e² studio

R20AN0673JJ1000
Rev.1.00
2022.04.05

Code Generator 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、Code Generator 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

- RL78/G23 グループ

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

関連ドキュメント

- RL78 スマート・コンフィグレータ ユーザーガイド：e² studio 編 (R20AN0579)
- 統合開発環境 e² studio ユーザーズマニュアル 入門ガイド (R20UT4819)
- RL78/G13 ルネサススタータキットのサンプルコード (CS+ for CC-RL) (R01AN2899)

目次

1. 概要 3
1.1 本ドキュメントの目的 3
1.2 動作環境 3

2. Code Generator 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 4
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 5
2.2 移行元プロジェクトのダウンロード 6
2.3 移行元プロジェクトのレポート生成 8
2.3.1 レポート生成の手順 8
2.4 移行先プロジェクトの新規作成 11
2.5 スマート・コンフィグレータでの周辺機能設定 11
2.5.1 Code Generator とスマート・コンフィグレータの周辺機能の対応 11
2.5.2 クロック発生回路の設定 12
2.5.3 タイマの設定 20
2.5.4 その他の周辺機能の設定 23
2.5.5 コードの生成 23
2.6 ユーザ定義ソースコードの移植 24
2.6.1 概要 24
2.6.2 ユーザ定義ソースコードを書き込む領域 24

2.6.3　ユーザ作成ソースファイルのコピー 25
2.6.4　main() 関数を含むソースコードのコピー 28
2.6.5　Code Generator とスマート・コンフィグレータの生成コードの対応 35
2.6.6　生成コード内のカスタムコードのコピー 37
2.6.7　include ディレクティブの修正 40
2.6.8　API 関数呼び出し箇所の変更 41
2.7　ビルドオプションの設定 43

3.　参考ドキュメント 44

改訂記録 45

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、RL78/G13 用 Code Generator を使用したプロジェクトを、同じパッケージの RL78/G23 デバイスを対象とした、設定方法や生成される関数名が異なるスマート・コンフィグレータ用の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。

e^2 studio の使い方については、「統合開発環境 e^2 studio 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	RL78/G23 グループ
エミュレータ	E2 Lite、E2
IDE	e^2 studio 2021-04 以降
ツールチェーン	RL78 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-RL78 V1.10.00

2. Code Generator 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、Code Generator 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。

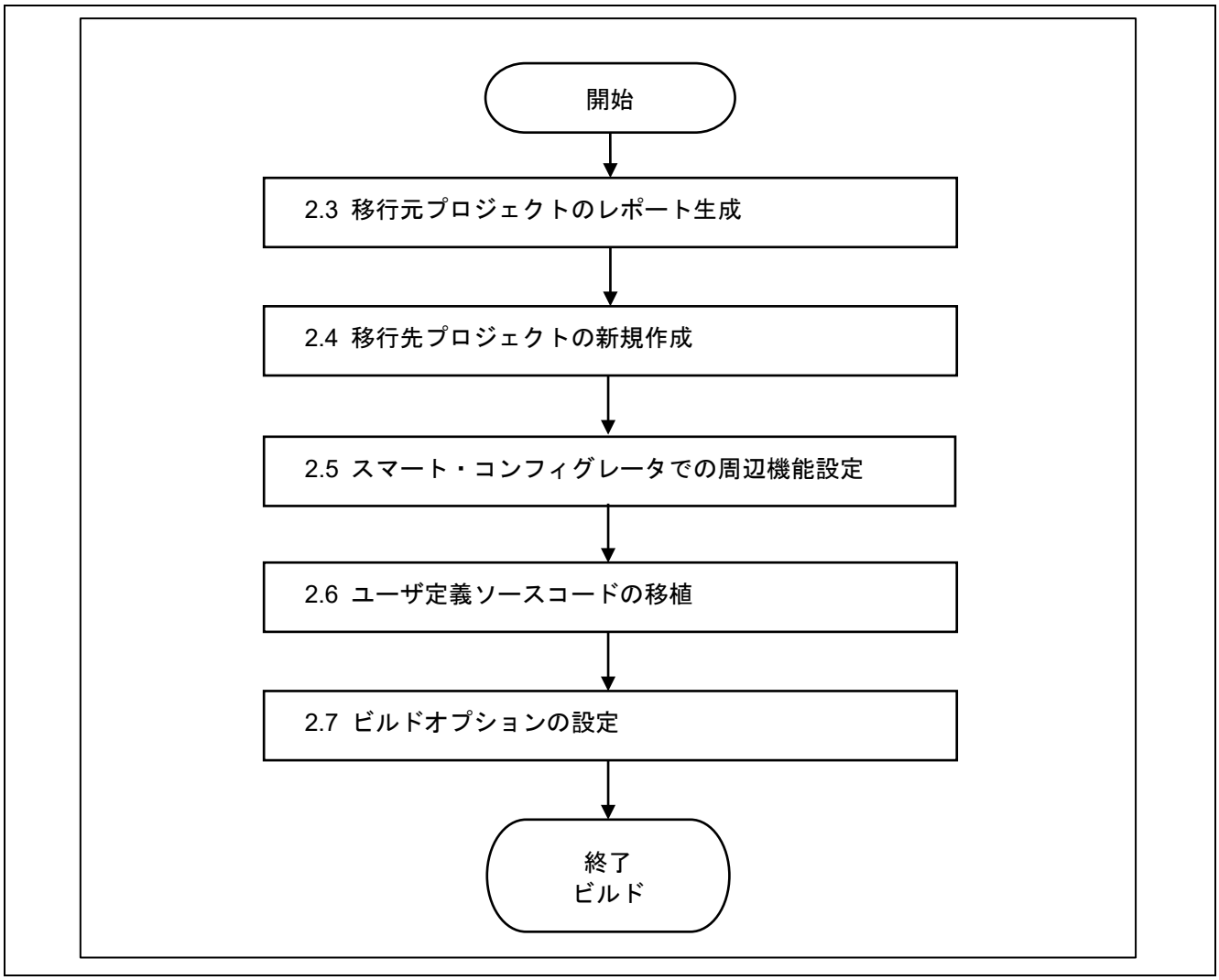

図 2.1　Code Generator を使用したプロジェクトをスマート・コンフィグレータを使用したプロジェクトに移行する手順

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、RL78/G13 ルネサススタータキットのサンプルコード (CS+ for CC-RL) 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
RSKRL78G13_Tutorial	移行元プロジェクトとして使用する、Code Generator を使用した、RL78/G13 ルネサススタータキットのサンプルコード (CS+ for CC-RL) 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。
RSKRL78G23_Tutorial_SC	スマート･コンフィグレータを使用するために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、図 2.1 に示す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 RL78/G13 ルネサススタータキットのサンプルコード (CS+ for CC-RL) 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】　プロジェクトをダウンロードするには、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/jp/ja）から、[製品情報]メニュー下の [RL78 低消費電力 8/16 ビット MCU] を選択し、[RL78 低消費電力 8/16 ビット MCU] ページの [ポートフォリオ] から [RL78/G13] 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) [RL78/G13] ページの [ボード&キット] のリストから [RL78/G13-Starter-Kit] 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) [RL78/G13-Starter-Kit] ページの [サンプルコード] 下の [RL78/G13 ルネサススタータキットのサンプルコード(CS+ for CC-RL) Rev.1.00 - Sample Code] をクリックして、ダウンロードを実行します。

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 Code Generator のレポート生成機能を使い、周辺機能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、スマート・コンフィグレータで移行先プロジェクトの周辺機能を設定します。

2.3.1　レポート生成の手順

- CS+の場合

(1) CS+を起動し、Code Generator を使用した移行元プロジェクト [RSKRL78G13_Tutorial] を開きます。
　プロジェクト・ツリーパネルの [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] メニューから [コード生成レポートを保存] を選択し、レポートを生成します。

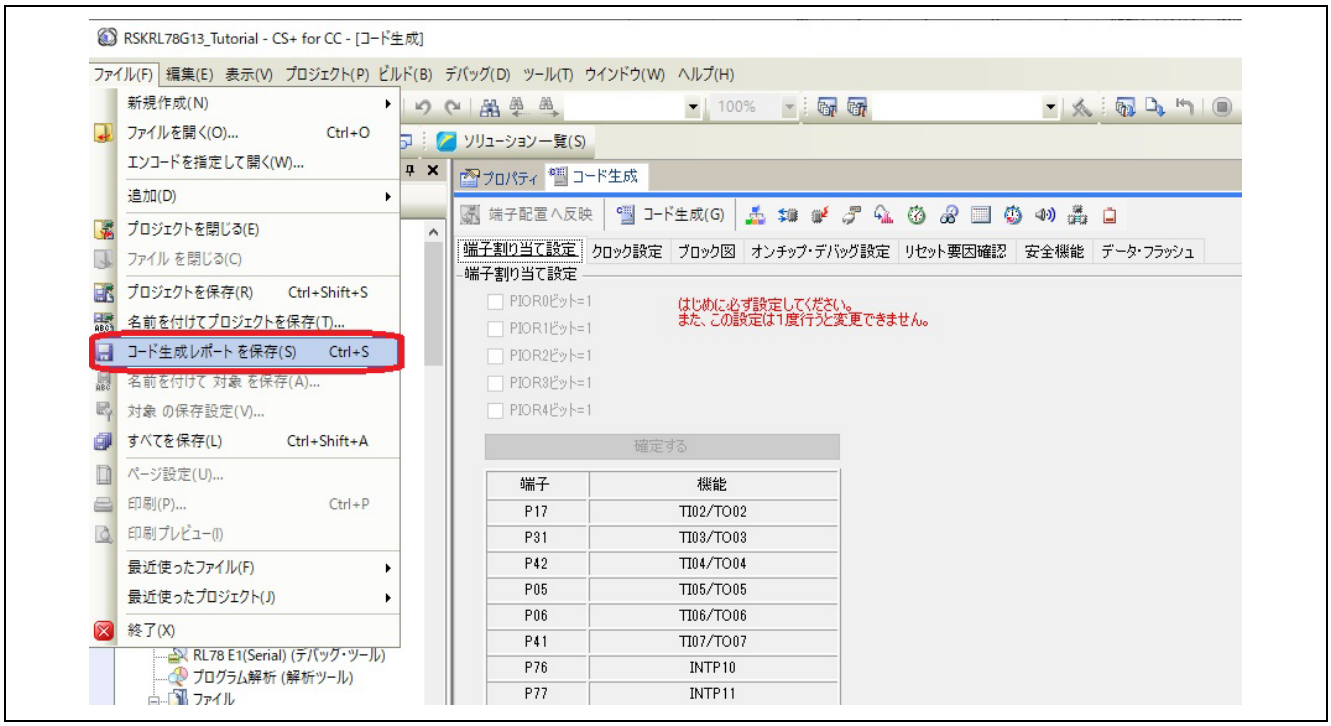

図 2.5　CS+のレポート生成の手順

(3) レポートの生成が完了すると、プロジェクトフォルダに、function.html と macro.html の 2 つのファイルが出力されます。

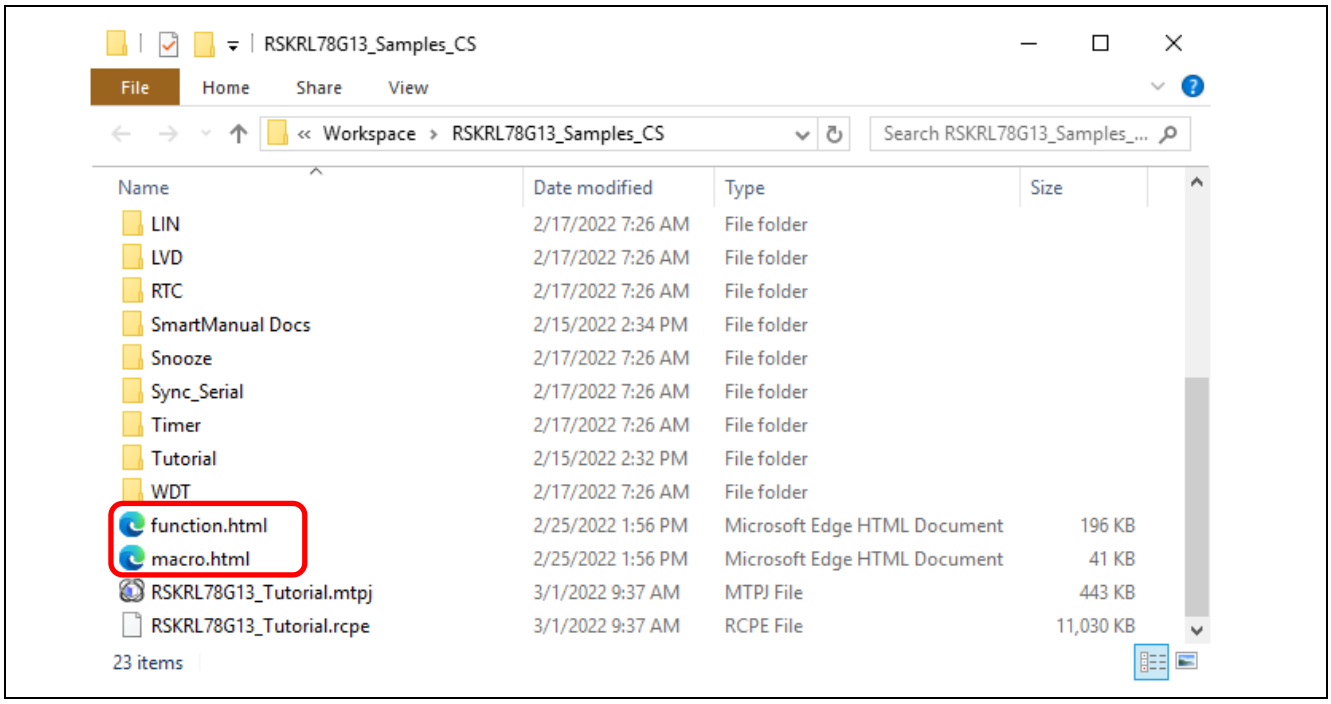

図 2.6　CS+の Code Generator のレポート機能が出力するレポートファイル

- e² studio の場合

(1) e² studio を起動し、Code Generator を使用した移行元プロジェクト [RSKRL78G13_Tutorial] を開きます。プロジェクト・ツリーパネルから [コード生成] を展開し、[周辺機能] をダブルクリックします。

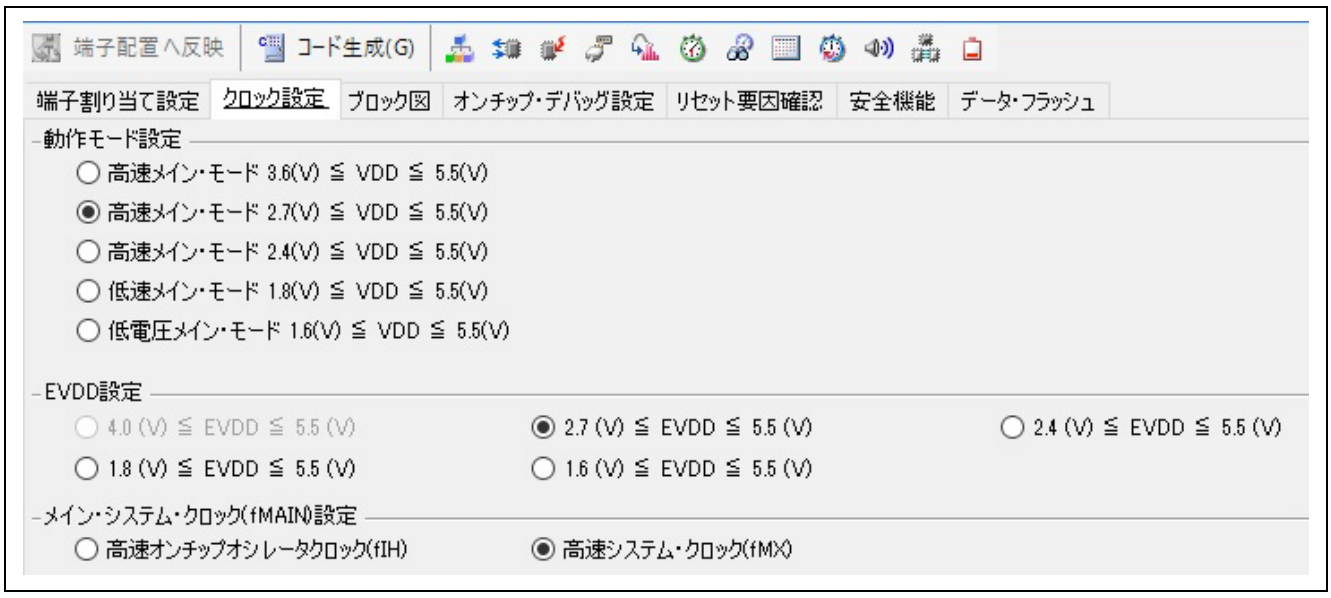

図 2.7　e² studio のレポート生成の手順

(2) レポートの生成が完了すると、Function.html と Macro.html の 2 つのファイルが doc フォルダに出力されます。

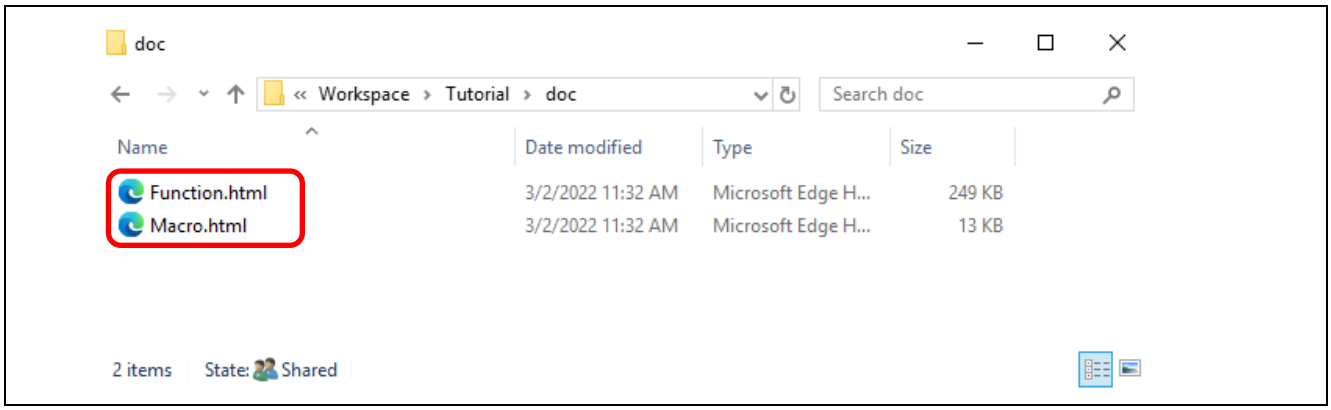

図 2.8　e^2 studio の Code Generator のレポート機能が出力するレポートファイル

表 2.2　Code Generator のレポート機能が出力するレポートファイル

ファイル名	概要
Function.html	Code Generator が生成する API 関数の一覧
Macro.html	Code Generator で設定した周辺機能の一覧

2.4　移行先プロジェクトの新規作成

スマート・コンフィグレータで使用する移行先プロジェクトとして、R7F100GLGxFA をターゲットデバイスとした C プロジェクトを新規に作成します。プロジェクトの作成方法については、「RL78 スマート・コンフィグレータ ユーザーガイド: e² studio 編」の「2.1　C 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　Code Generator とスマート・コンフィグレータの周辺機能の対応

RSKRL78G13_Tutorial プロジェクトで設定が必要な周辺機能について、Code Generator 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　Code Generator とスマート・コンフィグレータの周辺機能の対応

Code Generator			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	
ポート	ポート 5	P52	コンポーネント	ポート	ポート 5	P52
		P53				P53
		P54				P54
		P55				P55
	ポート 6	P62			ポート 6	P62
		P63				P63
	ポート 7	P70			ポート 7	P70
		P71				P71
		P72				P72
		P73				P73
割り込み	INTP1	ー	コンポーネント	割り込みコントローラ	INTP1	ー
	INTP2	ー	コンポーネント	割り込みコントローラ	INTP2	ー
	INTP4	ー	コンポーネント	割り込みコントローラ	INTP4	ー
A/D コンバータ	動作モードの設定	ー	コンポーネント	A/D コンバータ	動作モードの設定	ー
	A/D チャネルの選択	ー	コンポーネント	A/D コンバータ	A/D チャネルの選択	ー
タイマ	インターバル・タイマ	インターバル時間(16 ビット)	コンポーネント	インターバル・タイマ	16 ビットカウントモード	インターバル時間(16 ビット)

「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・コンフィグレータを設定します。

この節ではクロック発生回路とタイマの設定について説明します。他の周辺機能も同じ手順に従って設定してください。

2.5.2　クロック発生回路の設定

クロック発生回路を設定します。

(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: RL78/G13(ROM:64KB)
Chip name: R5F100LE

モジュール	マクロ	サブ	設定	状態
クロック発生回路				使用する
	CGC			使用する
			端子割り当て設定-PIOR0ビット=1	使用しない
			端子割り当て設定-PIOR1ビット=1	使用しない
			端子割り当て設定-PIOR2ビット=1	使用しない
			端子割り当て設定-PIOR3ビット=1	使用しない
			端子割り当て設定-PIOR4ビット=1	使用しない
			動作モード設定	高速メイン・モード 2.7(V) ≦ VDD ≦ 5.5(V)
			EVDD設定	2.7 (V) ≦ EVDD ≦ 5.5 (V)
			メイン・システム・クロック(fMAIN)設定	高速システム・クロック(fMX)
			fIH 動作	使用しない
			fMX 動作	使用する
			高速システム・クロック設定	X1発振(fX)
			fMX 周波数	20(MHz)
			発振安定時間	6553.6 (2^17/fX)(μs)
			fSUB 動作	使用する

図 2.9　Code Generator が出力したクロック発生回路のレポート

(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。

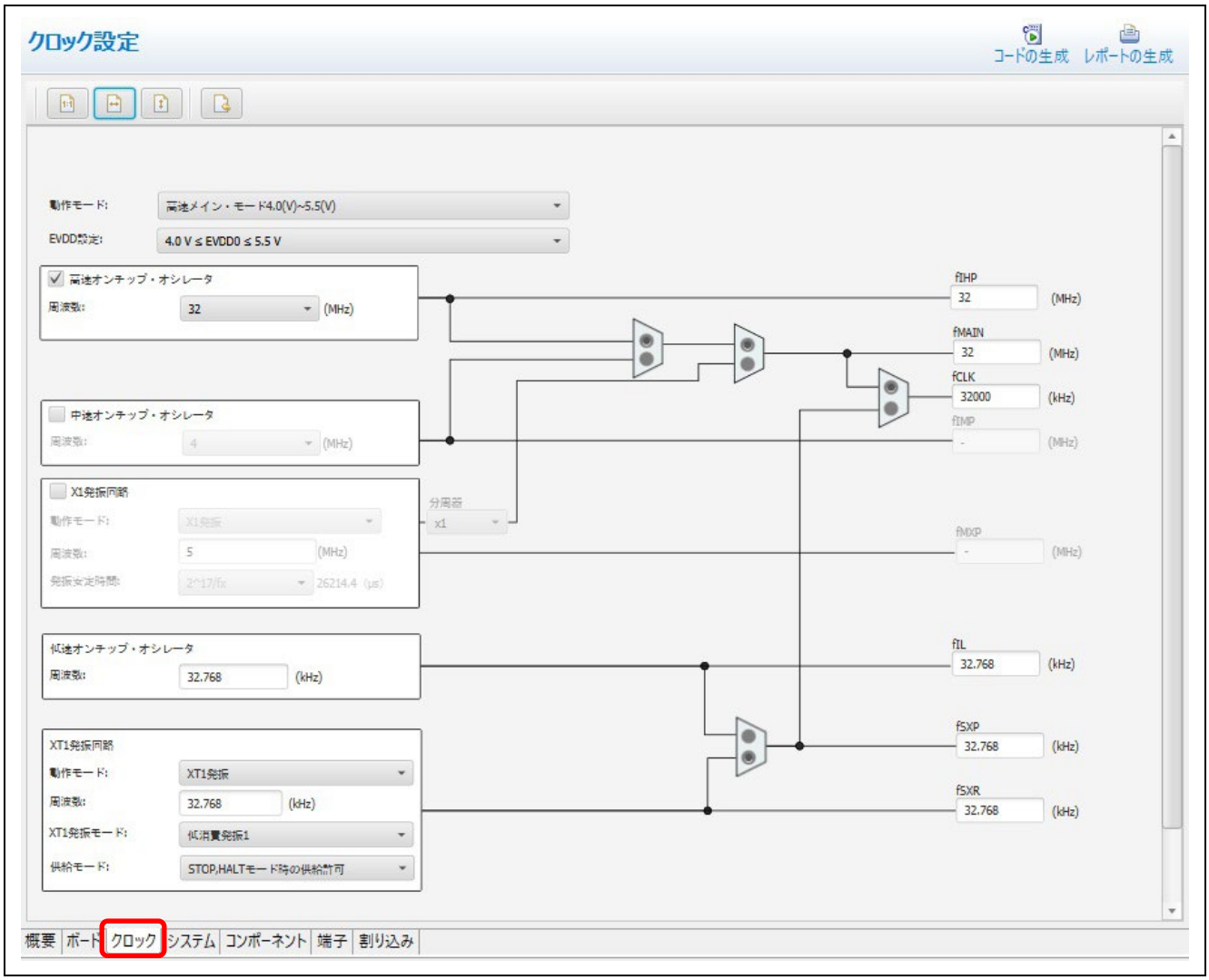

図 2.10　スマート・コンフィグレータのクロック設定画面

(3) レポートの Macro.html ファイルの [設定] と [状態] 欄の設定をスマート・コンフィグレータの設定に反映します。

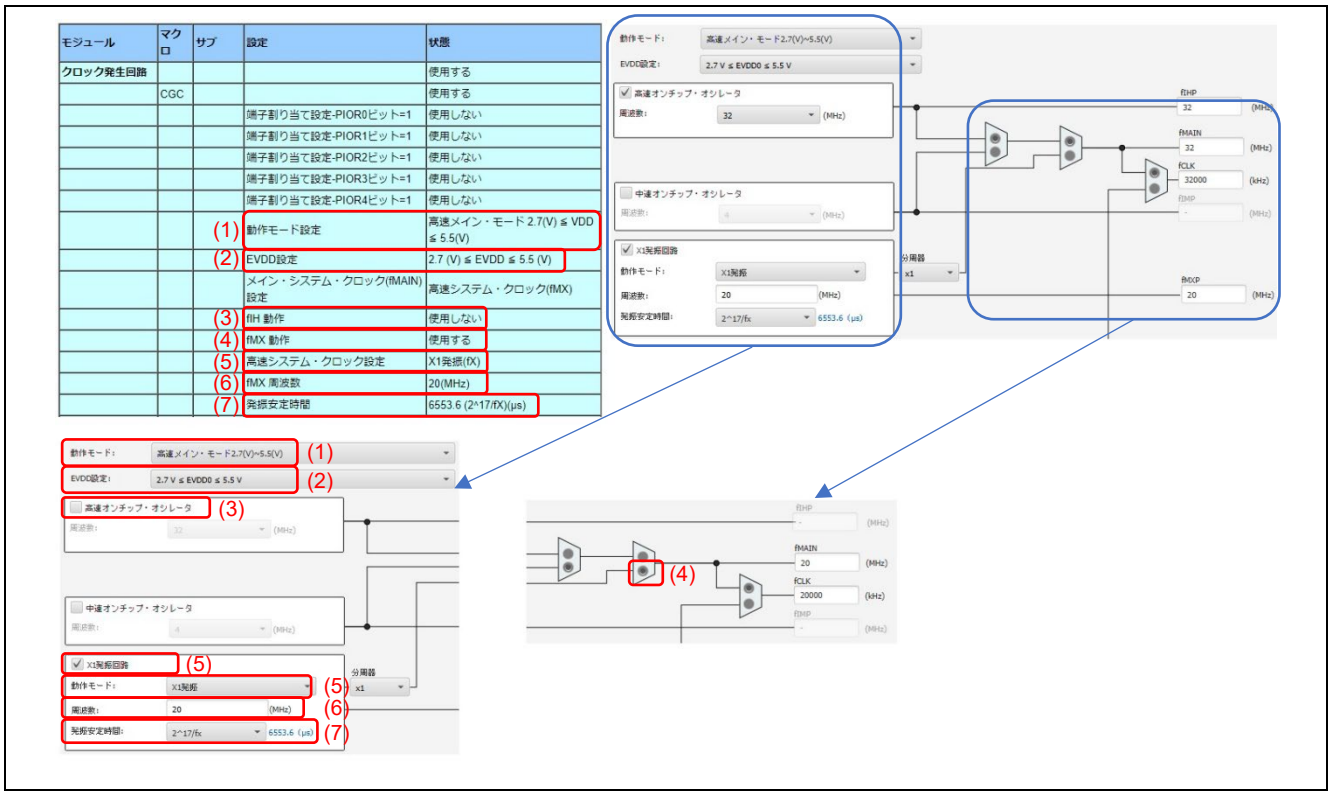

図 2.11　スマート・コンフィグレータのクロック設定 (1)

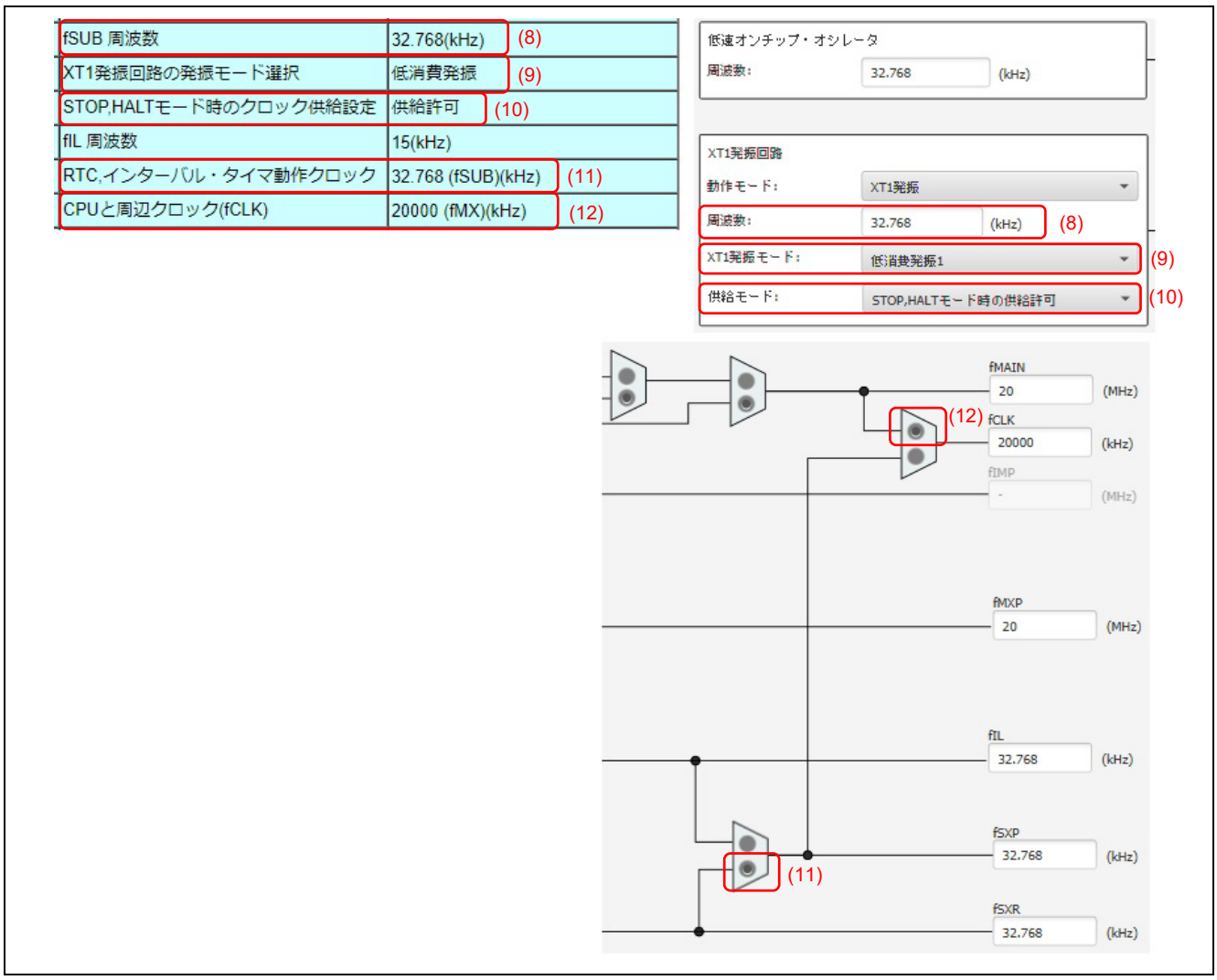

図 2.12　スマート・コンフィグレータのクロック設定 (2)

(4) [システム] タブを選択します。

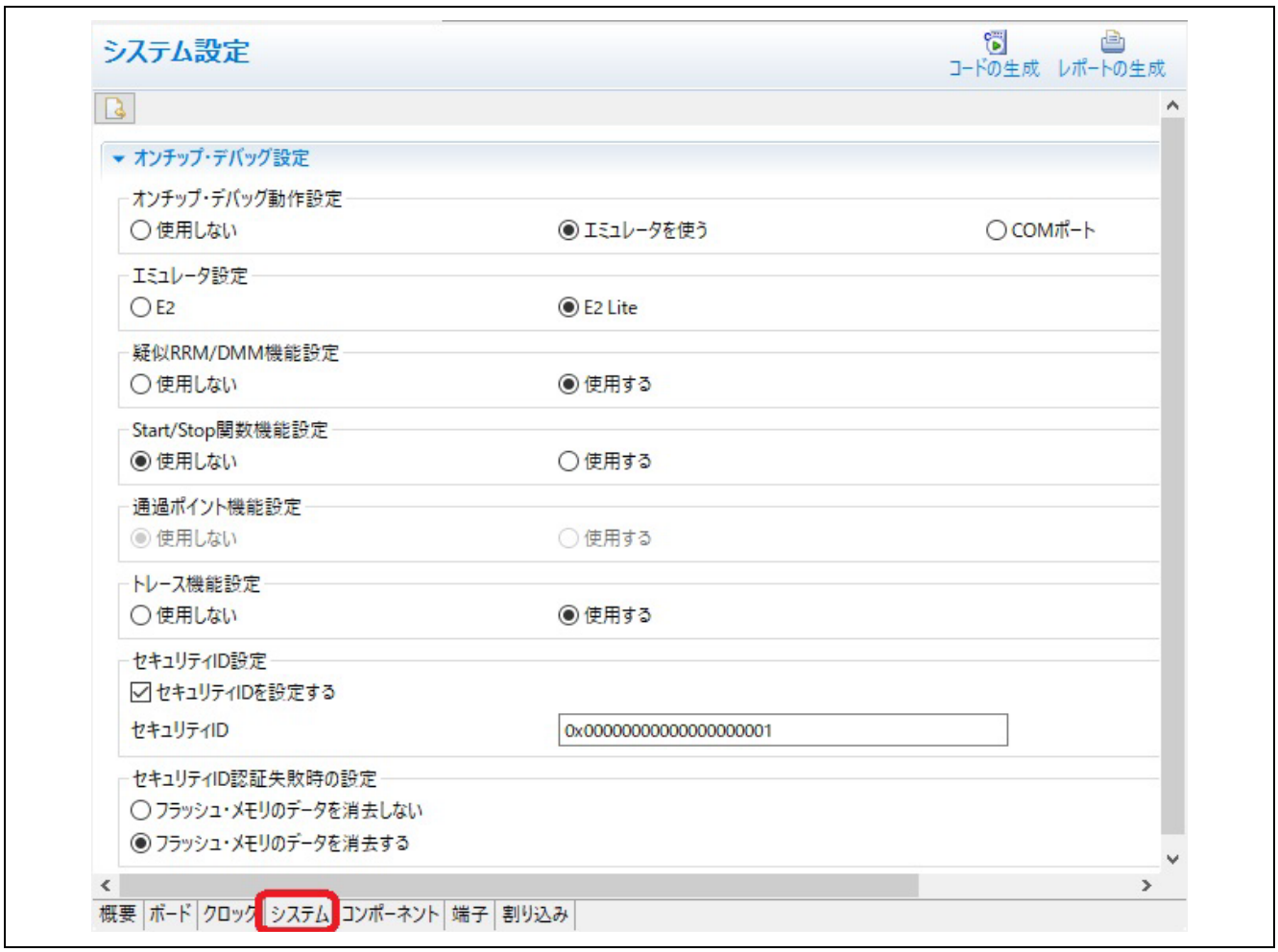

図 2.13　スマート・コンフィグレータのシステム設定画面

(5) レポートの Macro.html ファイルの [設定] と [状態] 欄の左側の設定をスマート・コンフィグレータの設定に反映します。

図 2.14　スマート・コンフィグレータのシステム設定

表 2.4　クロック発生回路の設定 (1)

	Code Generator		スマート・コンフィグレータ –[クロック]タブ	
	設定項目 （Macro.html の[マクロ]または[設定]）	設定 （Macro.html の[状態]）	設定項目	設定
(1)	動作モード設定	高速メイン・モード 2.7 (V)≦VDD≦5.5 (V)	動作モード	高速メイン・モード 2.7 (V)～5.5 (V)
(2)	EVDD 設定	2.7 (V)≦VDD≦5.5 (V)	EVDD 設定	2.7 V≦EVDD0≦5.5 V
(3)	fIH 動作	使用しない	高速オンチップ・オシレータ	チェックなし
(4)	fMX 動作	使用する	fMX クロックソースが選択されていることを確認する	
(5)	高速システム・クロック設定	X1 発振(fX)	X1 発振回路	チェックあり
(6)	fMX 周波数	20(MHz)	[X1]周波数	20
(7)	発振安定時間	6553.6 (2^17/fX)(μs)	発振安定時間	2^17/fx
(8)	HOCO 動作	使用しない	HOCO クロック	チェックなし
(9)	fSUB 周波数	32.768 (fSUB)(kHz)	[XT1]周波数	32.768
(10)	STOP, HALT モード時のクロック供給設定	供給許可	供給モード	STOP, HALT モード時の供給許可
(11)	RTC,インターバル・タイマ動作クロック	32.768 (fSUB)(kHz)	fSXR クロックソースが選択されていることを確認する	
(12)	CPU と周辺クロック (fCLK)	20000 (fMX)(kHz)	fMAIN クロックソースが選択されていることを確認する	

表 2.5　クロック発生回路の設定 (2)

	Code Generator		スマート・コンフィグレータ –[システム] タブ	
	設定項目（Macro.html の[マクロ]または[設定]）	設定（Macro.html の[状態]）	設定項目	設定
(13)	オンチップ・デバッグ動作設定	使用する	オンチップ・デバッグ動作設定	エミュレータを使う
(14)	セキュリティ ID 認証失敗時の設定	フラッシュ・メモリのデータを消去する	セキュリティ ID 認証失敗時の設定	フラッシュ・メモリのデータを消去する
(15)	疑似 RRM/DMM 機能設定	使用する	疑似 RRM/DMM 機能設定	使用する
(16)	Start/Stop 関数機能設定	使用しない	Start/Stop 関数機能設定	使用しない
(17)	通過ポイント機能設定	使用しない	通過ポイント機能設定	使用しない

2.5.3　タイマの設定

タイマを設定します。

(1)「RL78 スマート・コンフィグレータ ユーザーガイド: e² studio 編」の「4.4.2　コード生成コンポーネントの追加方法」の「a-1　[コンポーネントの追加] アイコンをクリック」を参照して、コンペアマッチタイマをプロジェクトのコンポーネントとして追加します。
[コンポーネントの追加] ダイアログボックスの [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] ページでは、以下のように、デフォルト名をリソースのコンフィグレーション名として使用します。

表 2.6　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

コンポーネントの種類	コンポーネント	リソース	コンフィグレーション名	動作モード
Code Generator	インターバル・タイマ	TAU0_1	Config_TAU0_1（デフォルト）	16 ビットカウントモード

(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を示す部分を表示します。

タイマ				使用する
	TAU0			使用する
		Channel1		
			チャネル 1	インターバル・タイマ
			動作モード設定	16ビット
			インターバル時間(16ビット)	100ms, (実際の値：100)
			カウント開始時にINTTM01割り込みを発生する	使用しない
			タイマ・チャネル1のカウント完了で割り込み発生(INTTM01)	使用する
			優先順位 (INTTM01)	低

図 2.15　Code Generator が出力したタイマのレポート

(3) (1)で作成したインターバル・タイマ TAU0_1 の設定画面を開きます。

図 2.16　スマート・コンフィグレータのインターバル・タイマ (TAU0_1) の設定画面

(4) Macro.html のタイマの設定内容を、スマート・コンフィグレータの TAU0 チャネル 1 の設定に反映します。

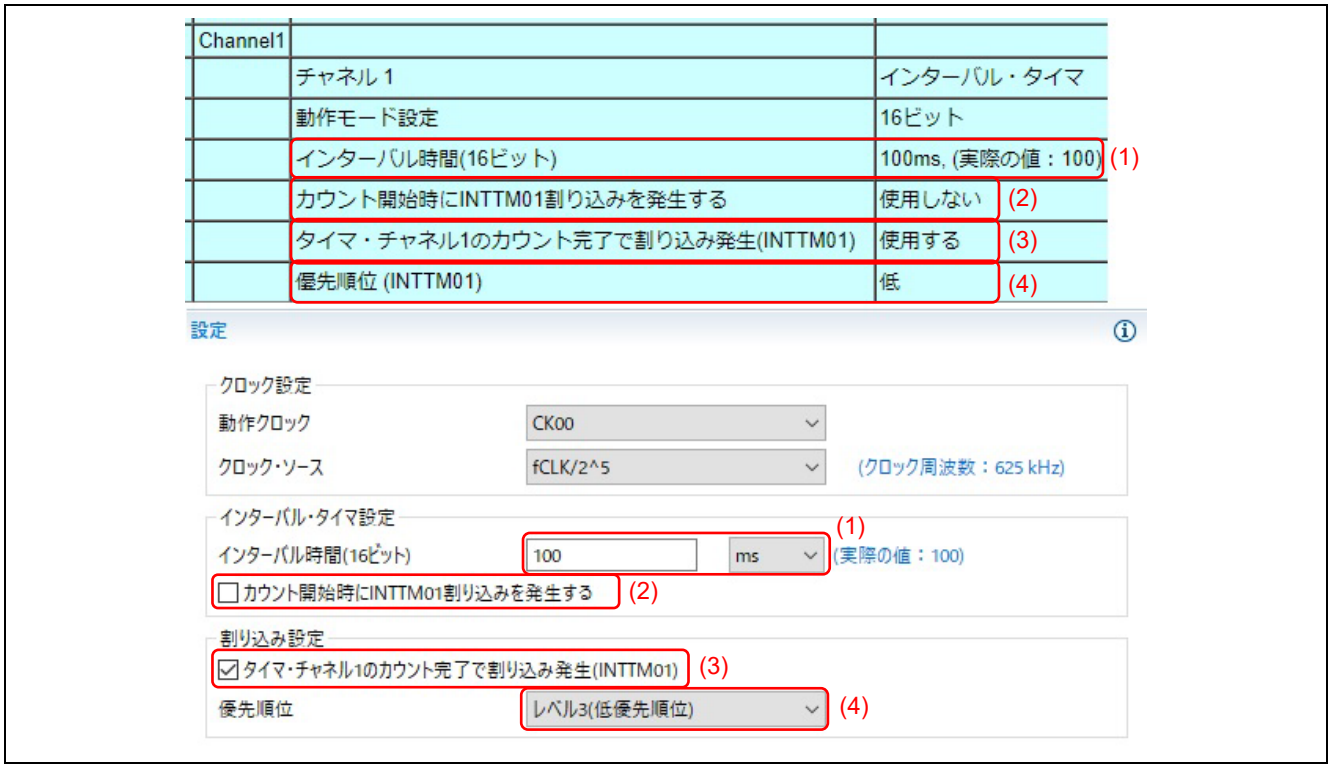

図 2.17　スマート・コンフィグレータのインターバル・タイマ (TAU0_1) の設定

表 2.7　タイマ（TAU0 チャネル 1）の設定

	Code Generator		スマート・コンフィグレータ	
	設定項目 （Macro.html の[マクロ]または[設定]）	設定 （Macro.html の[状態]）	設定項目	設定
(1)	インターバル時間(16 ビット)	100ms	インターバル時間(16 ビット)	100ms
(2)	カウント開始時に INTTM01 割り込みを発生する	使用しない	カウント開始時に INTTM01 割り込みを発生する	使用しない
(3)	タイマ・チャネル 1 のカウント完了で割り込み発生(INTTM01)	使用する	タイマ・チャネル 1 のカウント完了で割り込み発生(INTTM01)	使用する
(4)	優先順位 (INTTM01)	低	優先順位	レベル 3 (低優先順位)

2.5.4　その他の周辺機能の設定

ポートや A/D コンバータの設定については、「表 2.2　Code Generator のレポート機能が出力するレポートファイル」、「2.5.2　クロック発生回路の設定」および「2.5.3　タイマの設定」で説明されている手順を参照し、同様にスマート・コンフィグレータを設定してください。

2.5.5　コードの生成

すべ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 [コード生成] ボタン をクリックして、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　概要

Code Generator 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、Code Generator 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 2.18 にユーザ定義ソースコードを移植する手順を示します。

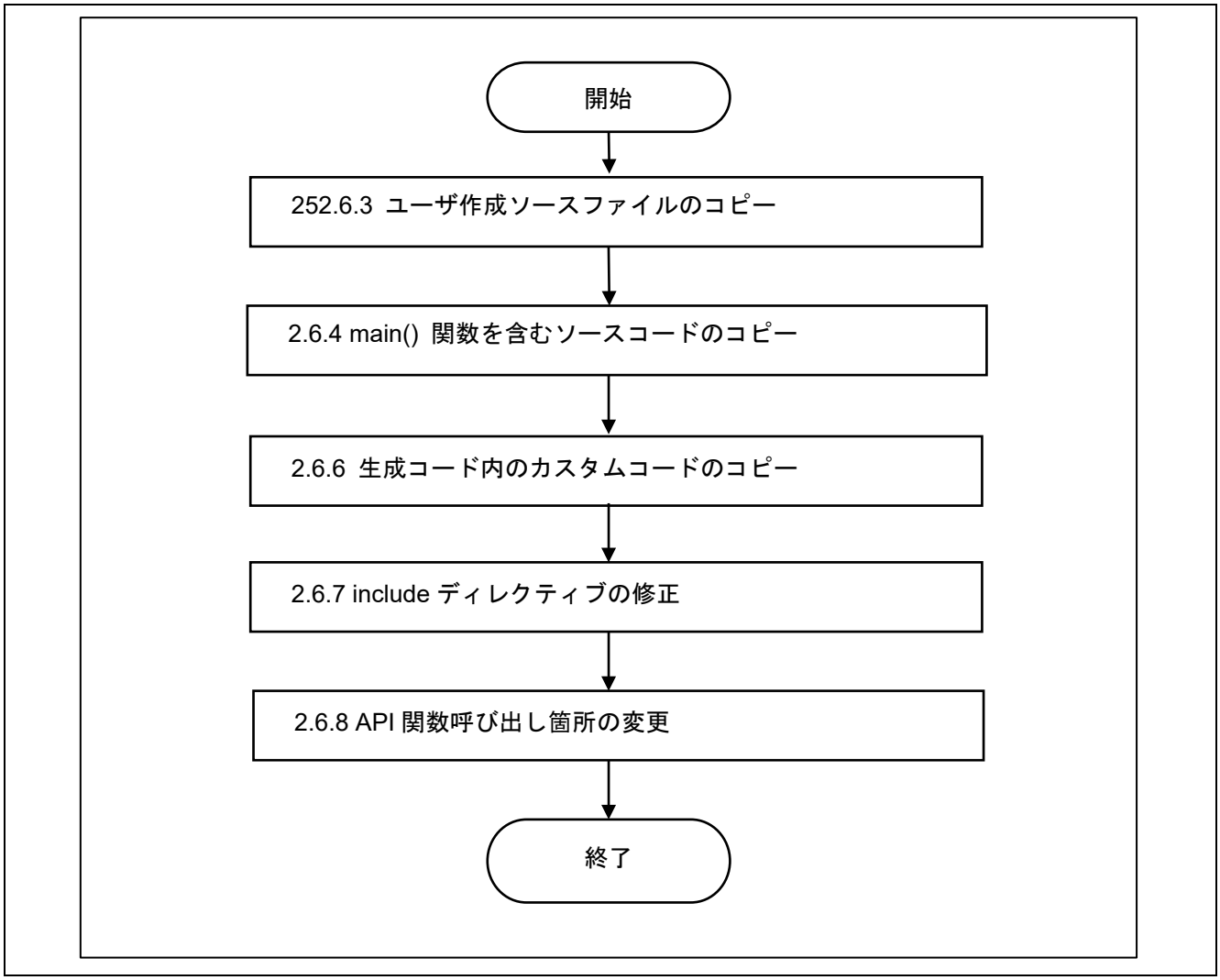

図 2.18　ユーザ定義ソースコードを移植する手順

2.6.2　ユーザ定義ソースコードを書き込む領域

Code Generator とスマート・コンフィグレータで生成されたファイルはユーザが自由にコードを書き込める領域を含みます。カスタムコード用の領域は以下のように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 */
```

これらのコメントでは、'xxxxxx' の部分はカスタムコードが追加される領域に依存します。例えば、インクルード宣言が書かれる場合には“include”になり、グローバル変数が定義される場合には 'global' になります。

これらのコメント間に位置する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に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

Code Generator 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移行元プロジェクトからコピーしてください。

下図に示すように、指定されたソースファイルとヘッダファイルを移行元プロジェクトの 'Tutorial' フォルダから移行先プロジェクトの 'src' フォルダにコピーします。

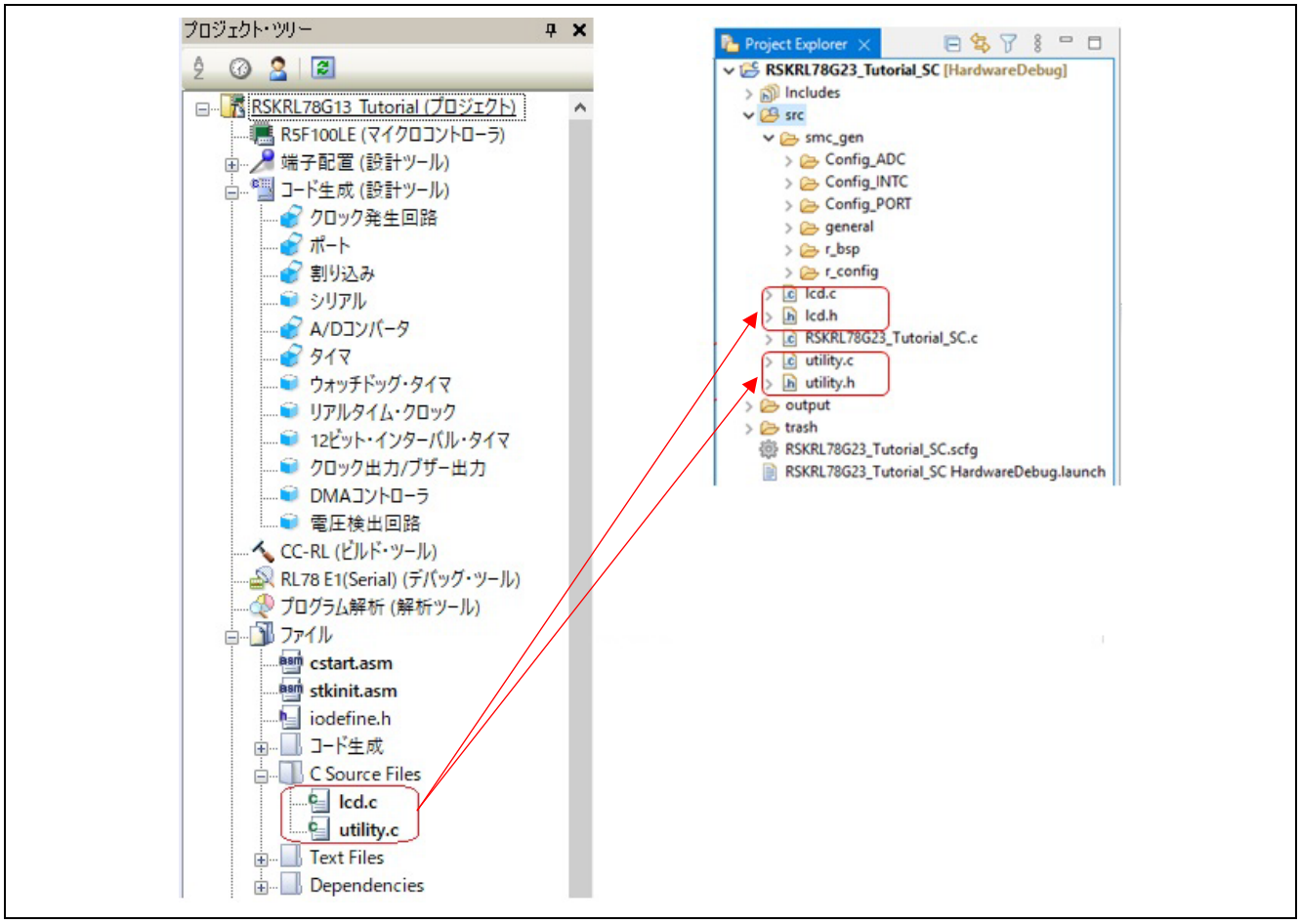

図 2.19　ユーザ作成ソースファイルのコピー

コピーされたソースファイルには Code Generator で生成した API 関数名が使われるため、これらの名前をスマート・コンフィグレータで生成した名前に修正する必要があります。さらに、インクルードされるヘッダファイル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」を参照してください。include ディレクティブの修正については、「2.6.7　include ディレクティブの修正」を参照してください。

'src' フォルダは、インクルードされるヘッダファイルを含むため、インクルード・ディレクトリに追加する必要があります。以下の手順でインクルード・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 [RSKRL78G23_Tutorial_SC] を右クリックし、[プロパティ: RSKRL78_Tutorial_SC] を開きます。左ペインの [C/C++ ビルド] 下の [設定] を選択します。右の設定画面で [ツール設定] タブを選択します。次に、[Compiler] 下の [ソース] を選択して、[インクルード・ファイルを検索するフォルダ] カテゴリの [追加] ボタンをクリックします。

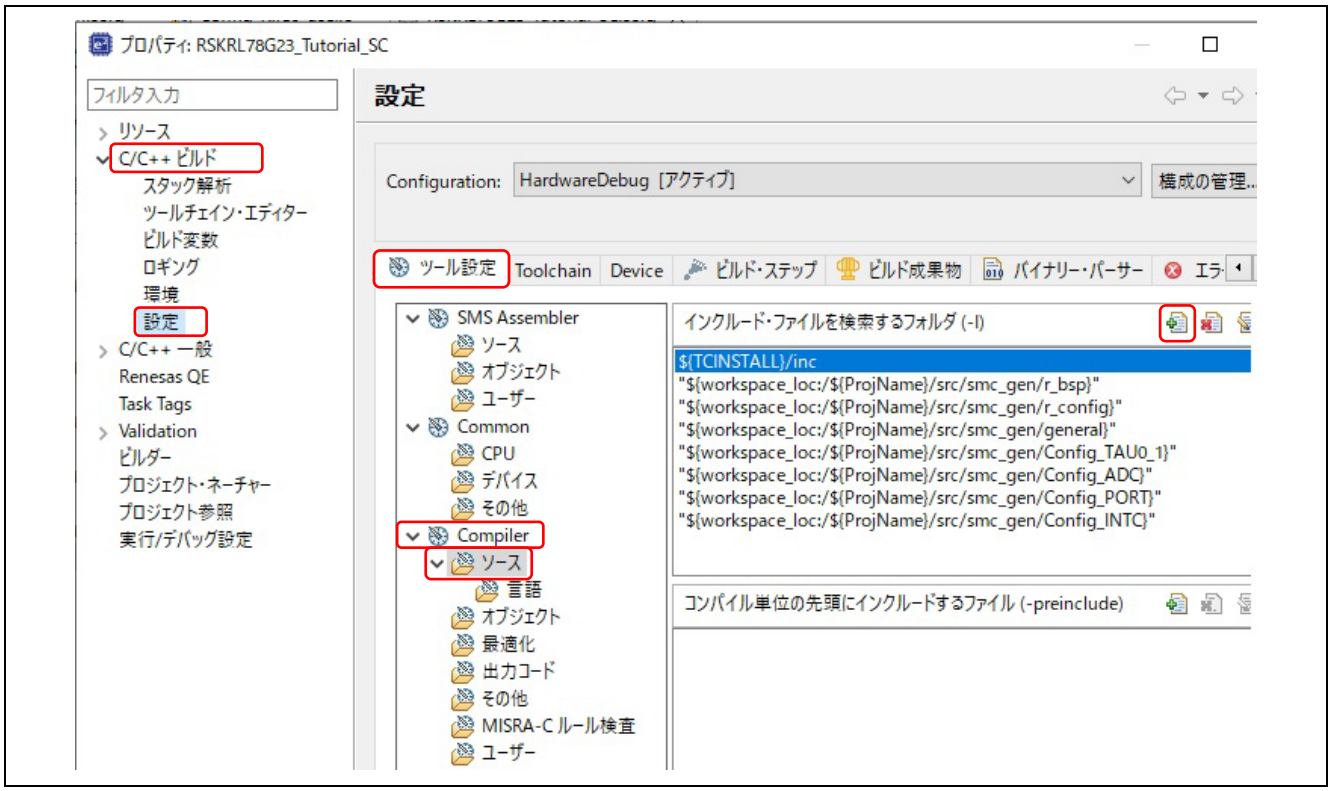

図 2.20　インクルード・ディレクトリの追加 (1)

(2) [ディレクトリー・パスの追加] ダイアログボックスの [ワークスペース] を選択します。[フォルダの選択] ダイアログボックスでは、インクルード・ディレクトリとして追加するフォルダ（例：src）を選択し、[OK] をクリックします。[ディレクトリー] に指定したフォルダが [ディレクトリー・パスの追加] ダイアログボックスに追加されたことを確認し、[OK] をクリックします。

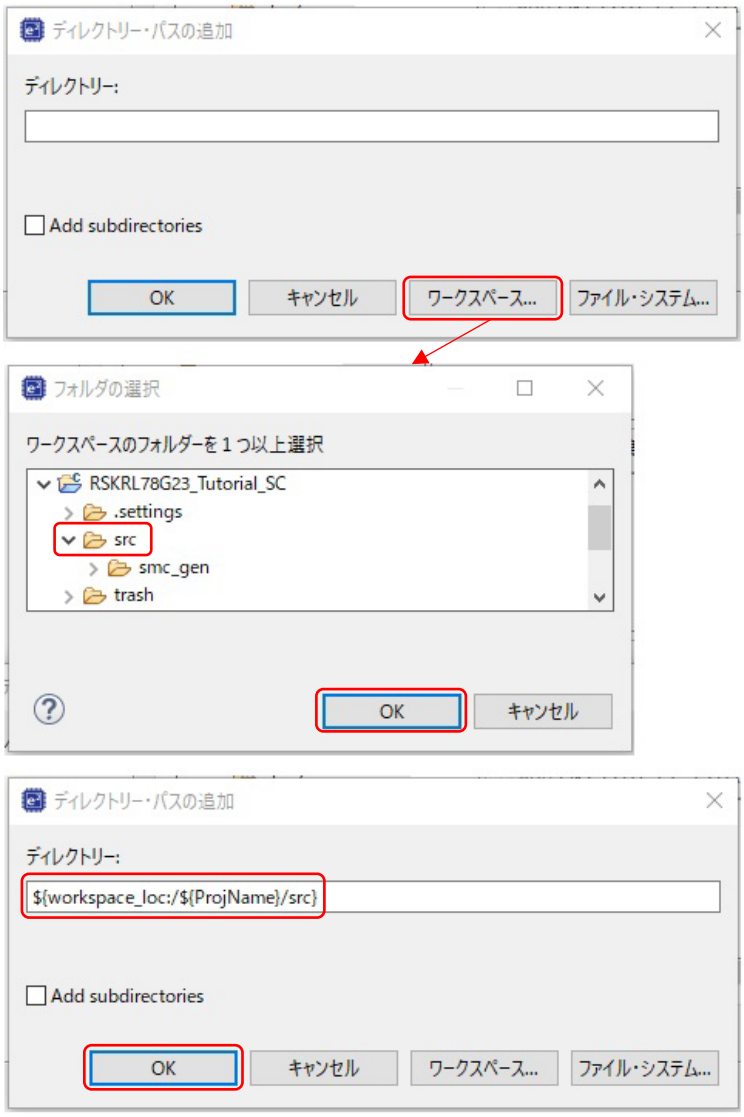

図 2.21　インクルード・ディレクトリの追加 (2)

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 'Tutorial' フォルダ内の 'r_main.c' ファイルに含まれます。'r_main.c' は Code Generator が生成するソースファイルのため、ユーザ定義ソースコードはコメント間の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では、移行先プロジェクト名が 'RSKRL78G23_Tutorial_SC' のため、main() 関数は 'RSKRL78G23_Tutorial_SC.c' に含まれます。'{ProjName}.c' 内のソースコードは、すべてユーザ定義コードになります。

'r_main.c' ファイルを開き、「2.6.2　ユーザ定義ソースコードを書き込む領域」に記載したコメント内に書かれているソースコードをすべてコピーしてください。

例として、include ディレクティブのコピー方法を以下に説明します。

インクルード・ファイルは、基本的には、「2.6.2　ユーザ定義ソースコードを書き込む領域」のコメント内に書かれているソースコードをコピーします。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（例：r_cg_userdefine.h）もコピーします。

新規プロジェクト作成時にこのソースファイルが生成された場合、'r_smc_entry.h' の include ディレクティブが自動的に '{ProjName}.c' に書き込まれます。

'r_main.c' に書かれた他のヘッダファイル（例：r_cg_macrodriver.h～r_cg_adc.h）をインクルードするためのプリプロセッサ・ディレクティブは、ヘッダファイルがユーザ定義ソースコードを含まない限りはコピーする必要がありません。

コピーされる文は以下で黄色にハイライトされています。

r_cg_main.c

```
/***********************************************************************
Includes
***********************************************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_cgc.h"
#include "r_cg_port.h"
#include "r_cg_intc.h"
#include "r_cg_adc.h"
#include "r_cg_timer.h"
/* Start user code for include. Do not edit comment generated here */

/* Following header file provides common defines for widely used items. */
#include "rskrl78g13def.h"
/* Following header file provides useful macros and function prototypes for
controlling the LCD interface. */
#include "lcd.h"
/* Header file with integer to string conversion functions */
#include "utility.h"

/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"
```

{ProjName}.c

```
#include "r_smc_entry.h"
/* Following header file provides useful macros and function prototypes for
controlling the LCD interface. */
#include "lcd.h"
/* Header file with integer to string conversion functions */
#include "utility.h"
#include "r_cg_userdefine.h"
```

ユーザ定義のプロトタイプ宣言および関数定義など、「2.6.2　ユーザ定義ソースコードを書き込む領域」に記載したコメント内に書かれたソースコードは、元の順序を保ったまま '{ProjName}.c' にコピーされます。

この例では、次のページで黄色にハイライトされたユーザ定義のプロトタイプ宣言および変数宣言をコピーしてください。

r_main.c

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

/* Define the RSK short name */
#define NICKNAME "RL78G13 "

/* Global initialised variable*/
int8_t ucStr[9]=" STATIC ";
/* Constant Data for replacement */
const int8_t ucReplace[] = "TESTTEST";
/* Global variable changed by pressing switches */
volatile int8_t gSwitchFlag = 0;

/* Static test function prototype */
void Statics_Test(void);

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

{ProjName}.c

```
int main(void);

/* Define the RSK short name */
#define NICKNAME "RL78G13 "

/* Global initialised variable*/
int8_t ucStr[9]=" STATIC ";
/* Constant Data for replacement */
const int8_t ucReplace[] = "TESTTEST";
/* Global variable changed by pressing switches */
volatile int8_t gSwitchFlag = 0;

/* Static test function prototype */
void Statics_Test(void);
```

以下で黄色にハイライトされた関数呼び出しと main() 関数の他のコードをコピーします。'- 中略 –' として省略している箇所もすべてコピーする必要があります。'R_MAIN_UserInit()' のように、Code Generator が生成する関数で、この関数内にユーザ定義コードが追加されていない場合は、コピーする必要はありません。

r_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 LCD module. */
    InitialiseDisplay();

    /* Display information on the debug LCD. */
    DisplayString(LCD_LINE1, (int8_t*)"Renesas");
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);

    /* Flash the user LEDs for some time or until a push button is pressed. */
    FlashLEDs();

    /* Flash the user LEDs at a rate set by the user potentiometer (ADC) using
       interrupts. */
    TimerADC();

    /* Demonstration of initialised variables. Use this function with the
       debugger.*/
    Statics_Test();

    /* Halt program in an infinite while loop */
    while (1U)
    {
        ;
    }

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

{ProjName}.c

```
int main(void)
{
    EI();
    /* Initialise the LCD module. */
    InitialiseDisplay();

    /* Display information on the debug LCD. */
    DisplayString(LCD_LINE1, (int8_t*)"Renesas");
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);

    /* Flash the user LEDs for some time or until a push button is pressed. */
    FlashLEDs();

    /* Flash the user LEDs at a rate set by the user potentiometer (ADC) using
       interrupts. */
    TimerADC();

    /* Demonstration of initialised variables. Use this function with the
       debugger.*/
    Statics_Test();
    return 0;
}
```

次のページで黄色にハイライトされた関数呼び出しとプライベート関数の他のコードをコピーします。'- 中略 –' として省略している箇所もすべてコピーする必要があります。'R_MAIN_UserInit()' のように、Code Generator が生成する関数で、この関数内にユーザ定義コードが追加されていない場合は、コピーする必要はありません。

r_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 */

    EI();

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

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

void Statics_Test(void)
{
    /* Declare loop count variable */
    uint8_t uiCount;
    /* Declare string variable to hold the string to be copied */
    uint8_t ucStr[] = "STATIC \0";
    /* Declare variable buffer to store the copied string */
    uint8_t ucReplace[] = "TESTTEST\0";

    /* Declare a delay count variable */
    uint32_t ulDelay;

    /* Write ucStr varaible, "STATIC" to LCD */
    DisplayString(LCD_LINE2, (int8_t*)ucStr);

    /* Begin for loop which writes one letter of ucReplace to the LCD at a time
       The nested while loops generate the delay bewteen each letter change */
    for (uiCount=0; uiCount<8; uiCount++)
    {
        /* Replace letter number uiCount of ucStr from ucReplace */
        ucStr[uiCount] = ucReplace[uiCount];

        /* Display the character on the debug LCD */
        DisplayString(LCD_LINE2, (int8_t*)ucStr);

        /* LED Flashing Delay */
        for(ulDelay=0; ulDelay<700000; ulDelay++)
        {
            /* Delay */
        }
    }

    /* Clear LCD Display */
    ucStr[uiCount] = '\0';

    /* Write MCU nickname to LCD again */
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);
}
/***********************************************************************
End of function Statics_Test
***********************************************************************/

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

{ProjName}.c

```
void Statics_Test(void)
{
    /* Declare loop count variable */
    uint8_t uiCount;
    /* Declare string variable to hold the string to be copied */
    uint8_t ucStr[] = "STATIC \0";
    /* Declare variable buffer to store the copied string */
    uint8_t ucReplace[] = "TESTTEST\0";

    /* Declare a delay count variable */
    uint32_t ulDelay;

    /* Write ucStr varaible, "STATIC" to LCD */
    DisplayString(LCD_LINE2, (int8_t*)ucStr);

    /* Begin for loop which writes one letter of ucReplace to the LCD at a time
       The nested while loops generate the delay bewteen each letter change */
    for (uiCount=0; uiCount<8; uiCount++)
    {
        /* Replace letter number uiCount of ucStr from ucReplace */
        ucStr[uiCount] = ucReplace[uiCount];

        /* Display the character on the debug LCD */
        DisplayString(LCD_LINE2, (int8_t*)ucStr);

        /* LED Flashing Delay */
        for(ulDelay=0; ulDelay<700000; ulDelay++)
        {
            /* Delay */
        }
    }

    /* Clear LCD Display */
    ucStr[uiCount] = '\0';

    /* Write MCU nickname to LCD again */
    DisplayString(LCD_LINE2, (int8_t*)NICKNAME);
}
/***********************************************************************************
End of function Statics_Test
***********************************************************************************/
```

2.6.5　Code Generator とスマート・コンフィグレータの生成コードの対応

Code Generator とスマート・コンフィグレータで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、適切なファイル間でのコピーが必要になります。

下表に移行元プロジェクトから移行先プロジェクトへのユーザ定義コードのコピーが必要となる主なファイルと出力フォルダを示します。

表 2.8　Code Generator とスマート・コンフィグレータの生成コードの対応

Code Generator		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	
Tutorial	r_main.c	src	{ProjName}.c	main() 関数を含むファイル。
Tutorial	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。
Tutorial	r_cg_xxx.c	src\smc_gen\Config_XXX	Config_XXX.c	周辺機能の初期化や動作に必要なソースファイル。スマート・コンフィグレータでは、各リソースに 1 つのファイルが出力されます。
	r_cg_xxx_user.c	src\smc_genConfig_XXX	Config_XXX_user.c	周辺機能の初期化後に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。スマート・コンフィグレータでは、各リソースに 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 レジスタの設定に必要なマクロ定義を含むヘッダファイル。これらのファイルは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。

注：'xxx'と'XXX'は、周辺機能の名前を表しています。

RSKRL78G13_Tutorial サンプルコードで、カスタムコードの移行が必要なファイルについて、Code Generator 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以下に示します。下表の例では、グレーの背景のファイルはカスタムコードの移行が不要です。他のファイルは移行が必要なカスタムコードを含んでいます。

表 2.9　RSKRL78G13_Tutorial サンプルコードでカスタムコードの移行が必要なファイル

周辺機能	Code Generator		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() 関数を含むファイル	Tutorial	r_main.c	src	{ProjName}.c
一般的な設定	Tutorial	r_cg_userdefine.h	src¥smc_gen¥general	r_cg_userdefine.h
割り込み	Tutorial	r_cg_intc.c	src¥smc_gen¥Config_INTC	Config_INTC.c
		r_cg_intc_user.c	src¥smc_gen¥Config_INTC	Config_INTC_user.c
		r_cg_intc.h	src¥smc_gen¥general	r_cg_intc.h
			src¥smc_gen¥Config_INTC	Config_INTC.h
ポート	Tutorial	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
タイマ	Tutorial	r_cg_timer.c	src¥smc_gen¥Config_TAU0_1	Config_TAU0_1.c
			src¥smc_gen¥general	r_cg_tau_common.c
		r_cg_timer_user.c	src¥smc_gen¥Config_TAU0_1	Config_TAU0_1_user.c
		r_cg_timer.h	src¥smc_gen¥general	r_cg_tau_common.h
			src¥smc_gen¥general	r_cg_tau.h
			src¥smc_gen¥Config_TAU0_1	Config_TAU0_1.h
A/D コンバータ	Tutorial	r_cg_adc.c	src¥smc_gen¥Config_ADC	r_cg_ad_common.c
			src¥smc_gen¥Config_ADC	Config_ADC.c
		r_cg_adc_user.c	src¥smc_gen¥Config_ADC	Config_ADC_user.c
		r_cg_adc.h	src¥smc_gen¥general	r_cg_ad_common.h
			src¥smc_gen¥general	r_cg_ad.h
			src¥smc_gen¥Config_ADC	Config_ADC.h

2.6.6　生成コード内のカスタムコードのコピー

タイマ（インターバル・タイマとして使用）を例として、表 2.9 の対応に従って、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方法を以下に説明します。

まず、以下で黄色にハイライトされたタイマ用のカスタムコードを 'r_cg_timer.h' から'Config_TAU0_1.h' にコピーします。

r_cg_timer.h

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

/* Declare TimerADC function prototype */
void TimerADC(void);

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

Config_TAU0_1.h

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

/* Declare TimerADC function prototype */
void TimerADC(void);

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

その後、以下で黄色にハイライトされたタイマ用のカスタムコードを 'r_cg_timer_user.cc' から 'Config_TAU0_1_user.c' にコピーします。

r_cg_timer_user.c

```
/* Start user code for include. Do not edit comment generated here */
#include "r_cg_adc.h"
/* rskrl78g13def.h provides common defines for widely used items. */
#include "rskrl78g13def.h"
/* Following header file provides function prototypes for LCD controlling
functions & macro defines */
#include "lcd.h"
/* Header file with integer to string conversion functions */
#include "utility.h"
/* End user code. Do not edit comment generated here */

/* Start user code for global. Do not edit comment generated here */
/* Declare a variable for storing the ADC value */
volatile uint16_t gTimerADCperiod = 0;
/* End user code. Do not edit comment generated here */ -

static void __near r_tau0_channel1_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    /* Toggle user LEDs */
    LED0 = ~LED0;
    LED1 = ~LED1;
    LED2 = ~LED2;
    LED3 = ~LED3;

    /* Store the ADC value into the lower 12 bits of the variable */
    gTimerADCperiod = ADCR >> 6;

    /* Ensure that the timer period is never set below 0x75 */
    if(gTimerADCperiod < 0x0075)
    {
        gTimerADCperiod = 0x0075;
    }

    /* Update timer period with respect to adc value */
    TDR01 = gTimerADCperiod * 58;

    /* Clear TM01 interrupt flag */
    TMIF01 = 0;
    /* End user code. Do not edit comment generated here */
}

/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************
* Function Name: TimerADC
* Description  : Uses the ADC to change the duty of the timer, used to flash
the LEDs.
* Arguments    : None
* Return Value : None
***********************************************************************/
void TimerADC(void)
{
    /* Start ADC operations */
    R_ADC_Start();
    /* Start timer TM01 operations */
    R_TAU0_Channel1_Start();
}
/***********************************************************************
End of function TimerADC
***********************************************************************/
/* End user code. Do not edit comment generated here */
```

Config_TAU0_1_user.c

```
  /* Start user code for include. Do not edit comment generated here */
  #include "Config_ADC.h"
  /* rskrl78g13def.h provides common defines for widely used items. */
  #include "rskrl78g13def.h"
  /* Following header file provides function prototypes for LCD controlling
functions & macro defines */
  #include "lcd.h"
  /* Header file with integer to string conversion functions */
  #include "utility.h"
  /* End user code. Do not edit comment generated here */

  /* Start user code for global. Do not edit comment generated here */
  /* Declare a variable for storing the ADC value */
  volatile uint16_t gTimerADCperiod = 0;
  /* End user code. Do not edit comment generated here */
  static void __near r_Config_TAU0_1_interrupt(void)
  {
      /* Start user code for r_Config_TAU0_1_interrupt. Do not edit comment
generated here */
     /* Toggle user LEDs */
     LED0 = ~LED0;
     LED1 = ~LED1;
     LED2 = ~LED2;
     LED3 = ~LED3;

       /* Store the ADC value into the lower 12 bits of the variable */
     gTimerADCperiod = ADCR >> 6;

       /* Ensure that the timer period is never set below 0x75 */
     if(gTimerADCperiod < 0x0075)
     {
        gTimerADCperiod = 0x0075;
     }

       /* Update timer period with respect to adc value */
       TDR01 = gTimerADCperiod * 58;

       /* Clear TM01 interrupt flag */
       TMIF01 = 0;
       /* End user code. Do not edit comment generated here */
  }
  /* Start user code for adding. Do not edit comment generated here */

  /***********************************************************************
**
  * Function Name: TimerADC
  * Description  : Uses the ADC to change the duty of the timer, used to flash the
LEDs.
  * Arguments    : None
  * Return Value : None
  ***********************************************************************
*/
  void TimerADC(void)
  {
       /* Start ADC operations */
       R_Config_ADC_Start();

       /* Start timer TM01 operations */
       R_Config_TAU0_1_Start();
```

水色でハイライトされた 'R_Config_ADC_Start' は、Code Generator 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。この修正の詳細な手順は「2.6.8　API 関数呼び出し箇所の変更」で説明されています。

2.6.7　include ディレクティブの修正

Code Generator は、主に周辺機能ごとに 'r_cg_xxx.h' という名前のファイルを出力します。スマート・コンフィグレータは、複数のヘッダファイルを周辺機能共通の 'r_cg_xxx.h' およびコンポーネントのリソースの 'Config_XXX.h' に分割して出力します。したがって、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'と'XXX'は、周辺機能の名前を表しています。）

例えば、「2.6.6　生成コード内のカスタムコードのコピー」でコピーされた 'r_cg_adc.h' をインクルードしているファイルを検索し、適切な include ディレクティブに修正します。

'#include r_cg_adc.h' を含むファイルを移行元プロジェクトで検索した結果は以下のようになります。

```
Tutorial
├──　r_systeminit.c
├──　r_main.c
├──　r_cg_adc.c
├──　r_cg_adc_user.c
└──　r_cg_timer_user.c
```

'#include r_cg_adc.h' を含むファイルのうち、'r_main.c'（移行先プロジェクトの{ProjName}.c）および 'r_cg_timer_user.c'（移行先プロジェクトの Config_TAU0_1_user.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をインクルードしているため、include ディレクティブを修正する必要はありません。

'r_cg.timer_user.c'（移行先プロジェクトの Config_TAU0_1_user.c）に関しては、対応する移行先プロジェクトのソースファイルの include ディレクティブを修正する必要があります。

- ソースファイル（Config_TAU0_1_user.c）の場合
  移行先プロジェクトの 'Config_TAU0_1_user.c' を開き、ADC 関数をコールしている箇所を検索します。'Config_TAU0_1_user.c' では、以下の関数をコールします。
  ― R_Config_ADC_Start()

  プロトタイプ宣言が 'Config_ADC.h' 内にあるため、このヘッダファイルをインクルードするようにディレクティブを修正します。

r_cg_timer_user.c（修正前）

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

#include "r_cg_adc.h"
```

Config_TAU0_1_user.c（修正後）

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

#include "Config_ADC.h"
```

2.6.8　API 関数呼び出し箇所の変更

「2.6.6　生成コード内のカスタムコードのコピー」でコピーしたカスタムコードの一部は、Code Generator の API 関数の呼び出し箇所を含みます。これらの API 関数名をスマート・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。

'r_cg_timer_user.c' 中のカスタムコードは、「2.6.6　生成コード内のカスタムコードのコピー」に従ってコピーされました。TimerADC() 関数の定義はファイルに含まれています。

例）

修正前の TimerADC() では、Code Generator が生成した 2 つの API 関数 R_ADC_Start() と R_TAU0_Channel1_Start() を呼び出します。これらの関数は、スマート・コンフィグレータが生成する関数では、R_Config_ADC_Start() と R_TAU0_1_Start() になるため（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使用した場合）、呼び出し箇所の API 関数名を変更する必要があります。

r_cg_timer_user.c（修正前）

```
void TimerADC(void)
{
    /* Start ADC operations */
    R_ADC_Start();

    /* Start timer TM01 operations */
    R_TAU0_Channel1_Start();
}
```

Config_TAU0_1_user.c（修正後）

```
void TimerADC(void)
{
    /* Start ADC operations */
    R_Config_ADC_Start();

    /* Start timer TM01 operations */
    R_Config_TAU0_1_Start();
}
```

表 2.10 に、Code Generator 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 関数名の対応を示します。表に従い、API 関数の呼び出し箇所を変更してください。

表 2.10 に示したスマート・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。ユーザがコンフィグレーション名を設定できるため、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、Web サイトを参照してください。
（https://www.renesas.com/smart-configurator）

表 2.10 Code Generator 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

	Code Generator		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	mcu_clocks.c	mcu_clock_setup
タイマ				
	r_cg_timer.c	R_TAU0_Create	Config_TAU0_1.c	R_Config_TAU0_1_Create
		R_TAU0_Channel1_Start		R_Config_TAU0_1_Start
		R_TAU0_Channel1_Stop		R_Config_TAU0_1_Stop
	r_cg_timer_user.c	–	Config_TAU0_1_user.c	R_Config_TAU0_1_Create_UserInit
		r_tau0_channel1_interrupt		r_tau0_channel1_interrupt
割り込み				
	r_cg_intc.c	R_INTC_Create	Config_INTC.c	R_Config_INTC_Create
		R_INTC1_Start		R_Config_INTC_INTP1_Start
		R_INTC1_Stop		R_Config_INTC_INTP1_Stop
		R_INTC2_Start		R_Config_INTC_INTP2_Start
		R_INTC2_Stop		R_Config_INTC_INTP2_Stop
		R_INTC4_Start		R_Config_INTC_INTP2_Start
		R_INTC4_Stop		R_Config_INTC_INTP2_Stop
	r_cg_intc_user.c	–	Config_INTC_user.c	R_Config_ICU_Create_UserInit
		r_intc1_interrupt		r_Config_INTC_intp1_interrupt
		r_intc2_interrupt		r_Config_INTC_intp2_interrupt
		r_intc4_interrupt		r_Config_INTC_intp4_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
A/D コンバータ				
	r_cg_adc.c	R_ADC_Create	Config_ADC.c	R_Config_ADC_Create
		R_ADC_Start		R_Config_ADC_Start
		R_ADC_Stop		R_Config_ADC_Stop
		R_ADC_Set_OperationOn		R_Config_ADC_Set_OperationOn
		R_ADC_Set_OperationOff		R_Config_ADC_Set_OperationOff
		R_ADC_Get_Result		R_Config_ADC_Get_Result_10bit
		–		R_Config_ADC_Set_SnoozeOn
		–		R_Config_ADC_Set_SnoozeOff
		–		R_Config_ADC_Set_ADChannel
		–		R_Config_ADC_Set_TestChannel
	r_cg_adc_user.c	–	Config_ADC_user.c	R_Config_ADC_Create_UserInit
		r_adc_ interrupt		r_Config_ADC_interrupt

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には、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。

「統合開発環境 e^2 studio 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ダウンロードしてください。

【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ダウンロードしてください。

【ユーザーズマニュアル：開発ツール】

CC-RL コンパイラ ユーザーズマニュアル (R20UT3123)

最新版をルネサスエレクトロニクスホームページからダウンロードしてください。

統合開発環境 e^2 studio ユーザーズマニュアル 入門ガイド (R20UT4819)

最新版をルネサスエレクトロニクスホームページからダウンロードしてください。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2022.04.05	ー	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. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

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

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

© 2022 Renesas Electronics Corporation. All rights reserved