	設定
SCK6	P02	SCK6	P02/TMCI1/SCK6/IRQ10/AN120
SMOSI6	P00	SMOSI6	P00/TMRI0/TXD6/SMOSI6/SSDA6/IRQ8/AN118
TXD7	P90	TXD7	P90/A16/D16/ET1_RX_DV/TXD7/SMOSI7/SSDA7/AN114
RXD7	P92	RXD7	P92/A18/D18/POE4#/ET1_CRS/RMII1_CRS_DV/RXD7/SMOSO7/SSCL7/AN116

2.5.5 その他の周辺機能の設定

12 ビット A/D コンバータ、割り込み機能、ポート設定については、「表 2.2 コード生成とスマート・コンフィグレータの周辺機能の対応」および、「2.5.2 クロック発生回路の設定」、「2.5.3 コンペアマッチタイマの設定」、「2.5.4 シリアルコミュニケーションインタフェースの設定」の手順を参照し、同様にスマート・コンフィグレータの設定を行ってください。

2.5.6 コードの生成

全ての設定が完了したら、プロジェクトを保存し、スマート･コンフィグレータの「コードの生成」ボタン  を押し、コードを生成してください。

2.6 ユーザ定義ソースコードの移植

2.6.1 ユーザ定義ソースコードの移植の概要

コード生成を使用した移行元プロジェクトで、ユーザが作成したソースファイルや、コード生成が生成したソースファイルに書かれたユーザ定義ソースコードを、スマート･コンフィグレータを使用した移行先プロジェクトにコピーする必要があります。

図 2.26 に、ユーザ定義ソースコードの移植の手順を示します。

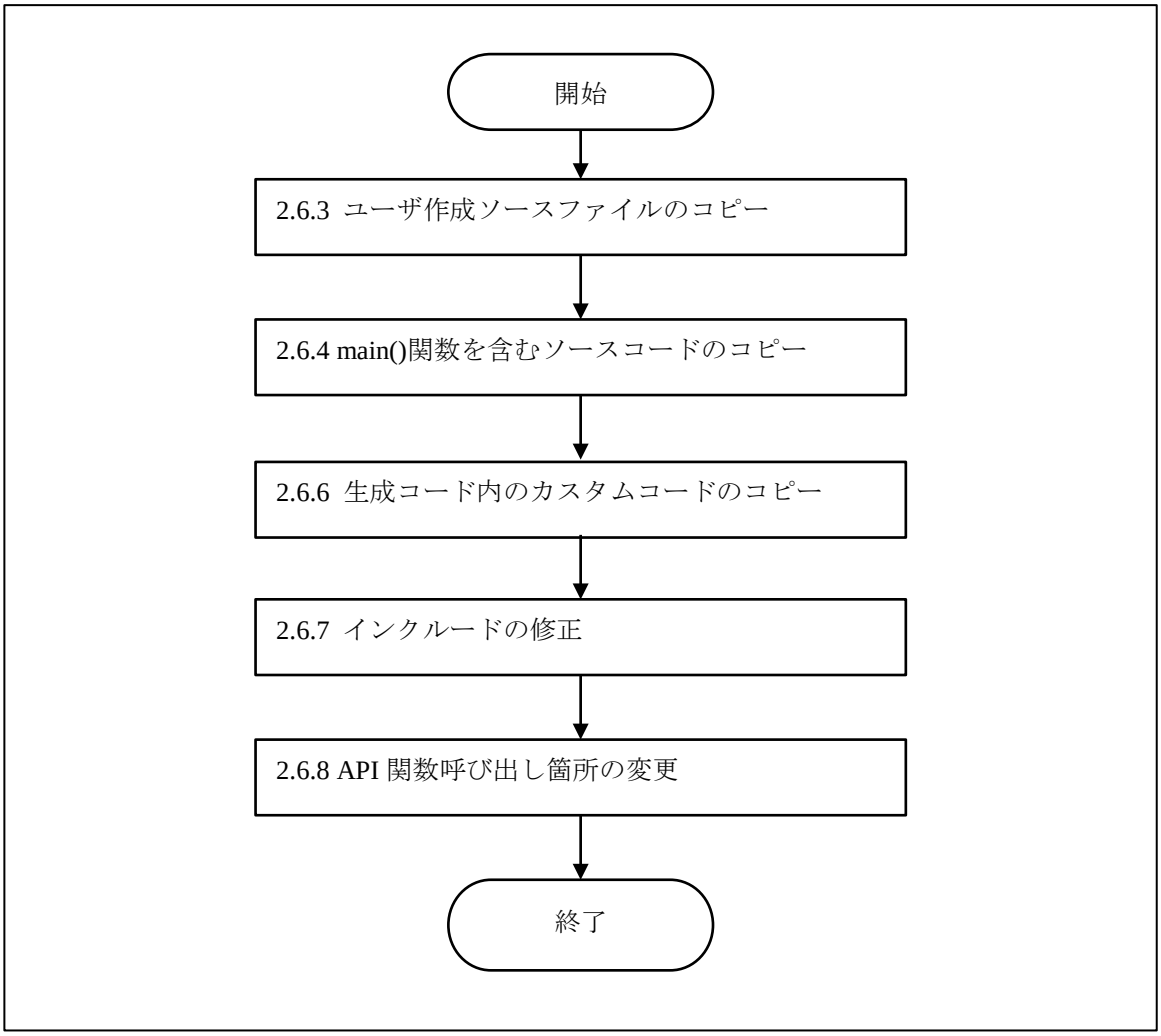


図 2.26 ユーザ定義ソースコードの移植の手順

2.6.2 ユーザ定義ソースコード記述領域について

コード生成、またはスマート・コンフィグレータによって生成されたファイル内には、ユーザが自由にコードを加えることができる領域が設けられています。カスタムコードは、以下のようなコメントで示されます。

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

上記のコメントのうち、‘xxxxxx’の部分は、カスタムコードを加えるエリアにより異なります。例えば、インクルードファイルを定義する箇所では’include’、グローバル変数を定義する箇所では’global’などのように記載されています。

これらのコメントで挟まれたカスタムコードを、移行元プロジェクトから移行先プロジェクトへコピーする必要があります。

2.6.3　ユーザ作成ソースファイルのコピー

移行元プロジェクトから、コード生成が出力したソースファイル以外の、ユーザ作成ソースファイルをコピーします。

以下の通り、移行元プロジェクトの’cg_src’フォルダ内以外のソースファイル、およびヘッダファイルを、移行先プロジェクトの’src’フォルダにコピーしてください。

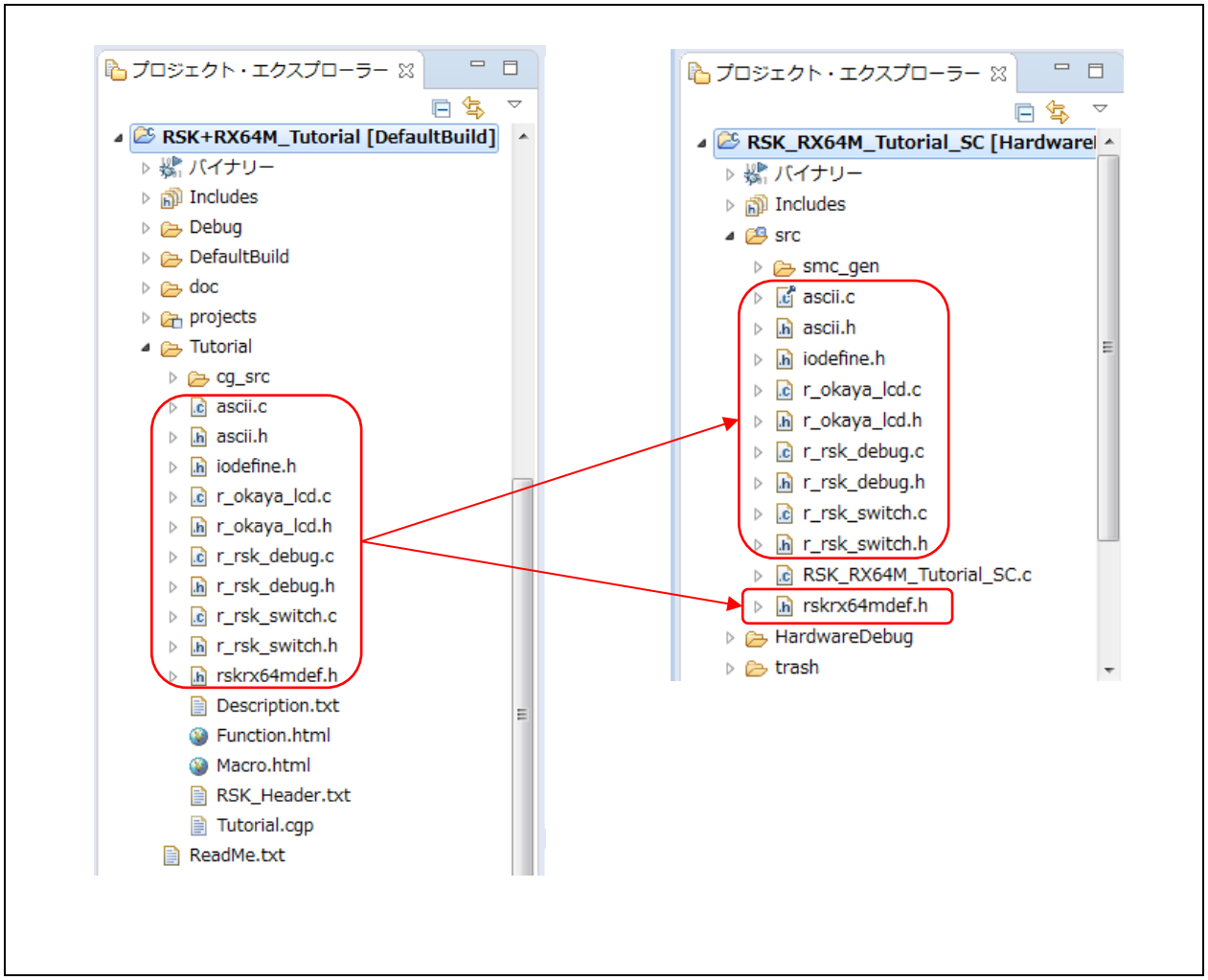


図 2.27 ユーザ作成ソースファイルのコピー

コピーしたソースファイルは、コード生成で生成した API 関数名を使用しているため、これらをスマート･コンフィグレータで生成した API 関数名に修正する必要があります。また、インクルードヘッダの記述についても、必要に応じて修正する必要があります。API 関数名の修正については、「2.6.8 API 関数呼び出し箇所の変更」を、インクルードの修正については、「2.6.7 インクルードの修正」を参照してください。

また、’src’フォルダにはインクルード対象となるヘッダファイルが含まれるため、’src’フォルダをインクルードディレクトリに追加する必要があります。以下の方法でインクルードディレクトリを追加してください。

(1) 移行先プロジェクト「RSK_RX64M_Tutorial_SC」を右クリックしてプロパティを開きます。左のペインで[C/C++ビルド]ー[設定]を選択します。右の設定画面で[ツール設定]タブを選択し、[Compiler]ー[ソース]を選択し、[インクルード・ファイルを検索するフォルダ]で追加ボタン をクリックします。

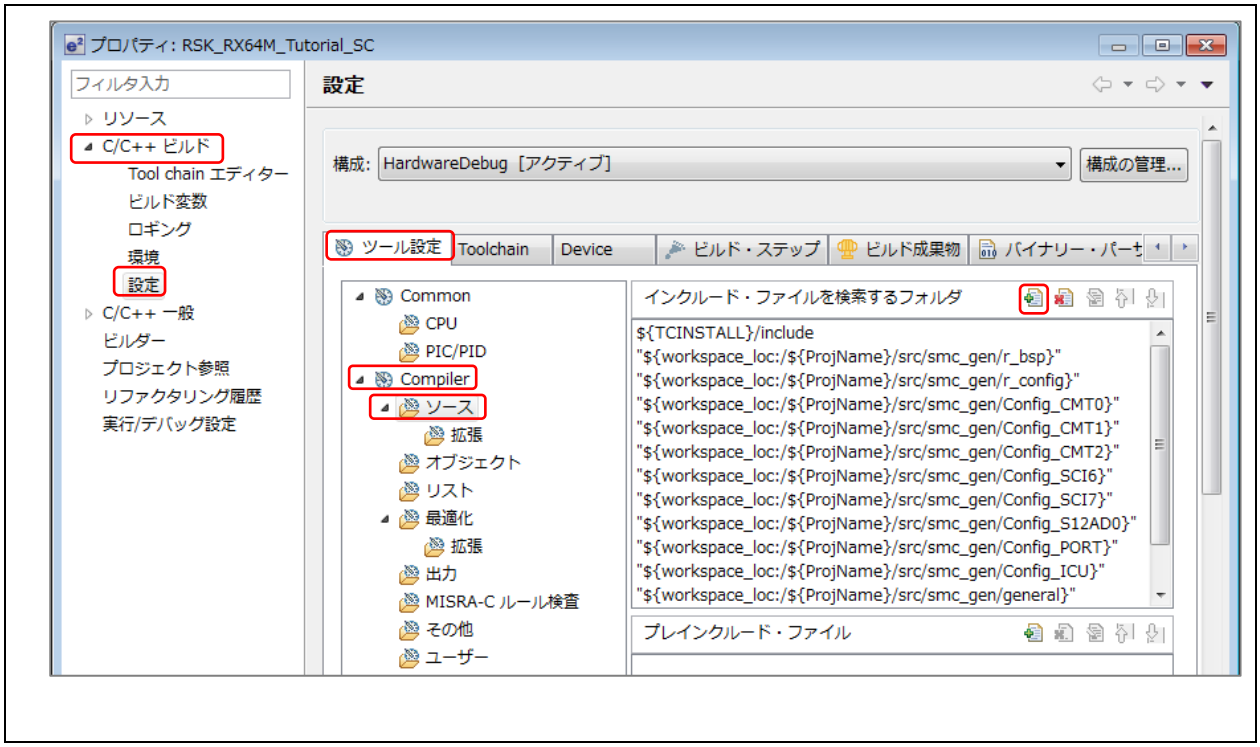


図 2.28 インクルードディレクトリの追加方法(1)

(2)　「ディレクトリー･パスの追加」画面で「ワークスペース」を選択し、「フォルダの選択」画面でインクルードディレクトリに追加するフォルダ(ここでは'src')を選択し「OK」を選択します。「ディレクトリー・パスの追加」画面で「ディレクトリー」に指定したフォルダが追加されたことを確認し、「OK」を選択します。

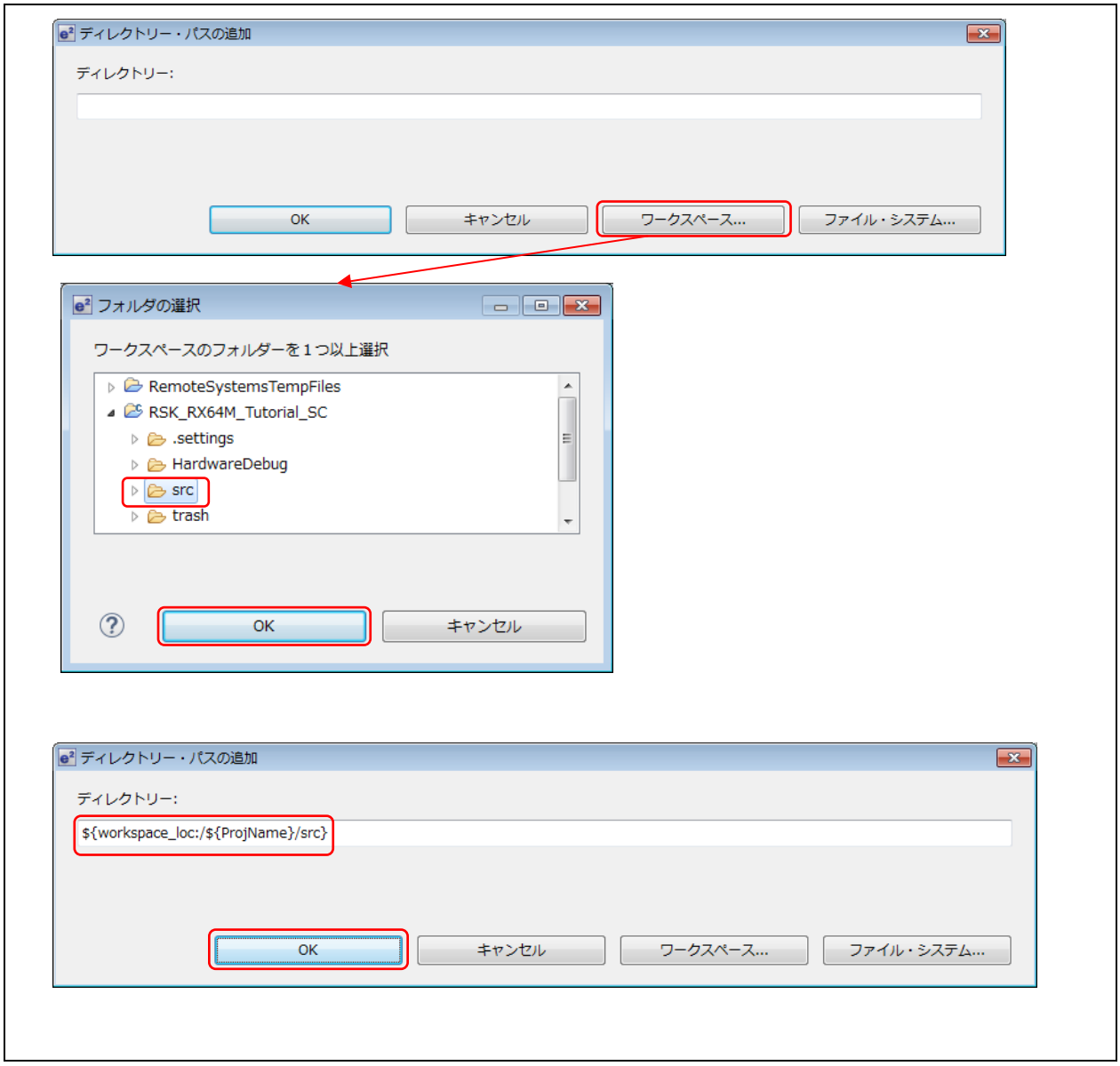


図 2.29 インクルードディレクトリの追加方法(2)

2.6.4 main()関数を含むソースコードのコピー

main()関数を含むソースファイルから、ユーザ定義ソースコードをコピーします。

移行元プロジェクトでは、main()関数は、’cg_src’フォルダの中の’r_cg_main.c’ファイルに含まれます。’r_cg_main.c’はコード生成が生成するソースファイルであるため、ユーザ定義ソースコードは、コメントに挟まれた領域に書かれています。

移行先プロジェクトでは、main()関数を含むファイルは、スマート･コンフィグレータが生成するファイルには含まれません。main()関数はプロジェクト新規作成時に自動で生成される{ProjName}.c ファイルに含まれます。本アプリケーションノートの移行先プロジェクト名は’RSK_RX64M_Tutorial_SC’であるため、main()関数は’ RSK_RX64M_Tutorial_SC.c’に含まれます。’{ProjName}.c’ 内のソースコードは全てユーザ定義ソースコードとなります。

‘r_cg_main.c’ を開き、「2.6.2 ユーザ定義ソースコード記述領域について」に記載したコメント内に書かれているソースコードを全てコピーしていきます。

例として、インクルードファイルの記述のコピーについて説明します。

インクルードファイルのコピーは、基本的には、「2.6.2 ユーザ定義ソースコード記述領域について」のコメント内に書かれている記述をコピーします。それ以外に、ユーザ定義ソースコードが含まれるヘッダファイル(ここでは、’r_cg_userdefine.h’)もコピーします。

‘{ProjName}.c’ 内の ’r_smc_entry.h’ のインクルードは、このソースファイルが新規プロジェクト作成時に生成された時に、自動で記述されるものです。

その他の’r_cg_main.c’に書かれているヘッダファイルのインクルード(ここでは、’r_cg_macrodriver.h’ ～ ‘r_cg_s12ad.h’) については、これらのヘッダファイルの中にユーザ定義ソースコードが含まれなければ、コピーする必要はありません。

コピー対象となる記述を、黄色のハイライトで示します。

r_cg_main.c

```
/***********************************************************************************************
************************************
Includes
***********************************************************************************************
************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
#include "r_cg_icu.h"
#include "r_cg_port.h"
#include "r_cg_cmt.h"
#include "r_cg_sci.h"
#include "r_cg_s12ad.h"
/* Start user code for include. Do not edit comment generated here */
#include "r_okaya_lcd.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "rskrx64mdef.h"
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"
```

{ProjName}.c

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "rskrx64mdef.h"
#include "r_cg_userdefine.h"
```

このほか、ユーザ定義のプロトタイプ宣言、関数定義など、「2.6.2 ユーザ定義ソースコード記述領域について」に記載したコメント内に書かれているソースコードを、順に’{ProjName}.c’にコピーしていきます。

黄色のハイライトで示したユーザ定義のプロトタイプ宣言をコピーしてください。

r_cg_main.c

```
/***********************************************************************************************
***********************************
Global variables and functions
***********************************************************************************************
***********************************/
/* Start user code for global. Do not edit comment generated here */

/* Prototype declaration for cb_switch_press */
static void cb_switch_press (void);

/* Prototype declaration for get_adc */
static uint16_t get_adc (void);

/* Prototype declaration for lcd_display_adc */
static void lcd_display_adc (const uint16_t adc_result);

/* Prototype declaration for uart_display_adc */
static void uart_display_adc (const uint8_t adc_count, const uint16_t adc_result);

/* Variable to store the A/D conversion count for user display */
static uint8_t adc_count = 0;

/* Prototype declaration for led_display_count */
static void led_display_count (const uint8_t count);

/* Variable for flagging user requested ADC conversion */
volatile uint8_t g_adc_trigger = FALSE;

/* End user code. Do not edit comment generated here */
```

{ProjName}.c

```
void main(void);

/* Prototype declaration for cb_switch_press */
static void cb_switch_press (void);

/* Prototype declaration for get_adc */
static uint16_t get_adc (void);

/* Prototype declaration for lcd_display_adc */
static void lcd_display_adc (const uint16_t adc_result);

/* Prototype declaration for uart_display_adc */
static void uart_display_adc (const uint8_t adc_count, const uint16_t adc_result);

/* Prototype declaration for led_display_count */
static void led_display_count (const uint8_t count);

/* Variable to store the A/D conversion count for user display */
static uint8_t adc_count = 0;

/* Variable for flagging user requested ADC conversion */
volatile uint8_t g_adc_trigger = FALSE;
```

黄色のハイライトで示した main()関数の関数定義をコピーしてください。「～中略～」として省略している箇所も全てコピー対象です。‘R_MAIN_UserInit()’ 関数のように、コード生成が生成する関数で、この関数内にユーザ定義コードを追加していない場合は、コピーする必要はありません。

r_cg_main.c

```
void R_MAIN_UserInit(void);

/***********************************************************************************************
***********************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************
***********************************/
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */

    /* Initialise the switch module */
    R_SWITCH_Init();


    /* Set the call back function when SW1 or SW2 is pressed */
    R_SWITCH_SetPressCallback(cb_switch_press);

    ～中略～

    /* Set up SCI7 receive buffer and callback function */
    R_SCI7_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI7 operations */
    R_SCI7_Start();

    while (1U)
    {
        uint16_t adc_result;

        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();

    ～中略～

            /* Send the result to the UART */
            uart_display_adc(adc_count, adc_result);

            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
    /* End user code. Do not edit comment generated here */
}
```

{ProjName}.c

```
void main(void)
{
    /* Initialise the switch module */
    R_SWITCH_Init();

    /* Set the call back function when SW1 or SW2 is pressed */
    R_SWITCH_SetPressCallback(cb_switch_press);

    ～中略～

    while (1U)
    {
        uint16_t adc_result;

        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();

    ～中略～

            /* Send the result to the UART */
            uart_display_adc(adc_count, adc_result);

            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
}
```

黄色のハイライトで示したプライベート関数の関数定義をコピーしてください。「～中略～」として省略している箇所も全てコピー対象です。‘R_MAIN_UserInit()’関数のように、コード生成が生成する関数で、この関数内にユーザ定義コードを追加していない場合は、コピーする必要はありません。

r_cg_main.c

```
/***********************************************************************************************
***********************************
* Function Name: R_MAIN_UserInit
* Description  : This function adds user code before implementing main function.
* Arguments    : None
* Return Value : None
***********************************************************************************************
***********************************/
void R_MAIN_UserInit(void)
{
    /* Start user code. 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 */

/***********************************************************************************
* Function Name : cb_switch_press
* Description   : Switch press callback function. Sets g_adc_trigger flag.
* Argument      : none
* Return value  : none
***********************************************************************************/
static void cb_switch_press (void)
{
    /* Check if switch 1 or 2 was pressed */
    if (g_switch_flag & (SWITCHPRESS_1 | SWITCHPRESS_2))
    {
        /* set the flag indicating a user requested A/D conversion is required */
        g_adc_trigger = TRUE;

        /* Clear flag */
        g_switch_flag = 0x0;
    }
}

/***********************************************************************************
* End of function cb_switch_press
***********************************************************************************/

    ～中略～

/***********************************************************************************
* Function Name : led_display_count
* Description   : Converts count to binary and displays on 4 LEDS0-3
* Argument      : uint8_t count
* Return value  : none
***********************************************************************************/
static void led_display_count (const uint8_t count)
{
    /* Set LEDs according to lower nibble of count parameter */
    LED0 = (uint8_t)((count & 0x01) ? LED_ON : LED_OFF);
    LED1 = (uint8_t)((count & 0x02) ? LED_ON : LED_OFF);
    LED2 = (uint8_t)((count & 0x04) ? LED_ON : LED_OFF);
    LED3 = (uint8_t)((count & 0x08) ? LED_ON : LED_OFF);
}

/***********************************************************************************
* End of function led_display_count
***********************************************************************************/


/* End user code. Do not edit comment generated here */
```

{ProjName}.c

```
/***********************************************************************
* Function Name : cb_switch_press
* Description   : Switch press callback function. Sets g_adc_trigger flag.
* Argument      : none
* Return value  : none
***********************************************************************/
static void cb_switch_press (void)
{
    /* Check if switch 1 or 2 was pressed */
    if (g_switch_flag & (SWITCHPRESS_1 | SWITCHPRESS_2))
    {
        /* set the flag indicating a user requested A/D conversion is required */
        g_adc_trigger = TRUE;

        /* Clear flag */
        g_switch_flag = 0x0;
    }
}

/***********************************************************************
* End of function cb_switch_press
***********************************************************************/


    ～中略～

/***********************************************************************
* Function Name : led_display_count
* Description   : Converts count to binary and displays on 4 LEDS0-3
* Argument      : uint8_t count
* Return value  : none
***********************************************************************/
static void led_display_count (const uint8_t count)
{
    /* Set LEDs according to lower nibble of count parameter */
    LED0 = (uint8_t)((count & 0x01) ? LED_ON : LED_OFF);
    LED1 = (uint8_t)((count & 0x02) ? LED_ON : LED_OFF);
    LED2 = (uint8_t)((count & 0x04) ? LED_ON : LED_OFF);
    LED3 = (uint8_t)((count & 0x08) ? LED_ON : LED_OFF);
}

/***********************************************************************
* End of function led_display_count
***********************************************************************/
```

2.6.5 コード生成とスマート･コンフィグレータの生成コードの対応について

コード生成により生成されたファイルと、スマート・コンフィグレータにより生成されたファイルは、対になっておらず、異なるフォルダ構成で出力されるため、適切なファイルにコピーする必要があります。

以下に、移行元プロジェクトから移行先プロジェクトへユーザ作成コードのコピーが必要な、主なファイル、および出力フォルダの概要を示します。

表 2.15 コード生成による生成コードとスマート・コンフィグレータによる生成コードの対応

コード生成		スマート･コンフィグレータ		備考
出力フォルダ	ソースファイル	出力フォルダ	ソースファイル	
cg_src	r_cg_main.c	src	{ProjName}.c	main()を含むファイルです。
cg_src	r_cg_userdefine.h	src¥smc_gen¥general	r_cg_userdefine.h	周辺機能共通で使用するユーザ定義コードのヘッダファイルです。
cg_src	r_cg_xxx.c	src¥smc_gen¥ Config_XXX	Config_XXX.c	周辺機能の初期化や操作を行うためのソースファイルです。スマート･コンフィグレータではリソースごとに 1 つのファイルが出力されます。
	r_cg_xxx_user.c	src¥smc_gen¥ Config_XXX	Config_XXX_user.c	周辺機能初期化の後に追加するユーザ定義コードや、割り込みコールバック関数を記述するためのソースファイルです。スマート･コンフィグレータではリソースごとに 1 つのファイルが出力されます。
	r_cg_xxx.h	src¥smc_gen¥ general	r_cg_xxx.h	SFR レジスタ設定のためのマクロ定義を含むヘッダファイルです。周辺機能に共通で使用されます。
		src¥smc_gen¥ Config_XXX	Config_XXX.h	Config_XXX.c のヘッダファイルです。

※ ‘xxx’および’XXX’は周辺機能名を表します。

以下に、RSK+RX64M サンプルコードで、カスタムコードを移行する必要のあるファイルについて、コード生成による生成コードと、スマート・コンフィグレータによる生成コードの対応を示します。背景色がグレーのファイルは、今回の例ではカスタムコードを移行する必要のないファイル、それ以外のファイルはカスタムコードを移行する必要のあるファイルです。

表 2.16 RSK+RX64M サンプルコードでカスタムコードを移行する必要のあるファイル(1)

周辺機能	コード生成		スマート･コンフィグレータ	
	出力フォルダ	ソースファイル	出力フォルダ	ソースファイル
main()を含むファイル	cg_src	r_cg_main.c	src	{ProjName}.c
全般設定	cg_src	r_cg_userdefine.h	src¥smc_gen¥general	r_cg_userdefine.h
割り込みコントローラ	cg_src	r_cg_icu.c	src¥smc_gen¥ Config_ICU	Config_ICU.c
			src¥smc_gen¥ general	r_smc_interrupt.c
		r_cg_icu_user.c	src¥smc_gen¥ Config_ICU	Config_ICU_user.c
		r_cg_icu.h	src¥smc_gen¥ general	r_cg_icu.h
			src¥smc_gen¥ Config_ICU	Config_ICU.h
			src¥smc_gen¥ general	r_smc_interrupt.h
I/O ポート	cg_src	r_cg_port.c	src¥smc_gen¥ Config_PORT	Config_PORT.c
		r_cg_port_user.c	src¥smc_gen¥ Config_PORT	Config_PORT_user.c
		r_cg_port.h	src¥smc_gen¥ general	r_cg_port.h
			src¥smc_gen¥ Config_PORT	Config_PORT.h
コンペアマッチタイマ	cg_src	r_cg_cmt.c	src¥smc_gen¥ Config_CMT0	Config_CMT0.c
			src¥smc_gen¥ Config_CMT1	Config_CMT1.c
			src¥smc_gen¥ Config_CMT2	Config_CMT2.c
		r_cg_cmt_user.c	src¥smc_gen¥ Config_CMT0	Config_CMT0_user.c
			src¥smc_gen¥ Config_CMT1	Config_CMT1_user.c
			src¥smc_gen¥ Config_CMT2	Config_CMT2_user.c
		r_cg_cmt.h	src¥smc_gen¥ general	r_cg_cmt.h
			src¥smc_gen¥ Config_CMT0	Config_CMT0.h
			src¥smc_gen¥ Config_CMT1	Config_CMT1.h
			src¥smc_gen¥ Config_CMT2	Config_CMT2.h

表 2.17 RSK+RX64M サンプルコードでカスタムコードを移行する必要のあるファイル(2)

周辺機能	コード生成		スマート･コンフィグレータ	
	出力フォルダ	ソースファイル	出力フォルダ	ソースファイル
シリアルコミュニケーションインタフェース	cg_src	r_cg_sci.c	src¥smc_gen¥ Config_SCI6	Config_SCI6.c
			src¥smc_gen¥ Config_SCI7	Config_SCI7.c
		r_cg_sci_user.c	src¥smc_gen¥ Config_SCI6	Config_SCI6_user.c
			src¥smc_gen¥ Config_SCI7	Config_SCI7_user.c
		r_cg_sci.h	src¥smc_gen¥ general	r_cg_sci.h
			src¥smc_gen¥ Config_SCI6	Config_SCI6.h
			src¥smc_gen¥ Config_SCI7	Config_SCI7.h
12 ビット A/D コンバータ	cg_src	r_cg_s12ad.c	src¥smc_gen¥ Config_S12AD0	Config_S12AD0.c
		r_cg_s12ad_user.c	src¥smc_gen¥ Config_S12AD0	Config_S12AD0_user.c
		r_cg_s12ad.h	src¥smc_gen¥ general	r_cg_s12ad.h
			src¥smc_gen¥ Config_S12AD0	Config_S12AD0.h

2.6.6 生成コード内のカスタムコードのコピー

表 2.16 の対応に従い、シリアルコミュニケーションインタフェースの SCI6(SPI マスタ送信機能)を例に、移行元プロジェクトのファイルから移行先プロジェクトのファイルにカスタムコードをコピーする手順について説明します。

まず、‘r_cg_sci.h’から、ハイライトで示した SCI6 についてのカスタムコードを、’ Config_SCI6.h’にコピーします。

r_cg_sci.h

```
/* Start user code for function. Do not edit comment generated here */
/* Exported functions used to transmit a number of bytes and wait for completion */
MD_STATUS R_SCI6_SPIMasterTransmit(uint8_t * const tx_buf, const uint16_t tx_num);
MD_STATUS R_SCI7_AsyncTransmit(uint8_t * const tx_buf, const uint16_t tx_num);
```

Config_SCI6.h

```
/* Start user code for function. Do not edit comment generated here */
/* Exported functions used to transmit a number of bytes and wait for completion */
MD_STATUS R_SCI6_SPIMasterTransmit(uint8_t * const tx_buf, const uint16_t tx_num);
/* End user code. Do not edit comment generated here */
```

次に、’r_cg_sci_user.c’から、ハイライトで示した SCI6 についてのカスタムコードを、’Config_SCI6_user.c’ にコピーします。

r_cg_sci_user.c

```
/* Start user code for global. Do not edit comment generated here */
/* Flag used locally to detect transmission complete */

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

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

/* Flag used locally to detect transmission complete */
static volatile uint8_t sci6_txdone;
static volatile uint8_t sci7_txdone;

～中略～

static void r_sci6_callback_transmitend(void)
{
    /* Start user code. Do not edit comment generated here */
    sci6_txdone = TRUE;
    /* End user code. Do not edit comment generated here */
}

～中略～

/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************
* Function Name: R_SCI6_SPIMasterTransmit
* Description  : This function sends SPI6 data to slave device.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************/
MD_STATUS R_SCI6_SPIMasterTransmit (uint8_t * const tx_buf,
                                    const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* clear the flag before initiating a new transmission */
    sci6_txdone = FALSE;

    /* Send the data using the API */
    status = R_SCI6_SPI_Master_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == sci6_txdone)
    {
        /* Wait */
    }

    return (status);
}

/***********************************************************************
* End of function R_SCI6_SPIMasterTransmit
***********************************************************************/
```

水色のハイライトで示した ’R_SCI6_SPI_Master_Send’ は、コード生成の API 関数名であるため、スマート・コンフィグレータの API 関数名に修正する必要があります。これについては、「2.6.8 API 関数呼び出し箇所の変更」で具体的な修正の手順を示します。

Config_SCI6_user.c

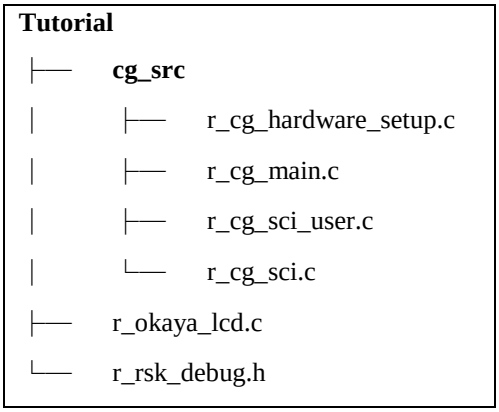
```
extern uint8_t *gp_sci6_tx_address;             /* SCI6 transmit buffer address */
extern uint16_t g_sci6_tx_count;                /* SCI6 transmit data number */
/* Start user code for global. Do not edit comment generated here */
/* Flag used locally to detect transmission complete */
static volatile uint8_t sci6_txdone;
/* End user code. Do not edit comment generated here */

～中略～

static void r_Config_SCI6_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI6_callback_transmitend. Do not edit comment
generated here */
        sci6_txdone = TRUE;
        /* End user code. Do not edit comment generated here */
}

～中略～

/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************
* Function Name: R_SCI6_SPIMasterTransmit
* Description  : This function sends SPI6 data to slave device.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************************/
MD_STATUS R_SCI6_SPIMasterTransmit (uint8_t * const tx_buf,
                                    const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* clear the flag before initiating a new transmission */
    sci6_txdone = FALSE;

    /* Send the data using the API */
    status = R_SCI6_SPI_Master_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == sci6_txdone)
    {
        /* Wait */
    }

    return (status);
}

/***********************************************************************************
* End of function R_SCI6_SPIMasterTransmit
***********************************************************************************/
/* End user code. Do not edit comment generated here */
```

2.6.7 インクルードの修正

コード生成は ’r_cg_xxx.h’ などのように、1 つの周辺機能につき主に 1 つのファイルを出力します。スマート･コンフィグレータでは、周辺機能共通のヘッダファイルである’ r_cg_xxx.h’ と、コンポーネントのリソースごとのヘッダファイルである ’Config_XXX.h’ に分け、複数のヘッダファイルを出力します。そのため、ヘッダファイルをインクルードしているソースコードの記述を、適切なヘッダファイル名に修正する必要があります。（’xxx’、’XXX’は周辺名を表します）

ここでは、例として「2.6.6 生成コード内のカスタムコードのコピー」でコピーした ‘r_cg_sci.h’ をインクルードしているファイルを検索し、適切なインクルード記述に修正します。

移行元のプロジェクトで ’#include “r_cg_sci.h”’ を含むファイルを検索します。検索結果は以下の通りです。



’#include “r_cg_sci.h”’ を含むこれらのファイルのうち、「cg_src」フォルダ以下の’r_cg_main.c’（移行先プロジェクトでは{ProjName}.c）以外のファイルは、スマート･コンフィグレータにより適切なヘッダファイルがインクルードされているため、インクルード記述の修正は必要ありません。

‘r_okaya_lcd.c’、’r_rsk_debug.h’の２つのファイルについて、対応する移行先プロジェクトのソースファイルのインクルード記述を修正します。

- ソースファイルの場合(r_okaya_lcd.c)

移行先プロジェクトで、’ r_okaya_lcd.c’ を開き、SCI の関数をコールしている箇所を検索します。’r_okaya_lcd.c’では、

- R_Config_SCI6_Start()
- R_SCI6_SPIMasterTransmit()

の 2 関数をコールしており、これらのプロトタイプ宣言は、’ Config_SCI6.h’にあるため、このヘッダファイルをインクルードするように修正します。

r_okaya_lcd.c (修正前)

```
/* SPI Driver Layer */
#include "r_cg_sci.h"
```

r_okaya_lcd.c (修正後)

```
/* SPI Driver Layer */
#include "Config_SCI6.h"
```

- ヘッダファイルの場合(r_rsk_debug.h)

移行先プロジェクトで’r_rsk_debug.h’ を開き、SCI6 または SCI7 で使用されている関数名やグローバル変数宣言などが含まれているかを検索します。’r_rsk_debug.h’ には、R_SCI7_AsyncTransmit()関数を使用したマクロ定義があるため、’Config_SCI7.h’ のヘッダファイルをインクルードするように修正します。

r_rsk_debug.h (修正前)

```
#include "r_cg_macrodriver.h"
#include "r_cg_sci.h"
```

r_rsk_debug.h (修正後)

```
#include "r_cg_macrodriver.h"
#include "Config_SCI7.h"
```

次に、移行元プロジェクトで’r_rsk_debug.h’をインクルードしているファイルを検索します。以下の 2 ファイルが該当します。

Tutorial
- **cg_src**
  - r_cg_main.c
- r_rsk_debug.c

’r_cg_main.c’（移行先プロジェクトでは{ProjName}.c）で SCI6 または SCI7 の関数をコールしている箇所を検索すると、

- R_SCI7_Start ()
- R_SCI7_Serial_Receive()

の 2 関数をコールしています。

これらのプロトタイプ宣言の記述がある’Config_SCI7.h’ は、’r_rsk_debug.h’にすでにインクルードしているため、修正は完了となります。

なお、R_SCI7_Start ()、R_SCI7_Serial_Receive()の API 関数呼び出し箇所は「2.6.8 API 関数呼び出し箇所の変更」を参照し移行先の{ProjName}.c の記述を修正する必要があります。

2.6.8 API 関数呼び出し箇所の変更

「2.6.3 ユーザ作成ソースファイルのコピー」でコピーしたソースコードに含まれる、コード生成の API 関数呼び出し箇所を、スマート・コンフィグレータの API 関数名に変更する必要があります。

ここでは、「2.6.3 ユーザ作成ソースファイルのコピー」でコピーしたユーザ作成ソースファイル’r_okaya_lcd.c’内の R_LCD_Init()関数を例に説明します。

R_LCD_Init()では、2 つのユーザ定義関数と、 1 つの API 関数をコールしています。init_pmod_lcd()と R_LCD_ClearDisplay()はユーザ定義関数で、r_okaya_lcd.c 内に宣言があります。これらの関数については修正の必要はありません。

API 関数については、修正前の r_okaya_lcd.c では、コード生成が生成した API 関数 R_SCI6_Start()を呼んでいます。この関数は、スマート・コンフィグレータが生成する関数では、R_Config_SCI6_Start()となる(コンポーネント追加時に、デフォルトのコンフィグレーション名を使用した場合)ため、呼び出し箇所の API 関数名を変更する必要があります。

r_okaya_lcd.c (修正前)

```
void R_LCD_Init (void)
{
    /* Start SPI comm channel to LCD Display */
    R_SCI6_Start();

    /* initialise Standard PMOD display */
    init_pmod_lcd();

    /* clear the display before use */
    R_LCD_ClearDisplay(back_colour);
}
```

r_okaya_lcd.c (修正後)

```
void R_LCD_Init (void)
{
    /* Start SPI comm channel to LCD Display */
    R_Config_SCI6_Start();

    /* initialise Standard PMOD display */
    init_pmod_lcd();

    /* clear the display before use */
    R_LCD_ClearDisplay(back_colour);
}
```

表 2.18 に、コード生成が生成する API 関数名とスマート・コンフィグレータが生成する API 関数名の対応表を示します。対応表に従い、API 関数の呼び出し箇所を変更してください。

なお、表 2.18 に記載しているスマート・コンフィグレータの API 関数名は、コンポーネントの追加時にデフォルトのコンフィグレーション名を設定した場合のものです。コンフィグレーション名はユーザ設定可能なため、コンフィグレーション名により、API 関数名は異なります。

スマート・コンフィグレータの API 関数については、[e² studio のヘルプ] ― [e2 studio ユーザーガイド] ― [ビルドに関する機能] ― [スマート・コンフィグレータ] ― [API リファレンス] を参照してください。

表 2.18 コード生成が生成する API 関数名と
スマート･コンフィグレータが生成する API 関数名の対応(1)

	コード生成		スマート・コンフィグレータ	
	ファイル名	**API 関数名**	**ファイル名**	**API 関数名**
クロック発生回路				
	r_cg_cgc.c	R_CGC_Create	r_smc_cgc.c	R_CGC_Create
コンペア・マッチ･タイマ				
	r_cg_cmt.c	R_CMT0_Create	Config_CMT0.c	R_Config_CMT0_Create
		R_CMT0_Start		R_Config_CMT0_Start
		R_CMT0_Stop		R_Config_CMT0_Stop
		R_CMT1_Create	Config_CMT1.c	R_Config_CMT1_Create
		R_CMT1_Start		R_Config_CMT1_Start
		R_CMT1_Stop		R_Config_CMT1_Stop
		R_CMT2_Create	Config_CMT2.c	R_Config_CMT2_Create
		R_CMT2_Start		R_Config_CMT2_Start
		R_CMT2_Stop		R_Config_CMT2_Stop
	r_cg_cmt_user.c	―	Config_CMT0_user.c	R_Config_CMT0_Create_UserInit
		r_cmt_cmi0_interrupt		r_Config_CMT0_cmi0_interrupt
		―	Config_CMT1_user.c	R_Config_CMT1_Create_UserInit
		r_cmt_cmi1_interrupt		r_Config_CMT1_cmi1_interrupt
		―	Config_CMT2_user.c	R_Config_CMT2_Create_UserInit
		r_cmt_cmi2_interrupt		r_Config_CMT2_cmi2_interrupt
割り込みコントローラ				
	r_cg_icu.c	R_ICU_Create	Config_ICU.c	R_Config_ICU_Create
		R_ICU_IRQ2_Start		R_Config_ICU_IRQ2_Start
		R_ICU_IRQ2_Stop		R_Config_ICU_IRQ2_Stop
		R_ICU_IRQ5_Start		R_Config_ICU_IRQ5_Start
		R_ICU_IRQ5_Stop		R_Config_ICU_IRQ5_Stop
	r_cg_icu_user.c	―	Config_ICU_user.c	R_Config_ICU_Create_UserInit
		r_icu_irq2_interrupt		r_Config_ICU_irq2_interrupt
		r_icu_irq5_interrupt		r_Config_ICU_irq5_interrupt
I/O ポート				
	r_cg_port.c	R_PORT_Create	Config_PORT.c	R_Config_PORT_Create
	r_cg_port_user.c	―	Config_PORT_user.c	R_Config_PORT_Create_UserInit

表 2.19 コード生成が生成する API 関数名と
スマート･コンフィグレータが生成する API 関数名の対応(2)

	コード生成		スマート・コンフィグレータ	
	ファイル名	API 関数名	ファイル名	API 関数名
12 ビット A/D コンバータ				
	r_cg_s12ad.c	R_S12AD0_Create	Config_S12AD0.c	R_Config_S12AD0_Create
		R_S12AD0_Start		R_Config_S12AD0_Start
		R_S12AD0_Start		R_Config_S12AD0_Stop
		R_S12AD0_Get_ValueResult		R_Config_S12AD0_Get_ValueResult
		R_S12AD0_Set_CompareValue		R_Config_S12AD0_Set_CompareValue
	r_cg_s12ad_user.c	—	Config_S12AD0_user.c	R_Config_S12AD0_Create_UserInit
		r_s12ad0_interrupt		r_Config_S12AD0_interrupt
		r_s12ad0_compare_interrupt		r_Config_S12AD0_compare_interrupt
シリアルコミュニケーションインタフェース				
	r_cg_sci.c	R_SCI6_Create	Config_SCI6.c	R_Config_SCI6_Create
		R_SCI6_Start		R_Config_SCI6_Start
		R_SCI6_Stop		R_Config_SCI6_Stop
		R_SCI6_SPI_Master_Send		R_Config_SCI6_SPI_Master_Send
		R_SCI7_Create	Config_SCI7.c	R_Config_SCI7_Create
		R_SCI7_Start		R_Config_SCI7_Start
		R_SCI7_Stop		R_Config_SCI7_Stop
		R_SCI7_Serial_Receive		R_Config_SCI7_Serial_Receive
		R_SCI7_Serial_Send		R_Config_SCI7_Serial_Send
	r_cg_sci_user.c	—	Config_SCI6_user.c	R_Config_SCI6_Create_UserInit
		r_sci6_transmit_interrupt		r_Config_SCI6_transmit_interrupt
		r_sci6_transmitend_interrupt		r_Config_SCI6_transmitend_interrupt
		r_sci6_callback_transmitend		r_Config_SCI6_callback_transmitend
		—	Config_SCI7_user.c	R_Config_SCI7_Create_UserInit
		r_sci7_transmit_interrupt		r_Config_SCI7_transmit_interrupt
		r_sci7_transmitend_interrupt		r_Config_SCI7_transmitend_interrupt
		r_sci7_receive_interrupt		r_Config_SCI7_receive_interrupt
		r_sci7_receiveerror_interrupt		r_Config_SCI7_receiveerror_interrupt
		r_sci7_callback_transmitend		r_Config_SCI7_callback_transmitend
		r_sci7_callback_receiveend		r_Config_SCI7_callback_receiveend
		r_sci7_callback_receiveerror		r_Config_SCI7_callback_receiveerror

2.7 ビルドオプションの設定

新規作成した移行先プロジェクトでは、デフォルトのビルドオプションが適用されます。そのため、移行元プロジェクトのビルドオプションを、移行先プロジェクトに反映する必要があります。

「e² studio 統合開発環境 ユーザーズマニュアル 入門ガイド」の「4.1 ビルドオプションの設定」を参照し、移行元プロジェクトのビルドオプションを、移行先プロジェクトに設定してください。

3. 参考ドキュメント

ユーザーズマニュアル：ハードウェア

（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース

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

ユーザーズマニュアル：開発環境

RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）

（最新版をルネサス エレクトロニクスホームページから入手してください。）

e² studio 統合開発環境 ユーザーズマニュアル 入門ガイド (R20UT2858)

（最新版をルネサス エレクトロニクスホームページから入手してください。）

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ

http://www.renesas.com/

お問合せ先

http://www.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2017.11.01	-	初版発行

製品ご使用上の注意事項

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

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子
   （または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（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、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、全部または一部を問わず、改造、改変、複製、その他の不適切に使用しないでください。かかる改造、改変、複製等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（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、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、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できません。たとえ、意図しない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 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。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計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。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。また、当社製品および技術を、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   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。
10. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）記載の諸条件に抵触して当社製品が使用され、その使用から損害が生じた場合、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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その総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

(Rev.3.0-1 2016.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

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

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

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