User's Manual

32

RX130 グループ

Renesas Starter Kit
スマート・コンフィグレータ チュートリアルマニュアル (CS+)

ルネサス 32 ビットマイクロコントローラ
RX ファミリ/RX100 シリーズ

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00 2017.06

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。

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。

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、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

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. リザーブアドレスのアクセス禁止
   【注意】リザーブアドレス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。

このマニュアルの使い方

1. 目的と対象者

このマニュアルは、統合開発環境CS+およびスマート・コンフィグレータを使用してRSKプラットフォーム用プロジェクトを作成するための方法を理解していただくためのマニュアルです。様々な周辺装置を使用して、RSKプラットフォーム上のサンプルコードを設計するユーザを対象にしています。

このマニュアルは、段階的に CS+中のプロジェクトをロードし、デバッグする指示を含みますが、RSK プラットフォーム上のソフトウェア開発のガイドではありません。

このマニュアルを使用する場合、注意事項を十分確認の上、使用してください。注意事項は、各章の本文中、各章の最後、注意事項の章に記載しています。

本マニュアル中のスクリーンショットと実際に表示される画面が一部異なる場合があります。読み進めるにあたって問題はありません。

改訂記録は旧版の記載内容に対して訂正または追加した主な箇所をまとめたものです。改訂内容すべてを記録したものではありません。詳細は、このマニュアルの本文でご確認ください。

RSKRX130-512KB では次のドキュメントを用意しています。ドキュメントは最新版を使用してください。最新版はルネサスエレクトロニクスのホームページに掲載されています。

ドキュメントの種類	記載内容	資料名	資料番号
ユーザーズマニュアル	RSK ハードウェア仕様の説明	RSKRX130-512KB ユーザーズマニュアル	R20UT3921JG
チュートリアルマニュアル	RSK および開発環境のセットアップ方法とデバッギング方法の説明	RSKRX130-512KB チュートリアルマニュアル	R20UT3922JG
クイックスタートガイド	A4 紙一枚の簡単なセットアップガイド	RSKRX130-512KB クイックスタートガイド	R20UT3923JG
スマート・コンフィグレータ チュートリアルマニュアル	スマート・コンフィグレータの使用方法の説明	RSKRX130-512KB スマート・コンフィグレータ チュートリアルマニュアル	R20UT3924JG (本マニュアル)
回路図	CPU ボードの回路図	RSKRX130-512KB CPU ボード回路図	R20UT3920EG
ユーザーズマニュアル ハードウェア編	ハードウェアの仕様（ピン配置、メモリマップ、周辺機能の仕様、電気的特性、タイミング）と動作説明	RX130 グループ ユーザーズマニュアル ハードウェア編	R01UH0560JJ

2. 略語および略称の説明

略語／略称	英語名	備考
ADC	Analog-to-Digital Converter	A/D コンバータ
API	Application Programming Interface	アプリケーションプログラムインタフェース
bps	bits per second	転送速度を表す単位、ビット/秒
CMT	Compare Match Timer	コンペアマッチタイマ
COM	COMmunications port referring to PC serial port	シリアル通信方式のインタフェース
CPU	Central Processing Unit	中央処理装置
DVD	Digital Versatile Disc	ディジタルヴァーサタイルディスク
E1/E2 Lite	Renesas On-chip Debugging Emulator	ルネサスオンチップデバッギングエミュレータ
GUI	Graphical User Interface	グラフィカルユーザインタフェース
IDE	Integrated Development Environment	統合開発環境
IRQ	Interrupt Request	割り込み要求
LCD	Liquid Crystal Display	液晶ディスプレイ
LED	Light Emitting Diode	発光ダイオード
LSB	Least Significant Bit	最下位ビット
LVD	Low Voltage Detect	電圧検出回路
MCU	Micro-controller Unit	マイクロコントローラユニット
MSB	Most Significant Bit	最上位ビット
PC	Personal Computer	パーソナルコンピュータ
PLL	Phase-locked Loop	位相同期回路
Pmod™	-	Pmod™は Digilent Inc.の商標です。Pmod™インタフェース明細は Digilent Inc.の所有物です。Pmod™明細については Digilent Inc.の Pmod™ License Agreement ページを参照してください。
RAM	Random Access Memory	ランダムアクセスメモリ
ROM	Read Only Memory	リードオンリーメモリ
RSK	Renesas Starter Kit	ルネサススタータキット
RTC	Real Time Clock	リアルタイムクロック
SAU	Serial Array Unit	シリアルアレイユニット
SCI	Serial Communications Interface	シリアルコミュニケーションインタフェース
SPI	Serial Peripheral Interface	シリアルペリフェラルインタフェース
TAU	Timer Array Unit	タイマアレイユニット
TFT	Thin Film Transistor	薄膜トランジスタ
TPU	Timer Pulse Unit	タイマパルスユニット
UART	Universal Asynchronous Receiver/Transmitter	調歩同期式シリアルインタフェース
USB	Universal Serial Bus	シリアルバス規格の一種
WDT	Watchdog Timer	ウォッチドッグタイマ

目次

1. 概要

1.1 目的

本 RSK はルネサスマイクロコントローラ用の評価ツールです。本マニュアルは、統合開発環境 CS+およびスマート・コンフィグレータを使用してプロジェクトを作成する方法について説明しています。

1.2 特徴

本 RSK は以下の特徴を含みます：

- スマート・コンフィグレータを使用してのコード生成
- CS+によるプロジェクト作成およびビルド
- スイッチ、LED、ポテンショメータ等のユーザ回路

CPU ボードはマイクロコントローラの動作に必要な回路を全て備えています。

2. はじめに

本マニュアルは統合開発環境 CS+およびスマート・コンフィグレータを使用してプロジェクトを作成する方法についてチュートリアル形式で説明しています。チュートリアルでは以下の項目について説明しています。

- プロジェクトの作成
- スマート・コンフィグレータを使用したコード生成について
- カスタムコードの統合
- CS+プロジェクトのビルド

プロジェクトジェネレータは、選択可能な 3 種類のビルドコンフィグレーションを持つチュートリアルプロジェクトを作成します。

- ‘DefaultBuild’はデバッガのサポートおよび最適化レベル 2 を含むプロジェクトを構築します。
- ‘Debug’はデバッガのサポートを含むプロジェクトを構築します。最適化レベルは 0 に設定されています。
- ‘Release’は最適化された製品リリース用に適したコードを構築します。最適化レベルは 2 に、デバッグ情報を出力しないように設定されています。

本チュートリアルの使用例はクイックスタートガイドに記載のインストールが完了していることを前提としています。

チュートリアルは RSK の使用方法の説明を目的とするものであり、CS+、コンパイラまたは E2 エミュレータ Lite の入門書ではありません。これらに関する詳細情報は各関連マニュアルを参照してください。

3. プロジェクトの作成

3.1 はじめに

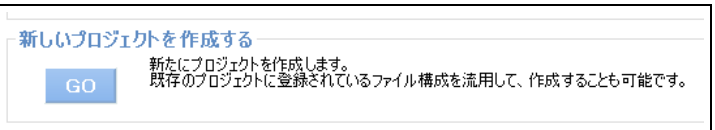
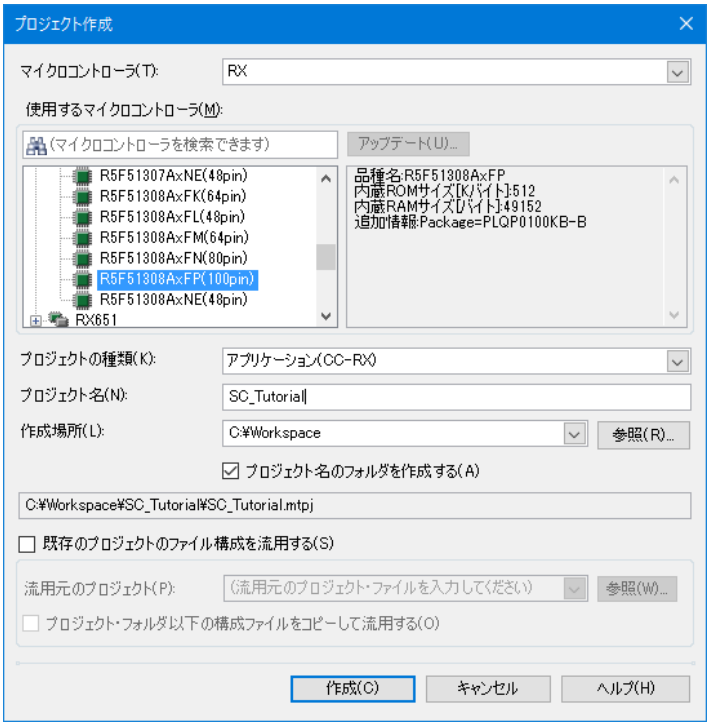
この章では RX130 マイクロコントローラのための新しい C ソースプロジェクトを作成するのに必要な手順をガイドします。
このプロジェクト作成の手順はマイクロコントローラ特有のプロジェクトを作成し、ソースをデバッグするのに必要です。

3.2 プロジェクトの作成

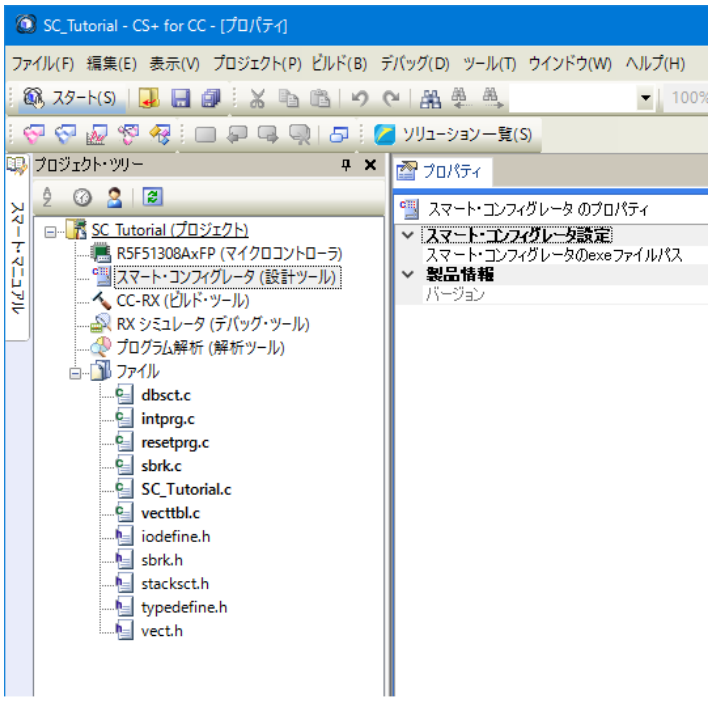
CS+起動方法は以下の通りです。
Windows™ 7: スタートメニュー > すべてのプログラム > Renesas Electronics CS+ >
CS+ for CC (RL78,RX,RH850)

Windows™ 8.1/8: ⓫をクリックして[アプリ]ビューを表示 > ‘CS+ for CC (RL78,RX,RH850)’アイコン

Windows™ 10: スタートメニュー > すべてのアプリ> Renesas Electronics CS+ > CS+ for CC (RL78,RX,RH850)

• スタートパネルが表示されたら、‘新しいプロジェクトを作成する’の<GO>をクリックしてください。	
• プロジェクト作成ダイアログのマイクロコントローラプルダウンメニューから‘RX’を選択してください。 • マイクロコントローラー覧で下にスクロールします。‘RX130’の‘+’を展開してR5F51308AxFP(100pin)を選択してください。 • ‘プロジェクトの種類(K):’のプルダウンから‘空のアプリケーション(CC-RX)’を選択してください。 • ‘プロジェクト名(N):’と‘作成場所(L)’を指定し、<作成>をクリックしてください。 注：右のスクリーンショットのプロジェクト名および作成場所は、本チュートリアル用のプロジェクト設定例です。 • ‘フォルダが存在しません。作成しますか？’のダイアログが表示された場合、‘はい(Y)’をクリックしてください。	

- CS+は標準的なプロジェクト・ツリーを持つプロジェクトを生成します。‘スマート・コンフィグレータ(設計ツール)’がプロジェクト・ツリー上に表示されます。



4. CS+ スマート・コンフィグレータによるコード生成

4.1 はじめに

スマート・コンフィグレータは C ソースコード生成とマイクロコントローラの生成のための GUI ツールです。スマート・コンフィグレータは直感的な GUI を使用することで、様々なマイクロコントローラの周辺機能や動作に必要なパラメータを設定することができ、開発工数の大幅な削減が可能です。

本書の手順を踏むことで、ユーザは SC_Tutorial と呼ばれる CS+プロジェクトを作成できます。完成済みのプロジェクトは DVD に収録されており、クイックスタートガイドの手順に従えば、完成済みのプロジェクトを使用できます。本書はオリジナルの CS+プロジェクトを作成し、スマート・コンフィグレータを使用したいユーザのためのチュートリアルマニュアルです。

ユーザがプロジェクトを設定すると、スマート・コンフィグレータによって選択されたソフトウェアコンポーネントのフォルダ、general フォルダ、r_bsp フォルダ、r_config フォルダ、そして r_pincfg フォルダにコードが生成されます。ソフトウェアコンポーネントフォルダのファイルには、「Config_xxx.h」、「Config_xxx.c」、「Config_xxx_user.c」と名付けられます。例えば A/D コンバータの場合、周辺を表す xxx は’S12AD’となります。これらのコードはユーザの要求を満たすために、カスタムコードを自由に加えることができます。カスタムコードを加える場合、以下に示すコメント文の間にカスタムコードを加えてください。

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

スマート・コンフィグレータの GUI 上で設定した内容を変更したい場合等、再度コード生成を行う場合にスマート・コンフィグレータはこれらのコメント文を見つけて、コメント文の間に加えられたカスタムコードを保護します。

SC_Tutorial プロジェクトは、スイッチによる割り込み、A/D モジュール、8 ビットタイマ（TMR）、コンペアマッチタイマ（CMT）、シリアルコミュニケーションインタフェース（SCI）を使用し、A/D 変換値をターミナルソフトや LCD ディスプレイに表示します。

セクション 4.2 以降スマート・コンフィグレータのユーザインタフェースと各周辺機能ダイアログについて説明します。5 章では生成されたコードの CS+プロジェクトへの組み込み、カスタムコードの追加方法、チュートリアルコードの構造について説明します。

4.2 スマート・コンフィグレータを使用したプロジェクト設定 – 概要ページ

このセクションでは、スマート・コンフィグレータの簡単な操作方法を示しています。各操作の詳細につきましては、スマート・コンフィグレータマニュアルを参照ください。
最新版は、https://www.renesas.com/smart-configurator からダウンロードしてください。

“スマート・コンフィグレータ(設計ツール)”をプロジェクト・ツリー上でダブルクリックすることにより、スマート・コンフィグレータが起動します。
スマート・コンフィグレータ起動後、**図 4-1** に示すような画面が表示されます。

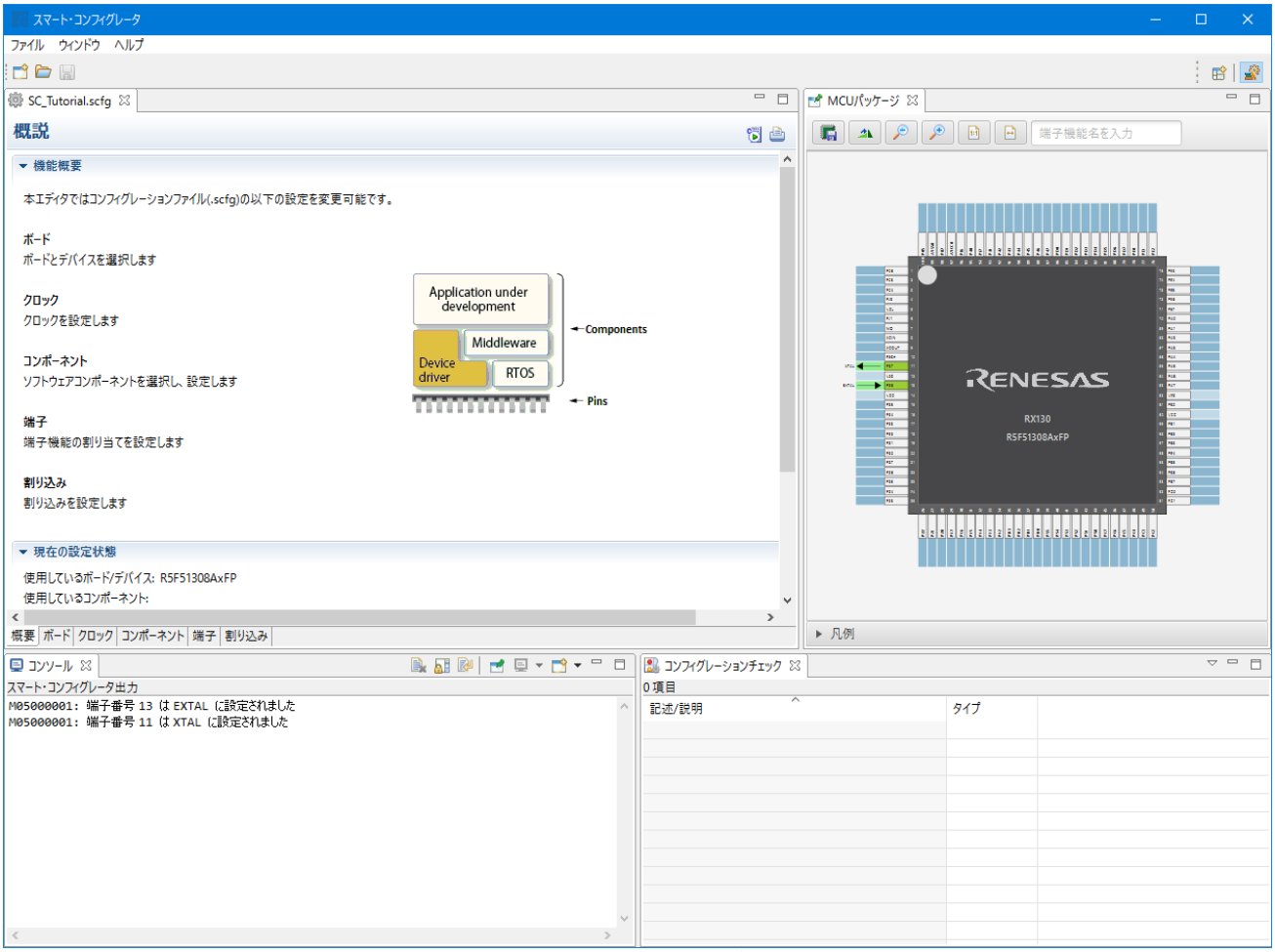


図 4-1: 概要ページ

スマート・コンフィグレータは MCU 設定を GUI で操作できます。ユーザが必要な設定を完了し、<コードを生成する>ボタンをクリックすると、設定した内容のコードが生成されます。

4.3 クロック設定ページ

クロック設定ページでは、選択したデバイスのクロックを設定します。クロック、周波数、PLL 設定、およびクロック分周器の設定をクロックに設定できます。

4.3.1 クロック設定

スマート・コンフィグレータのクロックタブを**図 4-2** に示します。‘クロック’タブをクリックしてください。図のように設定値を入力してください。チュートリアルでは、クロック発振源に本 CPU ボード搭載の 8MHz 水晶発振子を使用します。メイン・システム・クロック(fMAIN)に‘PLL 回路’を選択します。

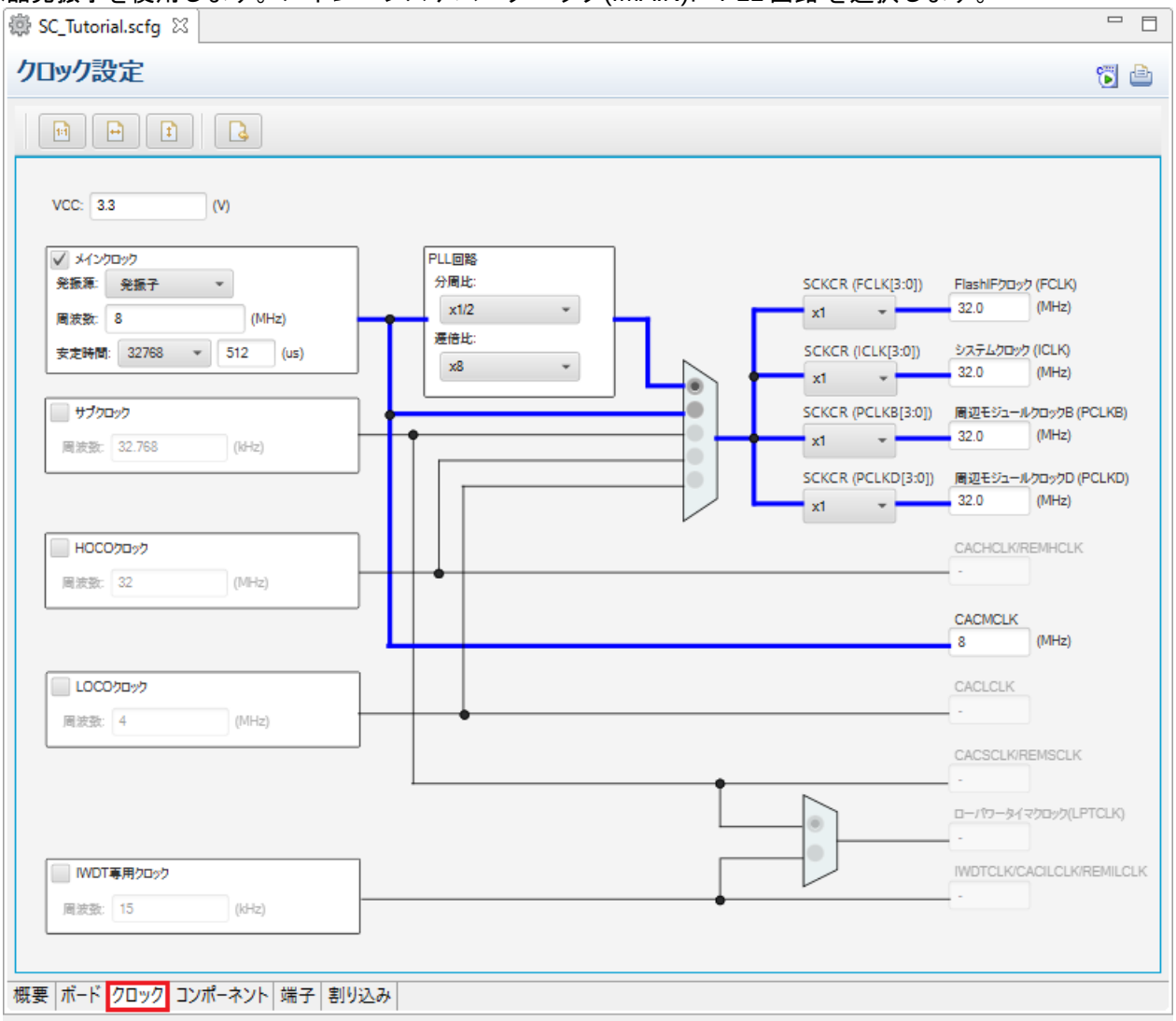


図 4-2: クロック設定

4.4 コンポーネントページ

ドライバとミドルウェアは、スマート・コンフィグレータのソフトウェアコンポーネントとして扱われます。コンポーネントページでは、ソフトウェアコンポーネントを選択して周辺機能を構成します。

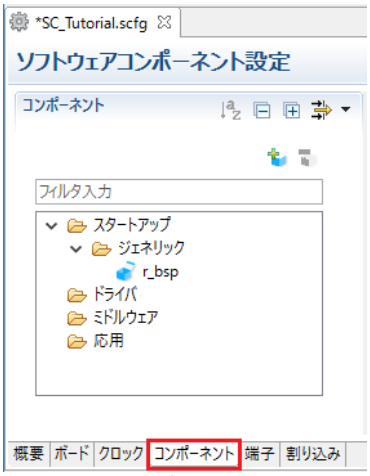


図 4-3: コンポーネントページ

4.4.1 ソフトウェアコンポーネントの追加

スマート・コンフィグレータは、コード生成ツールと Firmware Integration Technology(FIT)の 2 種類のソフトウェアコンポーネントをサポートしています。以降のセクションでは、コード生成ツールのコンポーネントによって、スイッチ入力、タイマ、ADC、および SCI の割り込みを含む簡単なプロジェクト用に MCU を設定する手順を説明します。

‘コンポーネントの追加’ アイコンをクリックします。

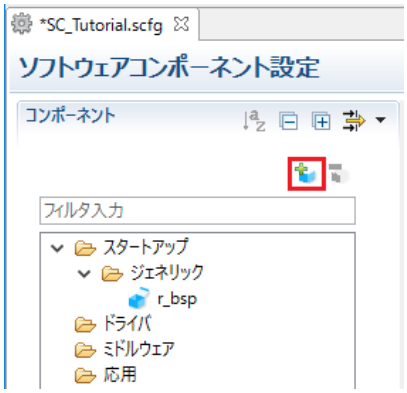


図 4-4: ソフトウェアコンポーネントの追加(1)

“ソフトウェアコンポーネントの選択”ダイアログ - > “コード生成”を選択します。

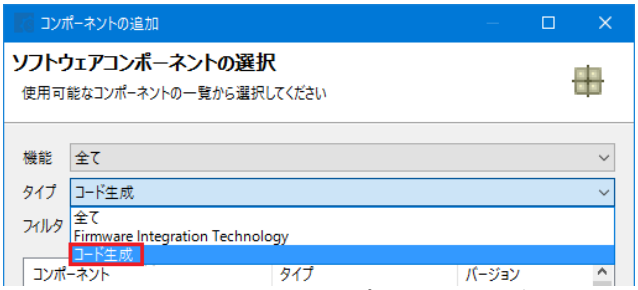


図 4-5: ソフトウェアコンポーネントの追加(2)

4.4.2 8 ビットタイマ

TMR0 をディレイ用インターバルタイマ割り込みに使用します。8 ビットタイマの設定を行います。**図 4-6** のように‘8 ビットタイマ’を選択し、“次へ(N)”をクリックします。

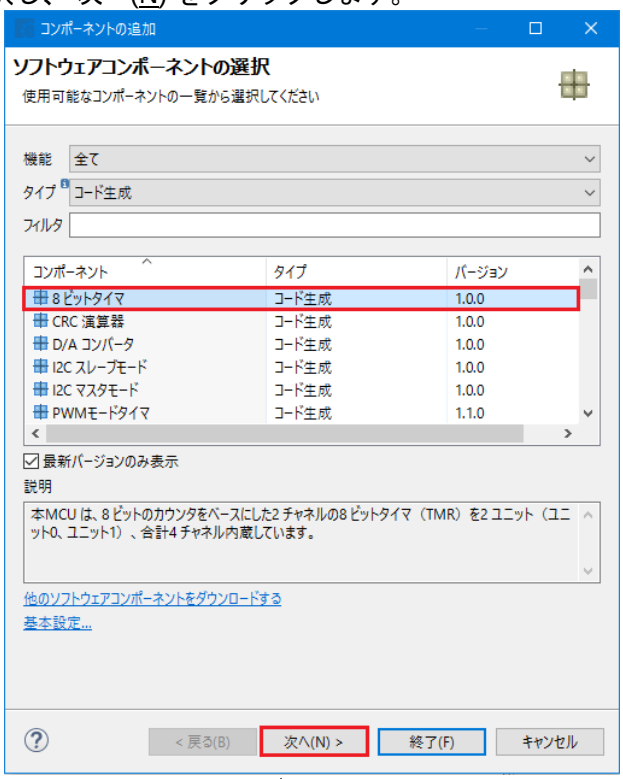


図 4-6: 8 ビットタイマ選択

“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 - > “リソース”で、**図 4-7** のように“TMR0”を選択し、“終了(F)”をクリックします。

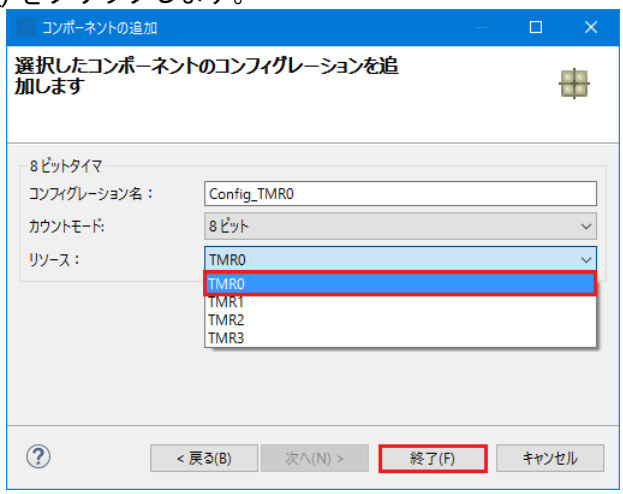


図 4-7: リソース選択 – TMR0

“Config_TMR0”の設定を**図 4-8** の通り設定してください。TMR0 は 1ms 毎に割り込みを発生させます。チュートリアルではアプリケーションのディレイ用として使用します。

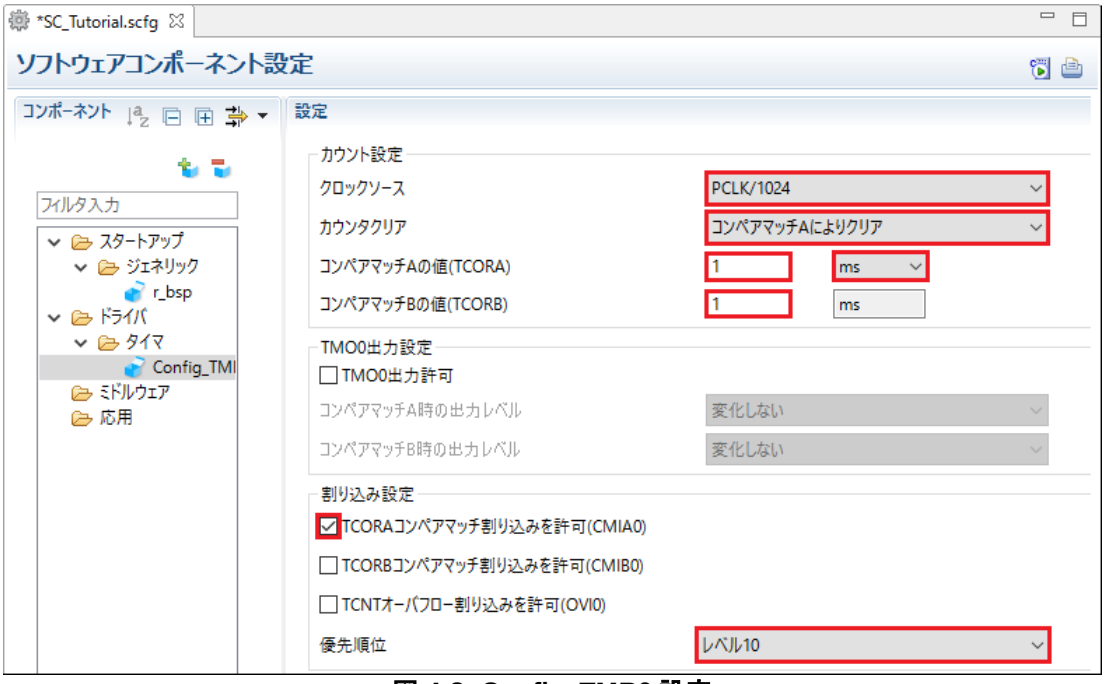


図 4-8: Config_TMR0 設定

4.4.3 コンペアマッチタイマ

CMT0 および CMT1 をスイッチのデバウンス用割り込みに使用します。

‘コンポーネントの追加’ アイコンをクリックします。“ソフトウェアコンポーネントの選択”ダイアログ - > “コード生成”を選択します。‘コンペアマッチタイマ’を選択し、“次へ(N)”をクリックします。

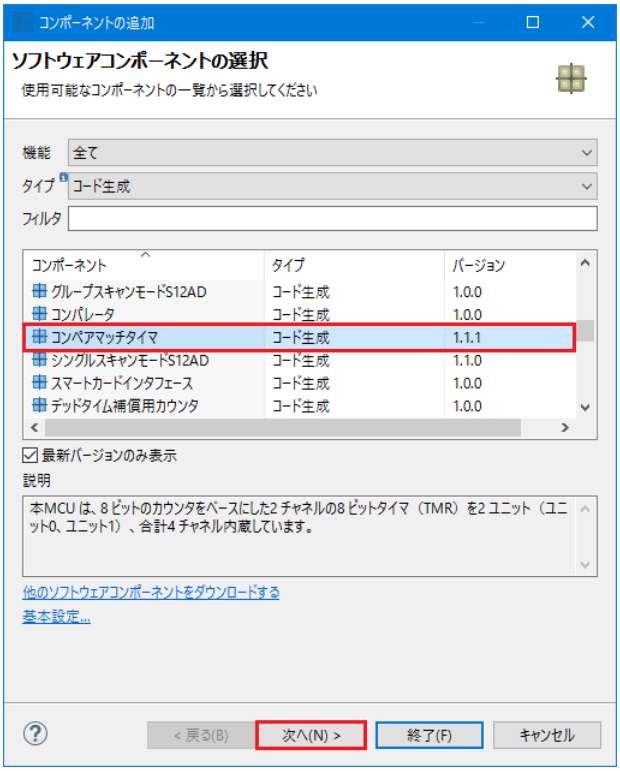


図 4-9: コンペアマッチタイマ選択

“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 - > “リソース”で、**図 4-10** のように“CMT0”を選択し、“終了(F)”をクリックします。

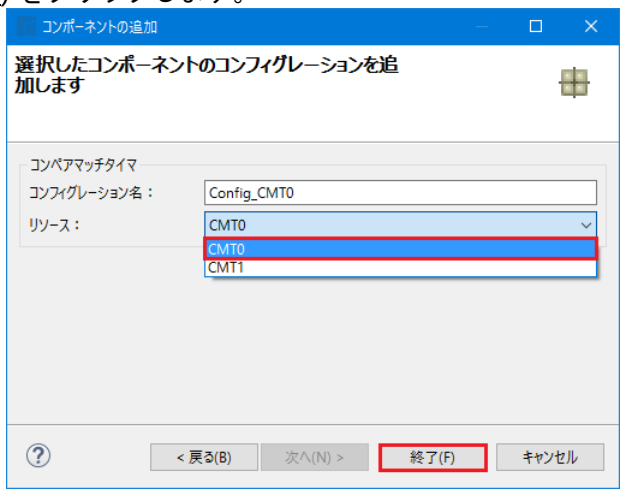


図 4-10: リソース選択 – CMT0

“Config_CMT0”の設定を**図 4-11** の通り設定してください。CMT0 は 20ms 毎に割り込みを発生させます。チュートリアルではスイッチのデバウンス用として使用します。

図 4-11: Config_CMT0 設定

‘コンポーネントの追加’ アイコンをクリックします。“ソフトウェアコンポーネントの選択”ダイアログ - > “コード生成”を選択します。‘コンペアマッチタイマ’を選択し、“次へ(N)”をクリックします。
“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 -> “リソース”で、**図 4-12** のように“CMT1”を選択し、“終了(F)”をクリックします。

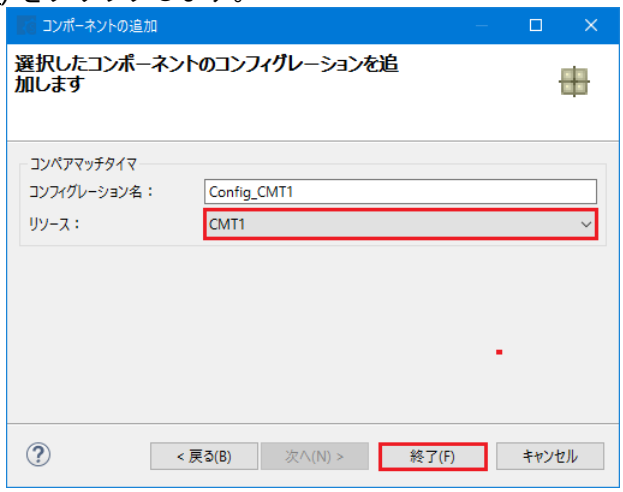


図 4-12: リソース選択 – CMT1

“Config_CMT1”の設定を**図 4-13** の通り設定してください。CMT1 は 200ms 毎に割り込みを発生させます。チュートリアルではスイッチのデバウンス用として使用します。

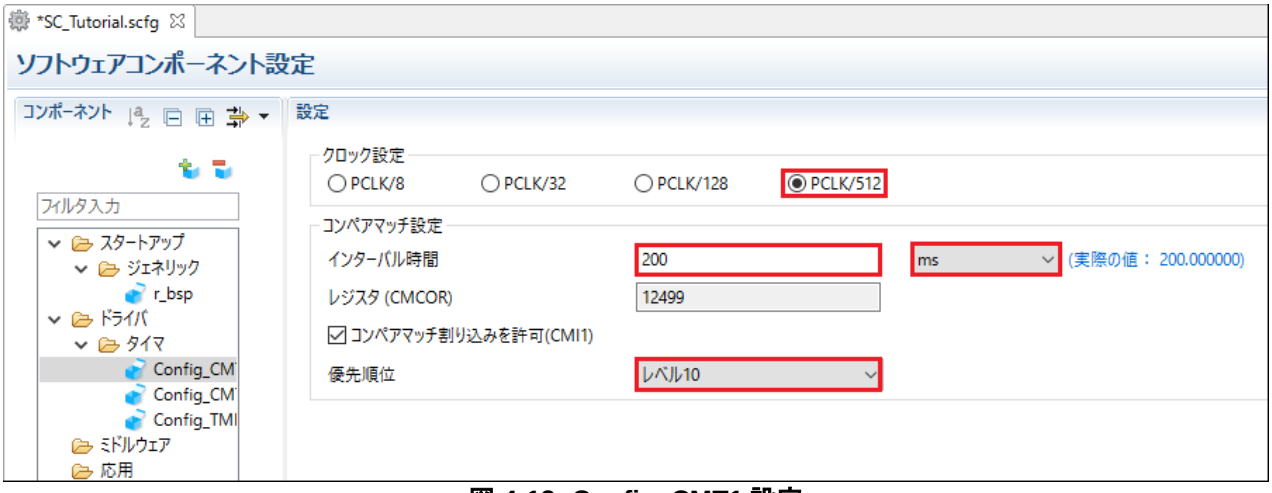


図 4-13: Config_CMT1 設定

4.4.4 割り込みコントローラ

RSKRX130-512KB の CPU ボードは SW1 に IRQ1(P31)、SW2 に IRQ2(P32)、SW3 に ADTRG0n が接続されています。ADTRG0n はセクション 4.4.8 で設定します。

‘コンポーネントの追加’ アイコンをクリックします。“ソフトウェアコンポーネントの選択”ダイアログ - > “コード生成”を選択します。‘割り込みコントローラ’を選択し、“次へ(N)”をクリックします。

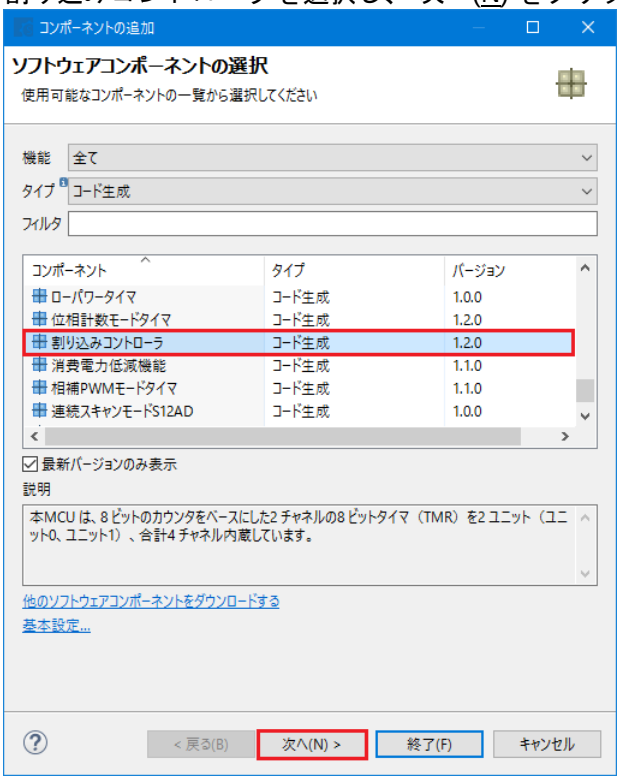


図 4-14: 割り込みコントローラ選択

“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 -> “リソース”で、**図 4-15** のように“ICU”を選択し、“終了(F)”をクリックします。

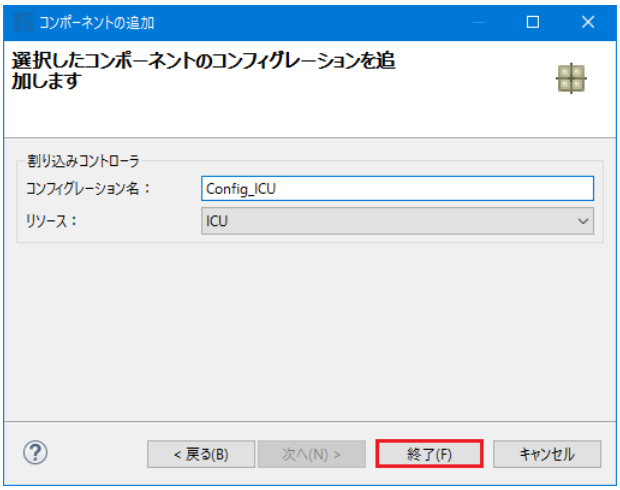


図 4-15: リソース選択 – ICU

‘Config_ICU’で IRQ1、IRQ2 を**図 4-16** のように設定してください。

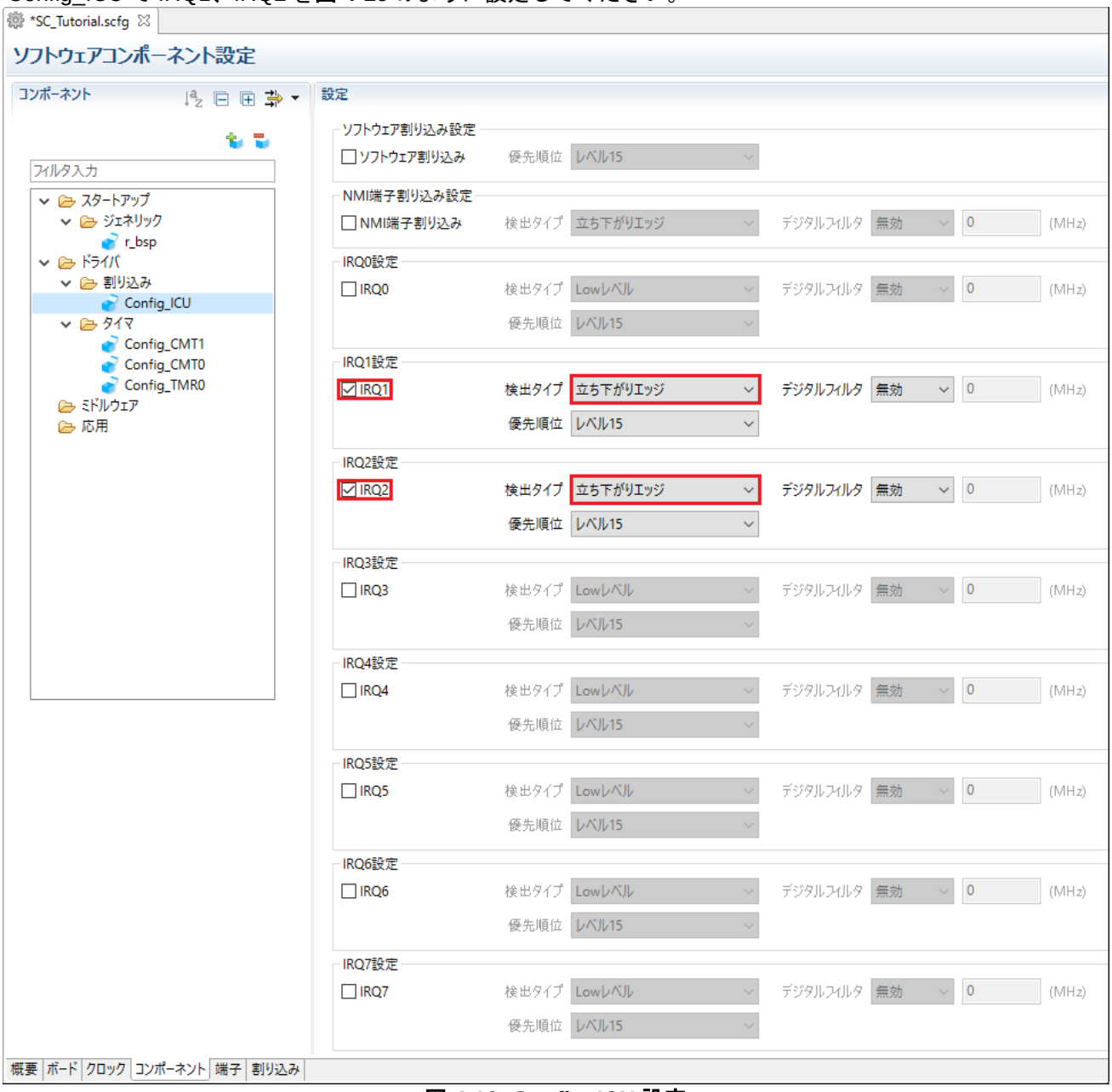


図 4-16: Config_ICU 設定

4.4.5 ポート設定

RSKRX130-512KB の CPU ボードは LED0 に PD3、LED1 に PD4、LED2 に PE6、LED3 に PE7 が接続されています。また、Pmod LCD に P17、PB2、PC2、PC3 が接続されています。LED と Pmod LCD ポートを設定します。

コンポーネントの追加’ アイコンをクリックします。“ソフトウェアコンポーネントの選択”ダイアログ -> “コード生成”を選択します。‘ポート’を選択し、“次へ(N)”をクリックします。

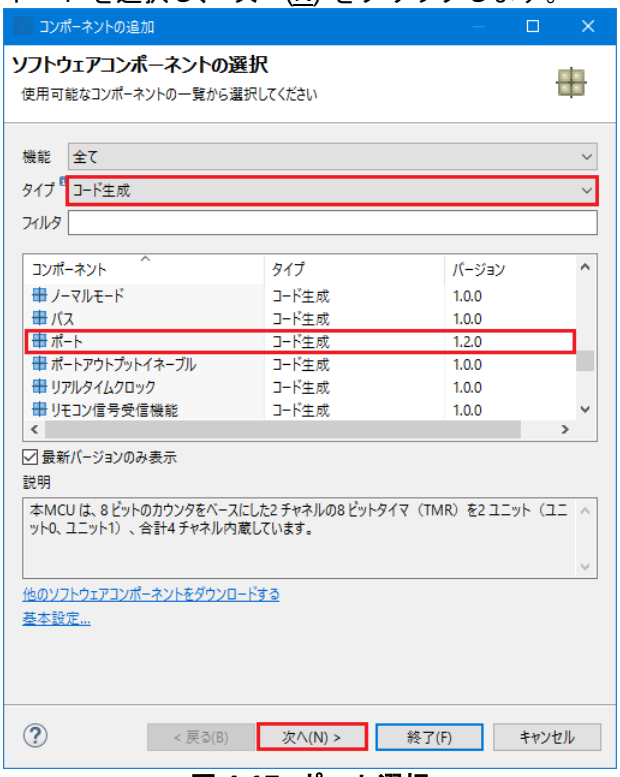


図 4-17: ポート選択

“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 -> “リソース”で、**図 4-18** のように“PORT”を選択し、“終了(F)”をクリックします。

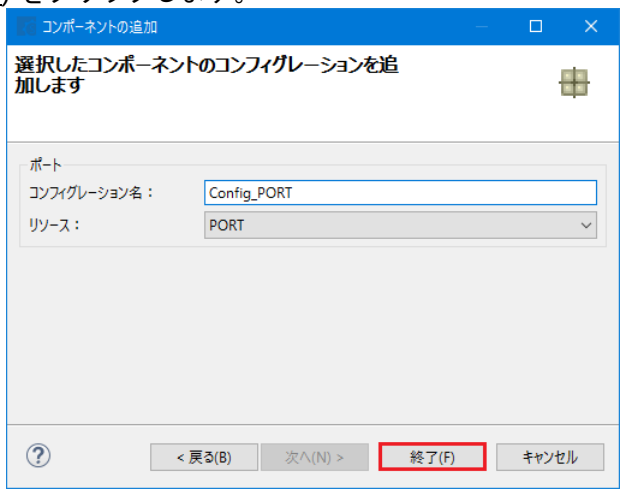


図 4-18: リソース選択 – PORT

図 4-19 の通り PORT1、PORTB、PORTC、PORTD、PORTE のチェックボックスをチェックしてください。

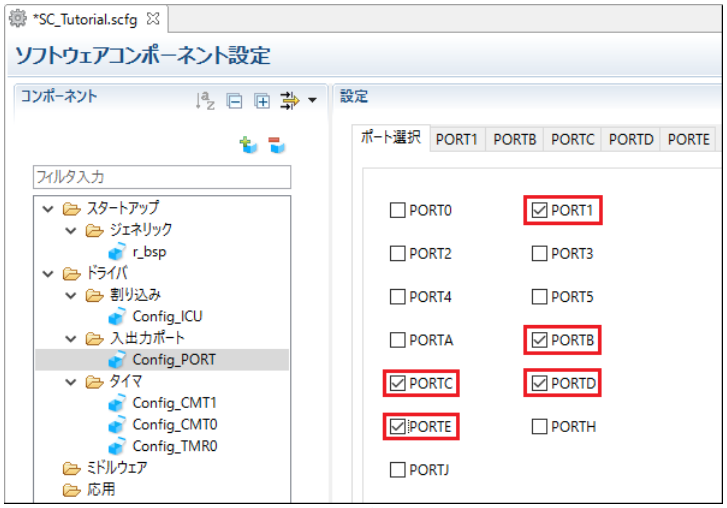


図 4-19: ポート選択

PORT1 の設定を**図 4-20**、PORTB の設定を**図 4-21**、PORTC の設定を**図 4-22**、PORTD の設定を**図 4-23**、PORTE の設定を**図 4-24** に示します。また、PC3 以外は、“1 を出力”チェックボックスにチェックを入れてください。PORT1 タブを選択してください。

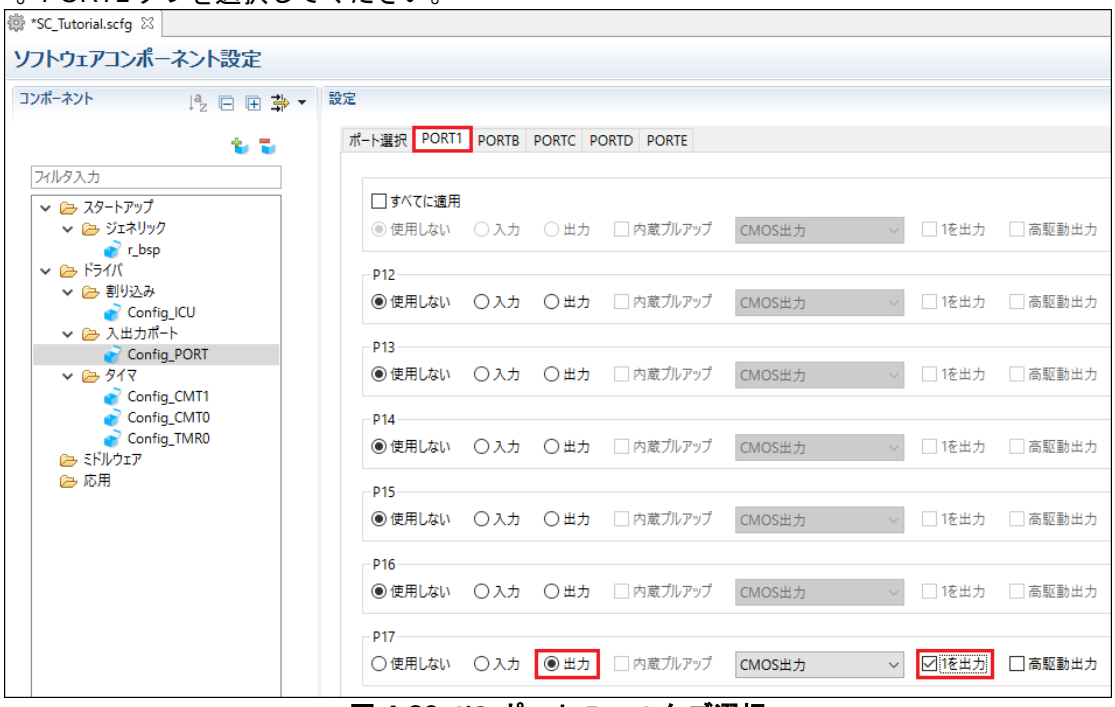


図 4-20: I/O ポート Port1 タブ選択

PORTB タブを選択してください。

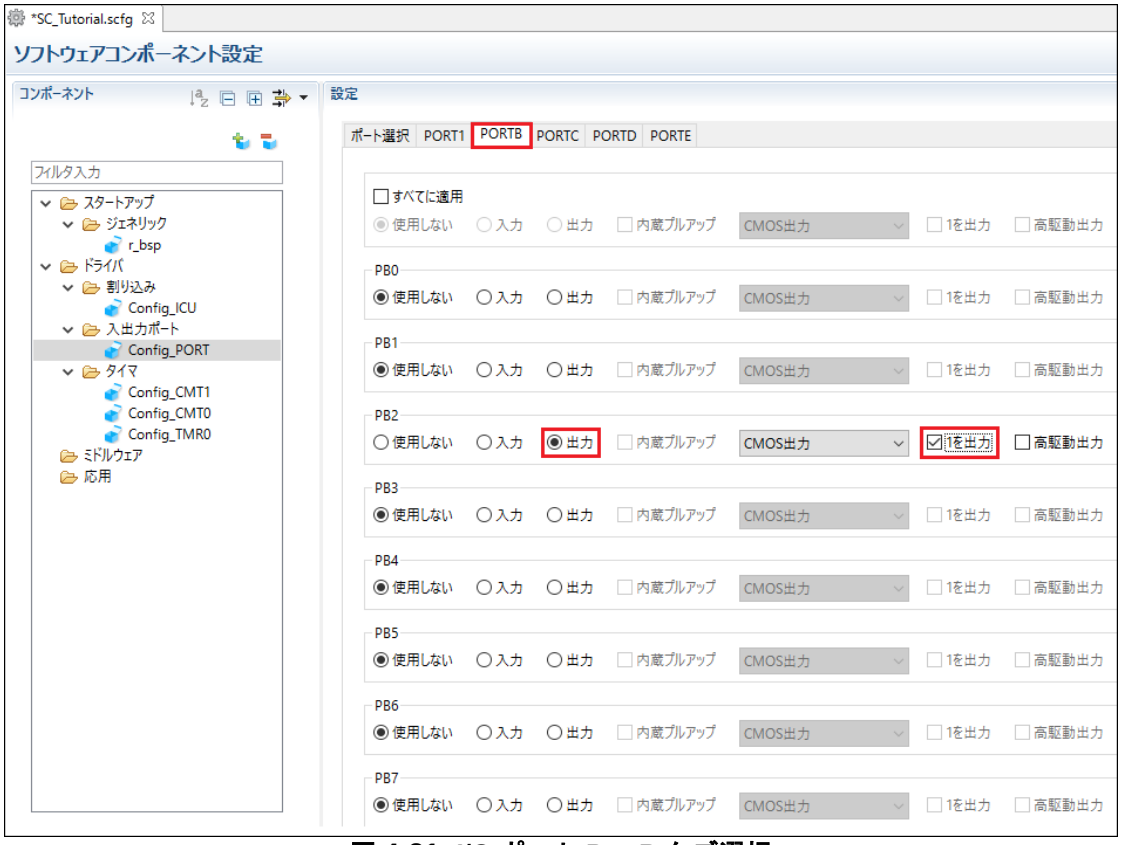


図 4-21: I/O ポート PortB タブ選択

PORTC タブを選択してください。

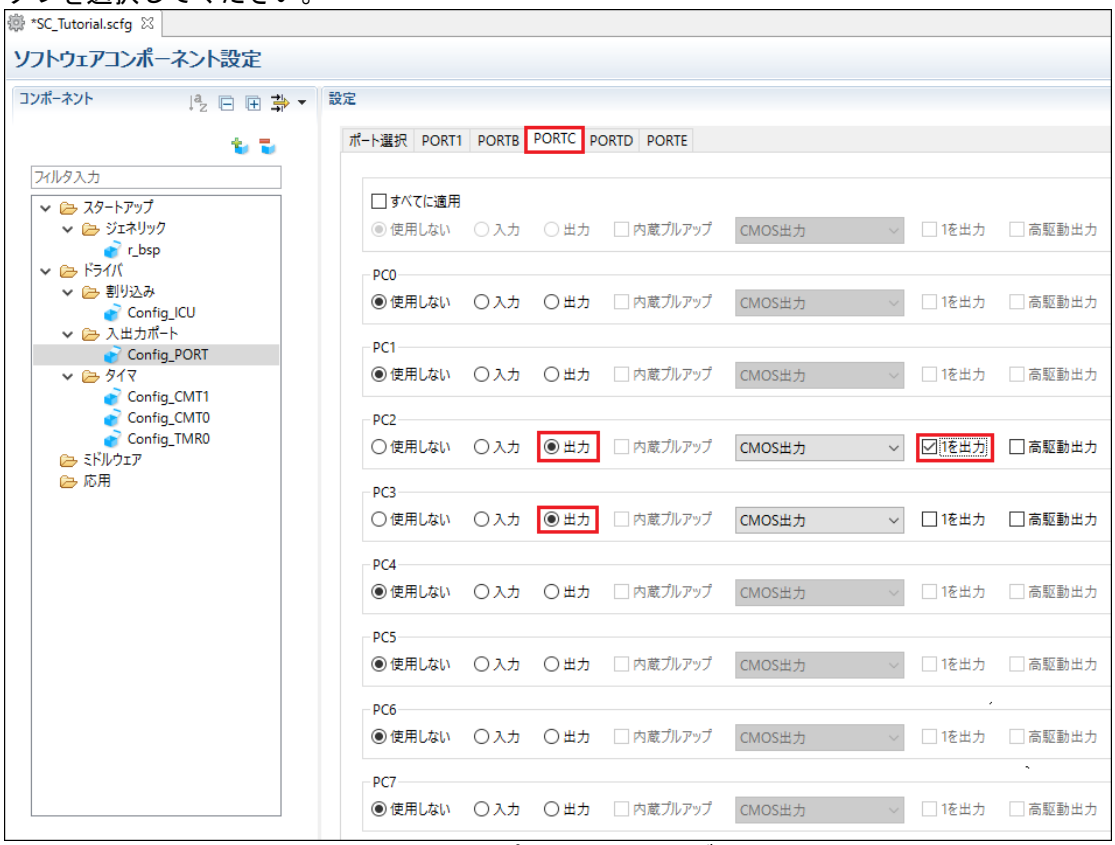


図 4-22: I/O ポート PortC タブ選択

PORTD タブを選択してください。

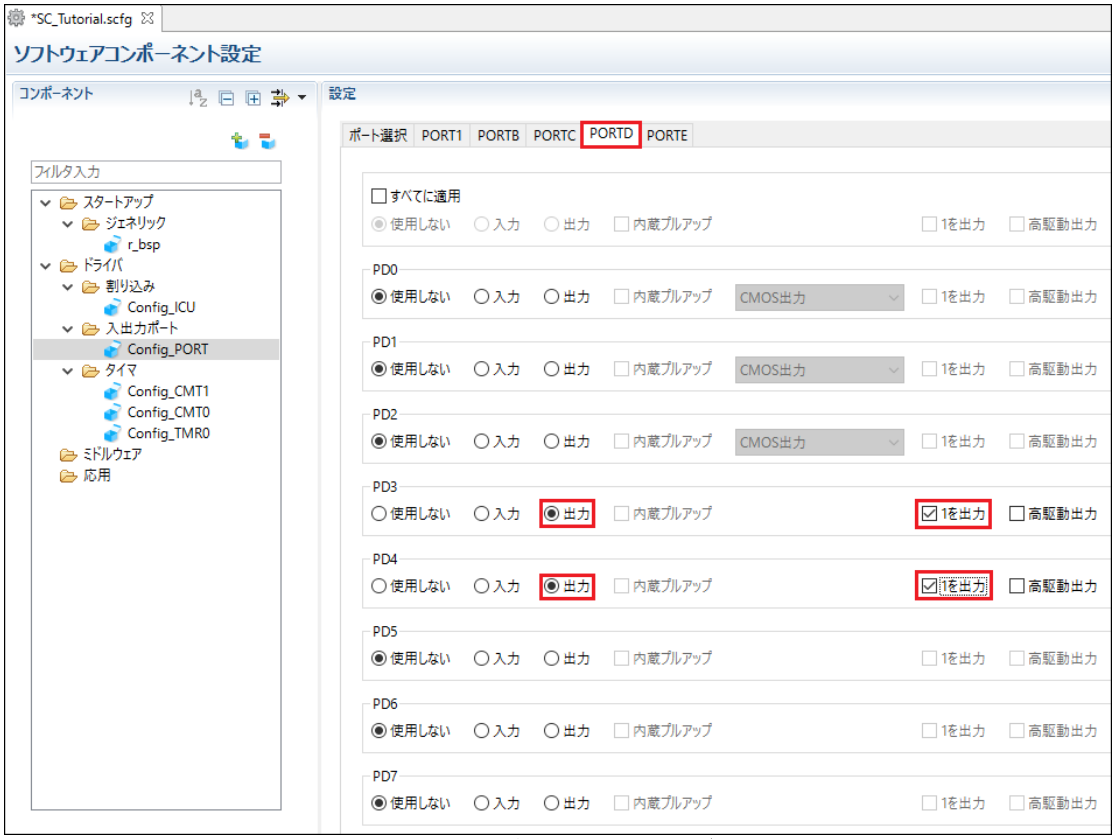


図 4-23: I/O ポート PortD タブ選択

PORTE タブを選択してください。

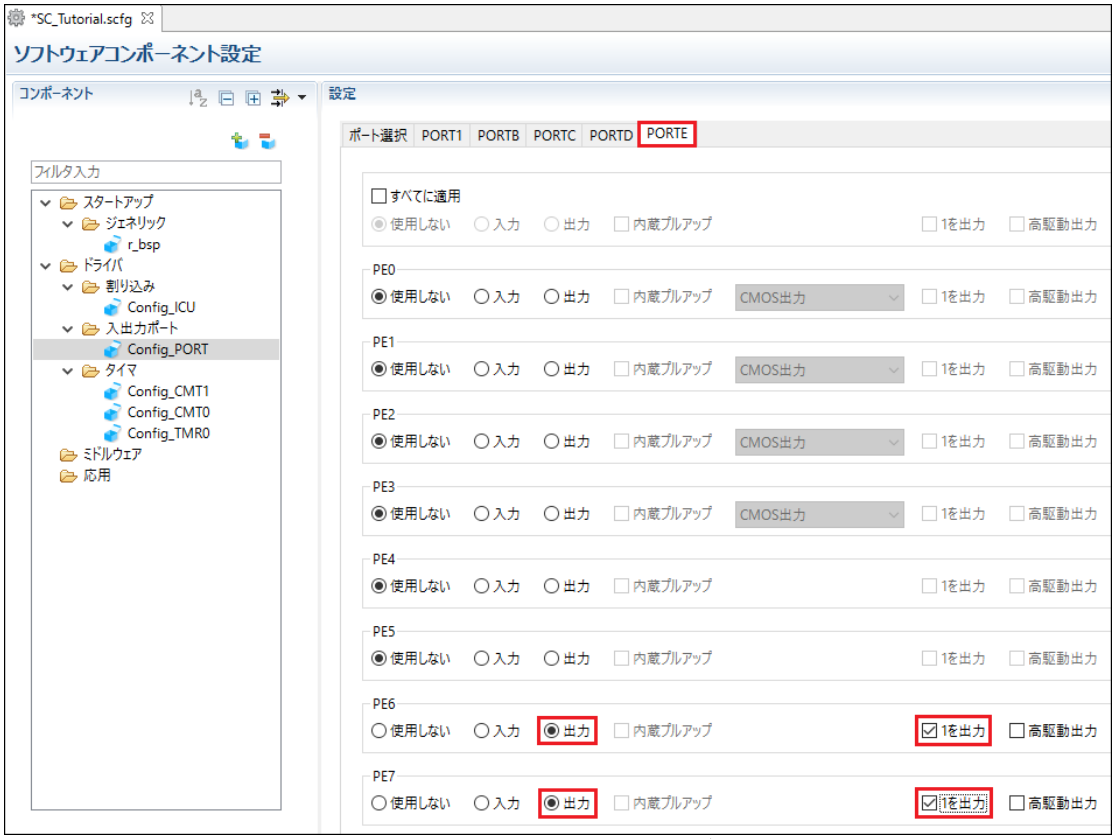


図 4-24: I/O ポート PortE タブ選択

4.4.6 SCI(SCIF)調歩同期式モード

RSKRX130-512KB の CPU ボードは SCI1 が RL78/G1C マイクロコントローラのシリアルポートに接続されており、仮想 COM ポートとして使用します。

‘コンポーネントの追加’ アイコンをクリックします。“ソフトウェアコンポーネントの選択”ダイアログ - > “コード生成”を選択します。‘SCI(SCIF)調歩同期式モード’を選択し、“次へ(N)”をクリックします。

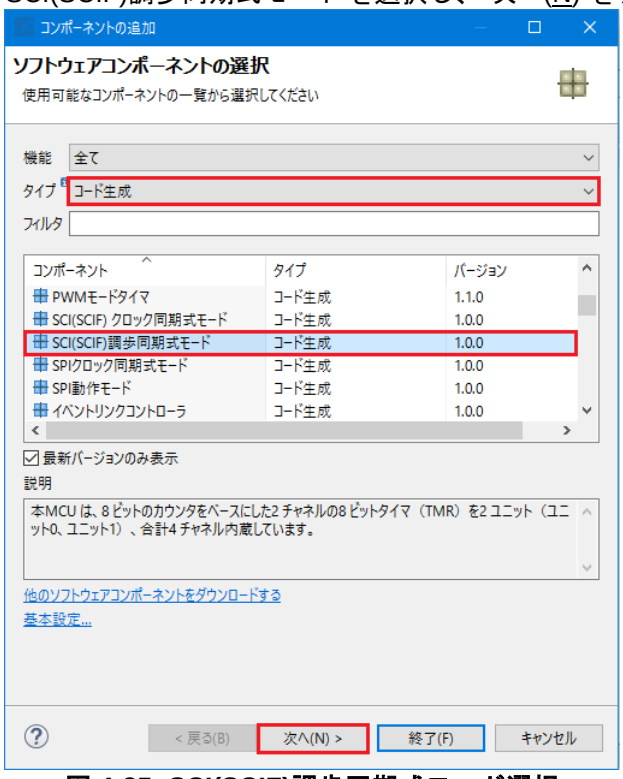


図 4-25: SCI(SCIF)調歩同期式モード選択

“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 -> “作業モード”で、**図 4-26** のように“送信/受信”を選択します。

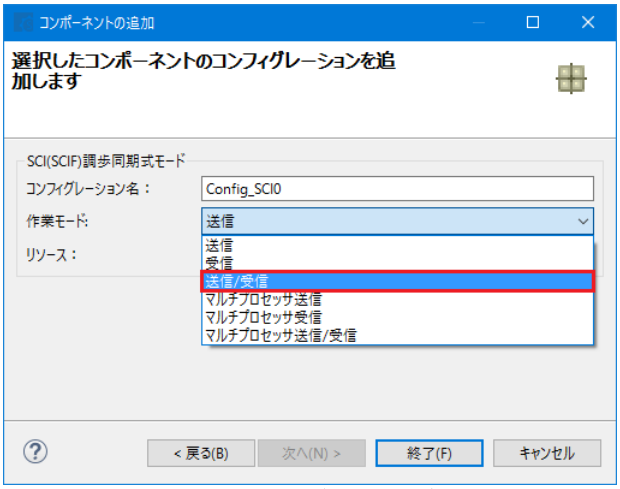


図 4-26: 作業モード選択 – 送信/受信

“リソース”で、**図 4-27** のように“SCI1”を選択してください。

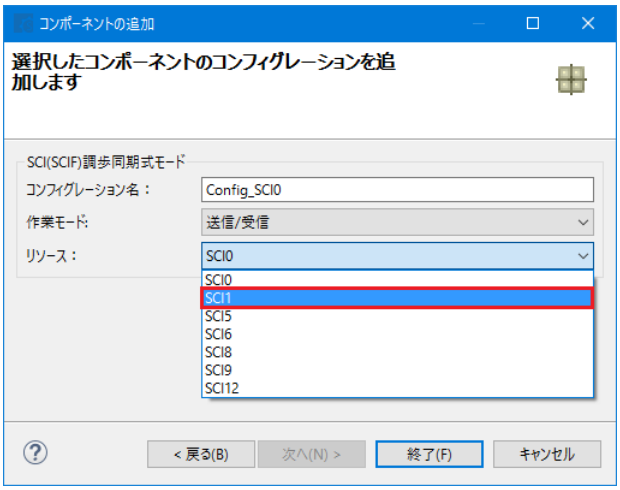


図 4-27: リソース選択 – SCI1

図 4-28 のようにコンフィグレーション名が“Config_SCI1”であることを確認し、“終了(F)”をクリックします。

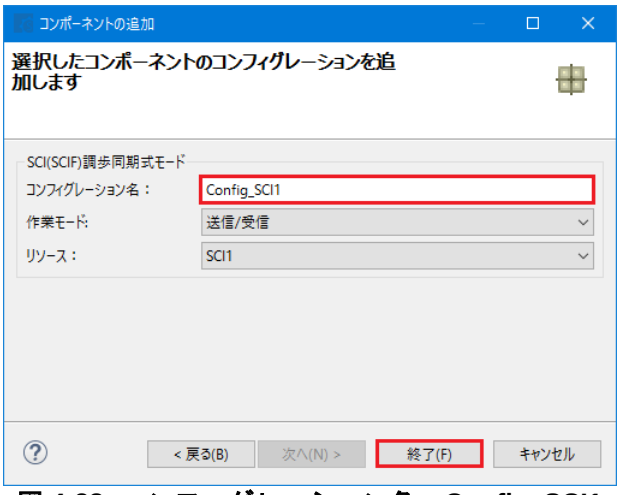


図 4-28: コンフィグレーション名 – Config_SCI1

図 4-29 のように、スタートビット検出設定を RXD1 端子の立ち下がりエッジ、ビットレートを 19200 に設定してください。

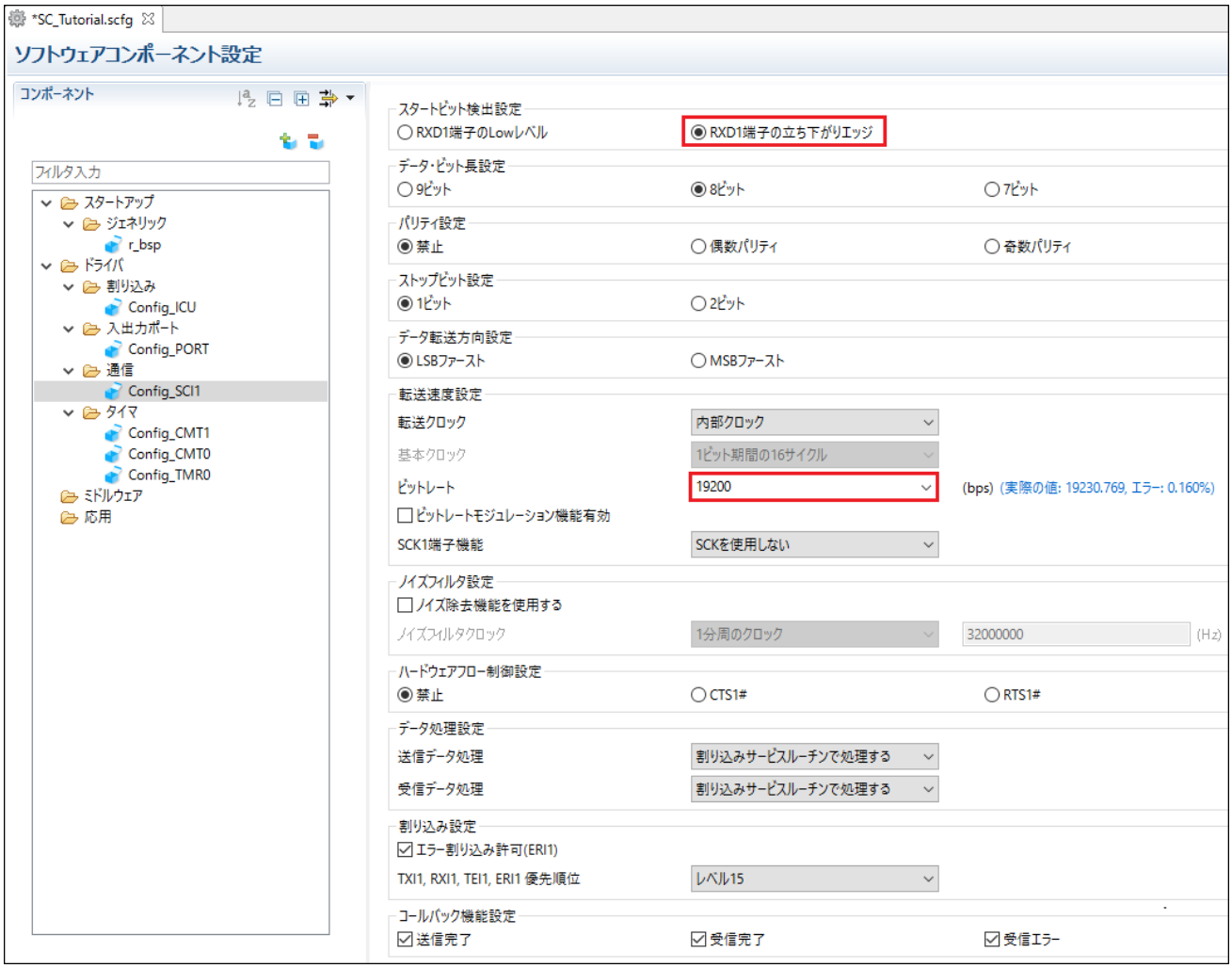


図 4-29: Config_SCI1 設定

4.4.7 SPI クロック同期式モード

チュートリアルでは SCI6 で Pmod LCD を制御します。

‘コンポーネントの追加’ アイコンをクリックします。“ソフトウェアコンポーネントの選択”ダイアログ - > “コード生成”を選択します。‘SPI クロック同期式モード’を選択し、“次へ(N)”をクリックします。

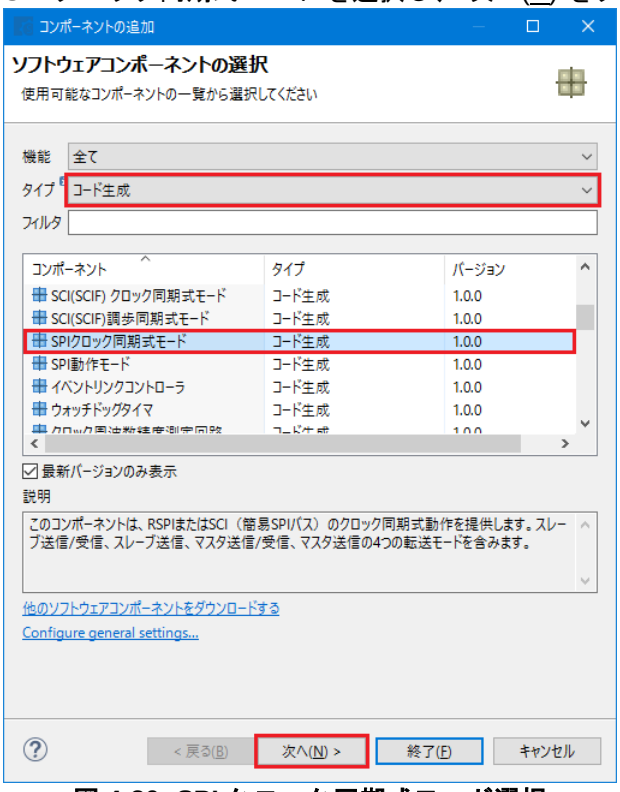


図 4-30: SPI クロック同期式モード選択

“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 -> “動作”で、**図 4-31** のように“マスタ送信機能”を選択します。

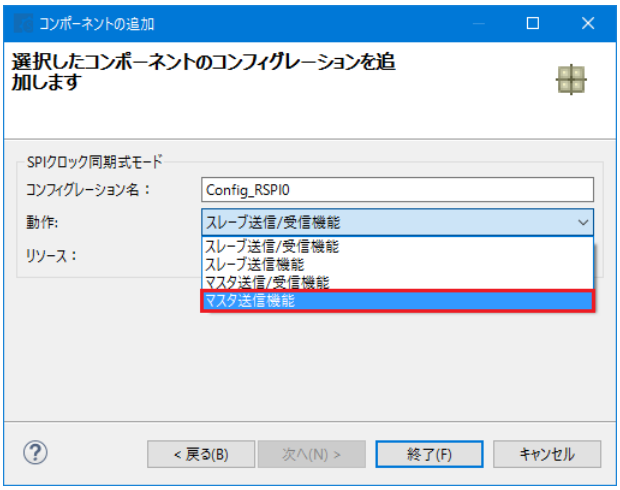


図 4-31: 動作選択 – マスタ送信機能

“リソース”で、**図 4-32** のように“SCI6”を選択してください。

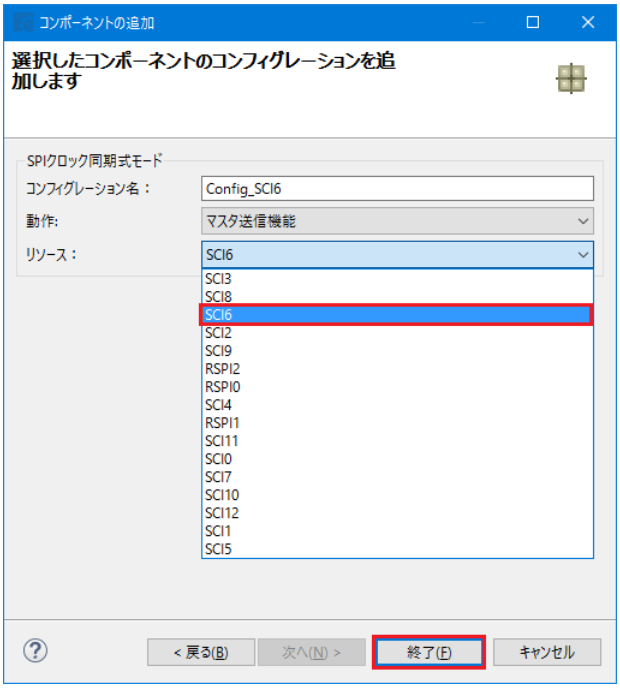


図 4-32: リソース選択 – SCI6

図 4-33 のように、コンフィグレーション名が“Config_SCI6”であることを確認し、“終了(F)”をクリックします。

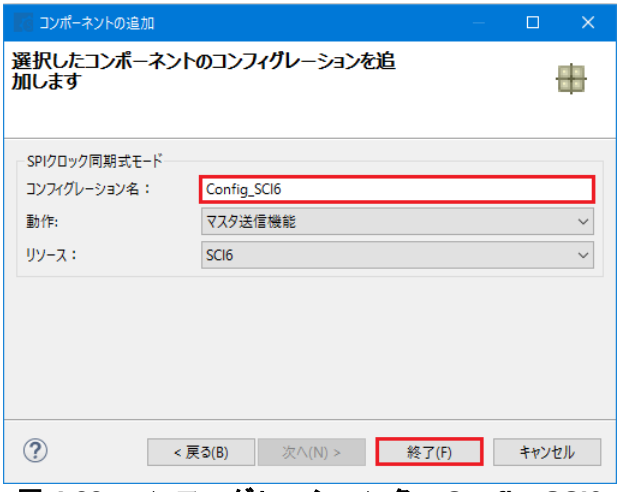


図 4-33: コンフィグレーション名 – Config_SCI6

図 4-34 のように、データ転送方向設定を MSB ファースト、ビットレートを 8000kbps に設定してください。

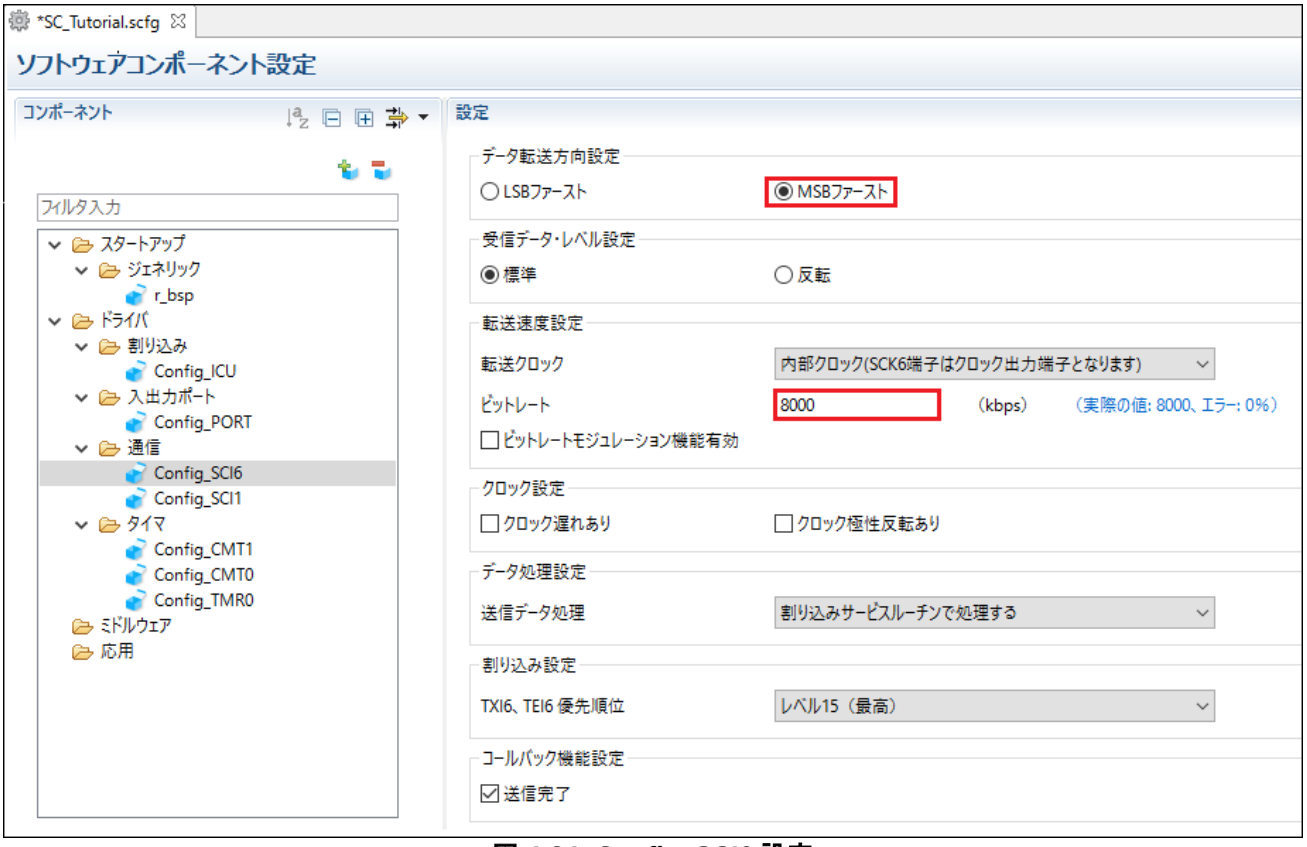


図 4-34: Config_SCI6 設定

4.4.8 シングルスキャンモード S12AD

CPU ボード上のポテンショメータ RV1 に接続される AN000 端子の入力電圧をシングルスキャンモード S12AD で A/D 変換を行います。SW3 を A/D 変換開始トリガとして設定します。

‘コンポーネントの追加’ アイコンをクリックします。“ソフトウェアコンポーネントの選択”ダイアログ - > “コード生成”を選択します。**図 4-35** のように‘シングルスキャンモード S12AD’を選択し、“次へ(N)”をクリックします。

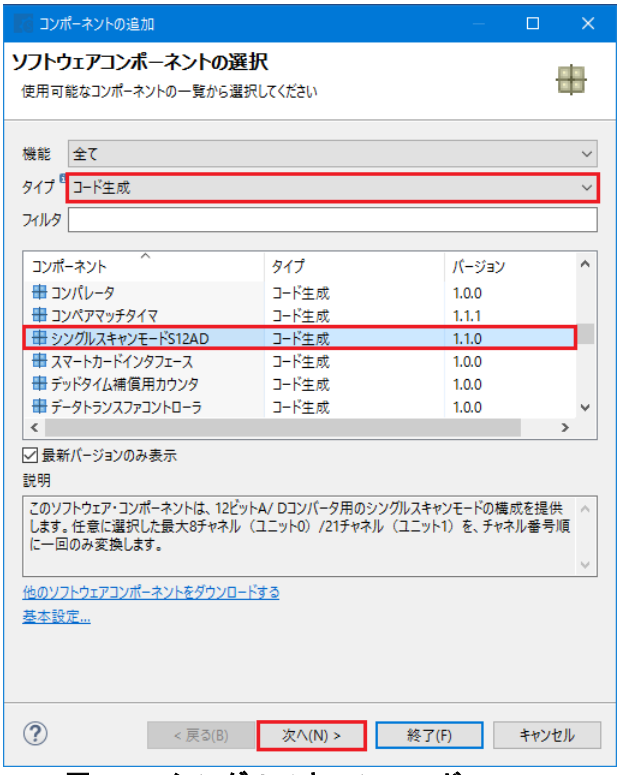


図 4-35: シングルスキャンモード S12AD

“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 -> “リソース”で、**図 4-36** のように“S12AD0”を選択し、“終了(F)”をクリックします。

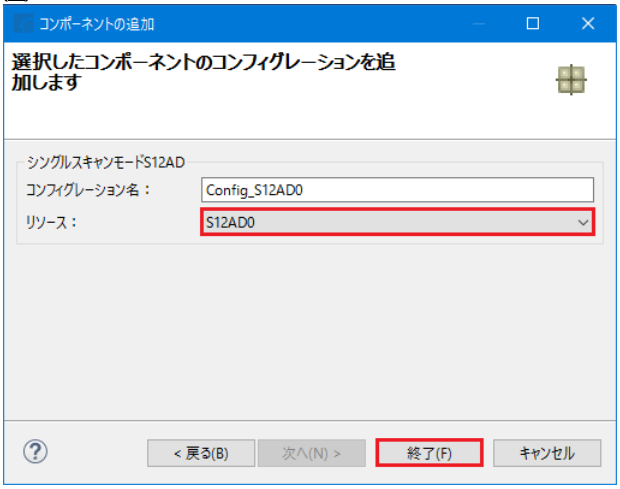


図 4-36: リソース選択 – S12AD0

図 4-37、図 4-38 のように、“アナログ入力チャネル設定”の AN000 チェックボックスをチェックしてください。“変換開始トリガ設定”の開始トリガソースを’トリガ入力端子’に設定してください。

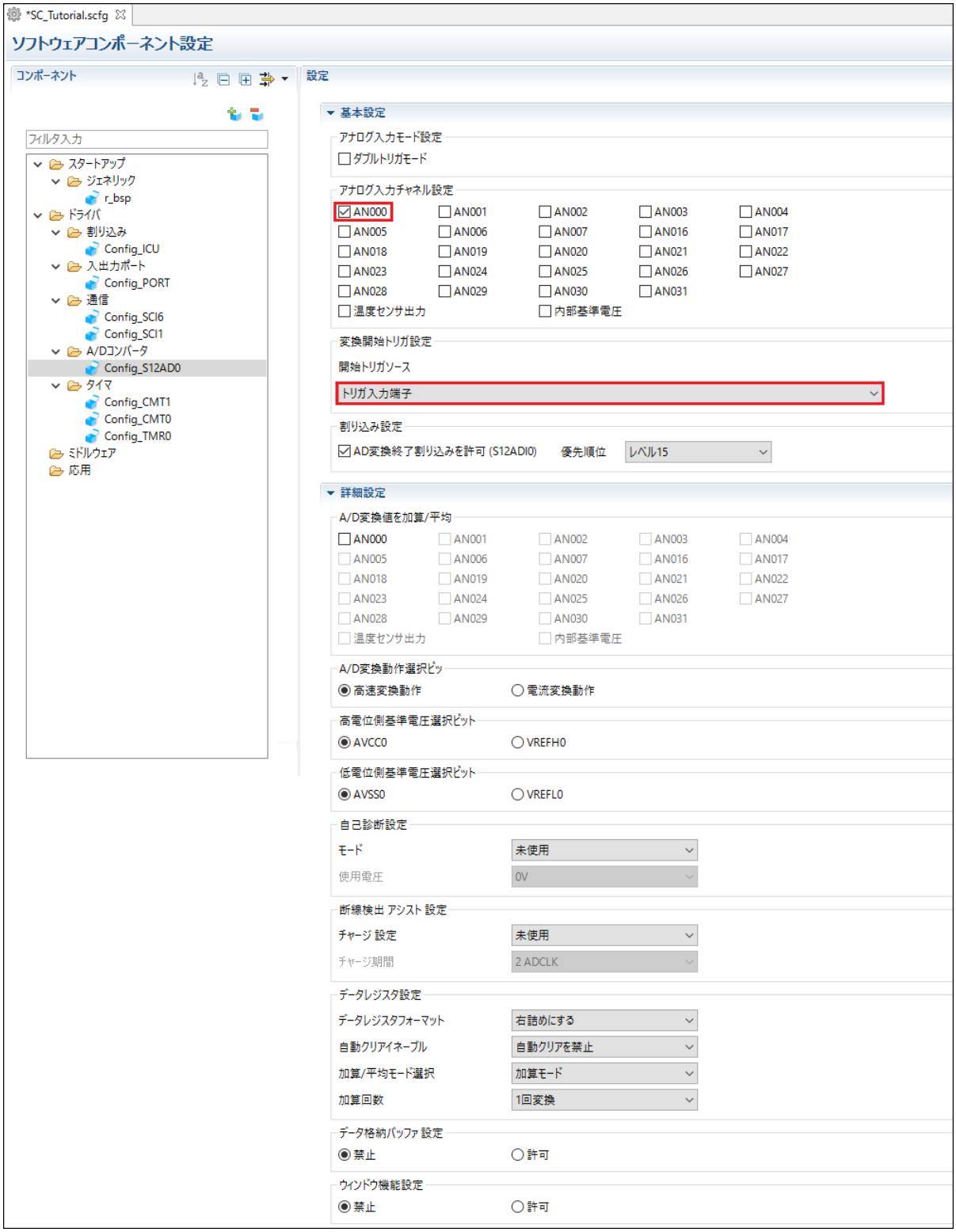


図 4-37: Config_S12AD0 設定(1)

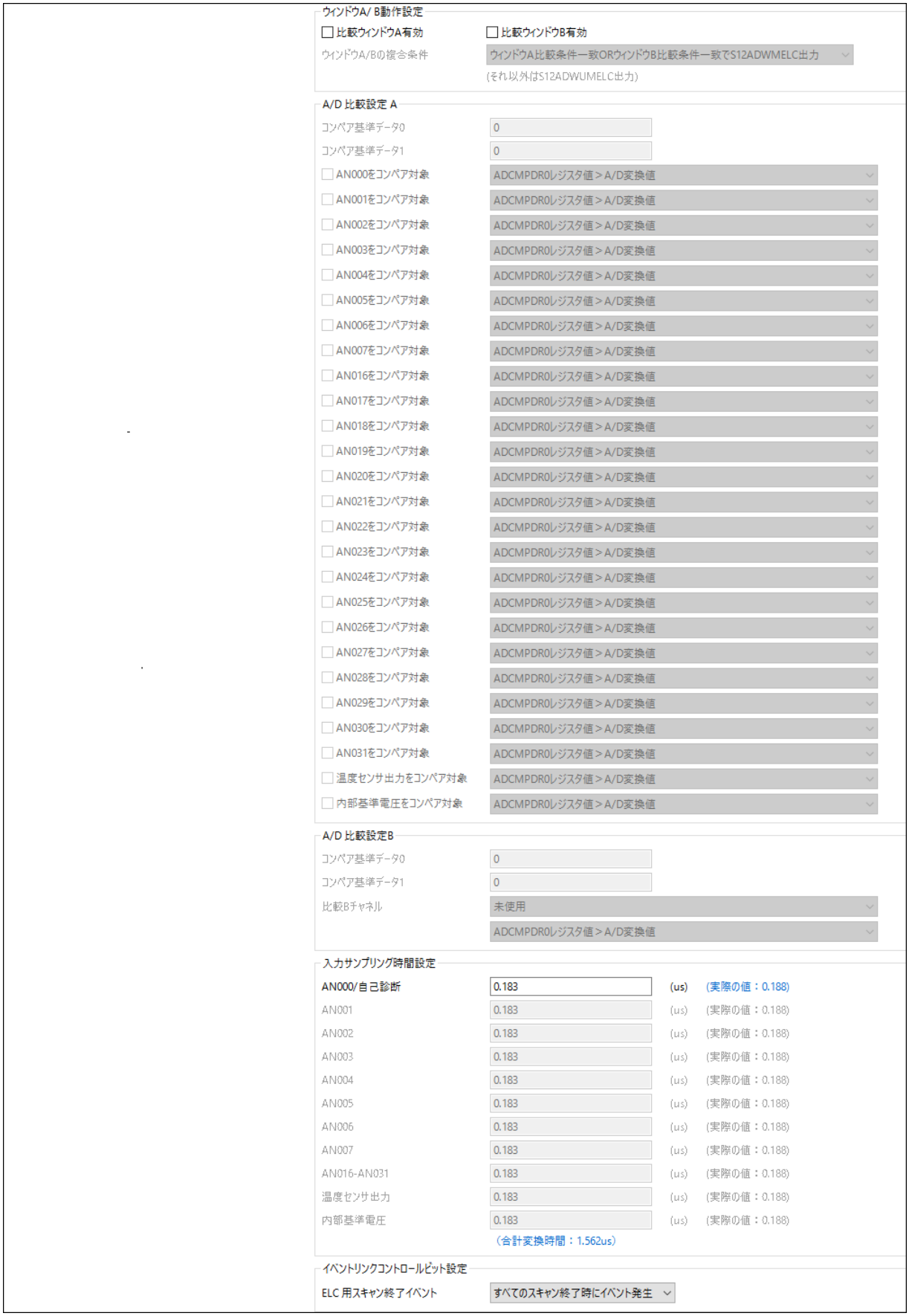


図 4-38: Config_S12AD0 設定(2)

4.5 端子設定ページ

スマート・コンフィグレータは、プロジェクトに追加されたソフトウェアコンポーネントにピンを割り当てます。ピンの割り当ては端子設定ページで変更できます。

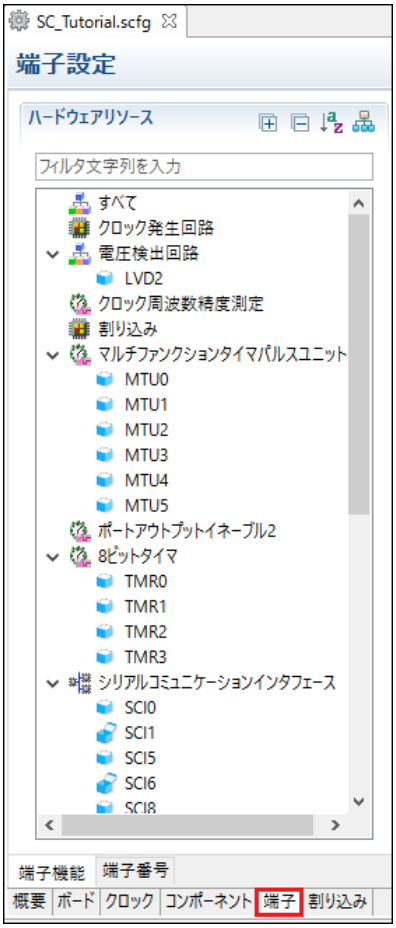


図 4-39: 端子設定ページ

4.5.1 ソフトウェアコンポーネントのピン設定変更

ピン機能一覧にあるソフトウェアコンポーネントのピン割り当てを変更します。  をクリックすると、選択したソフトウェアコンポーネントを表示できます。

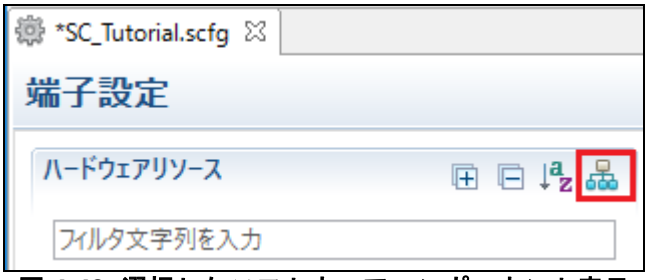


図 4-40: 選択したソフトウェアコンポーネント表示

ソフトウェアコンポーネントの Config_ICU を選択します。‘端子機能’ -> ‘端子割り当て’で、IRQ1 に P31、IRQ2 に P32 を設定してください。**図 4-41** に示すように、IRQ1 と IRQ2 の‘使用する’チェックボックスがオンであることを確認してください。

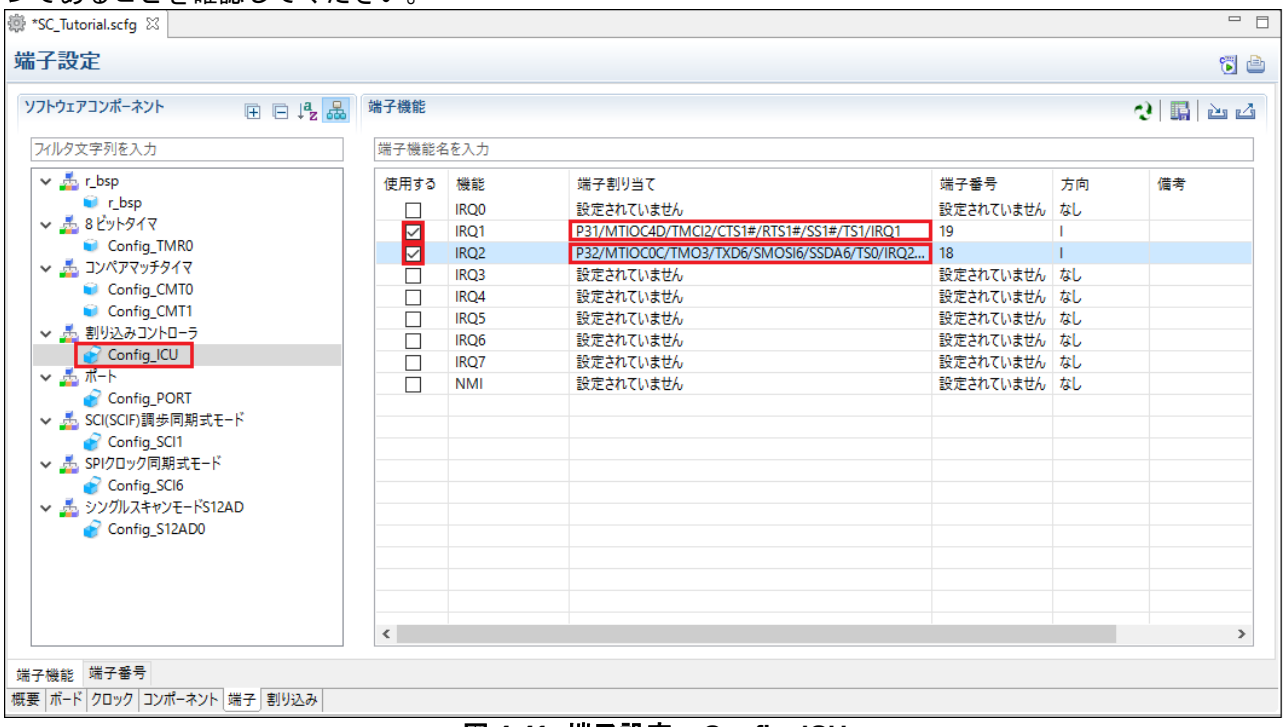


図 4-41: 端子設定 – Config_ICU

ソフトウェアコンポーネントの Config_SCI1 を選択します。‘端子機能’ -> ‘端子割り当て’で、RXD1 に P30、TXD1 に P26 を設定してください。**図 4-42** に示すように、RXD1 と TXD1 の‘使用する’チェックボックスがオンであることを確認してください。

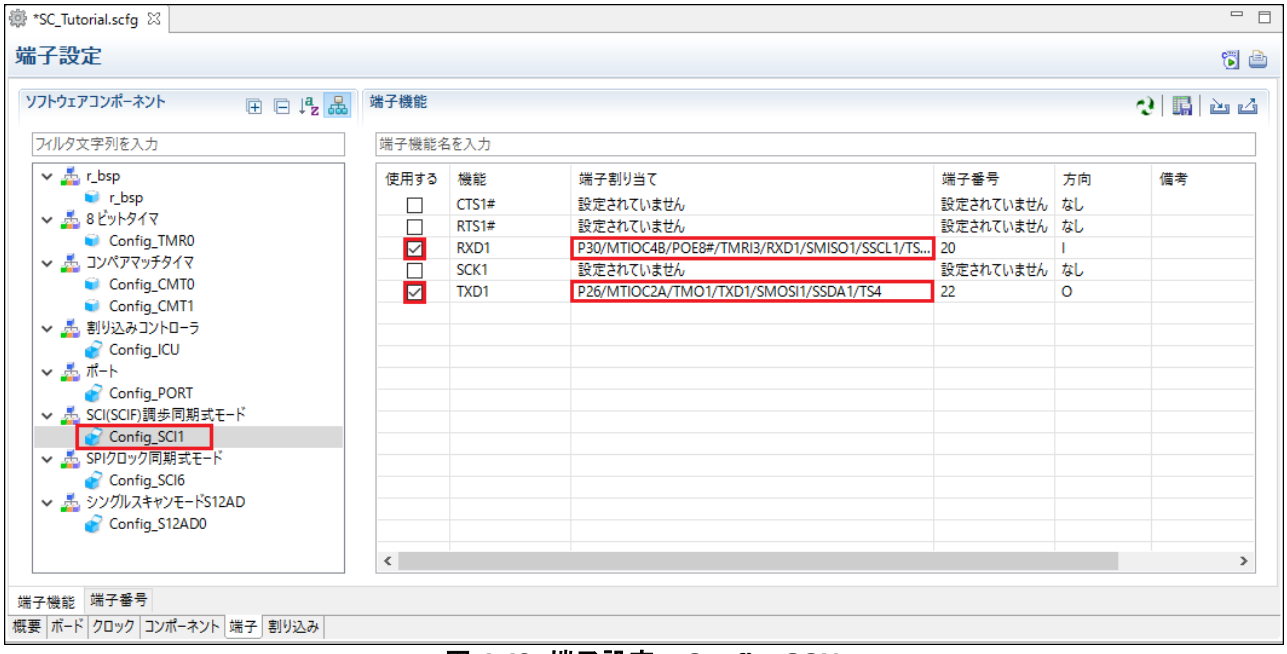


図 4-42: 端子設定 – Config_SCI1

ソフトウェアコンポーネントの Config_SCI6 を選択します。 ‘端子機能’ -> ‘端子割り当て’で SCK6 に PB3、SMOSI6 に PB1 を設定してください。**図 4-43** に示すように、SCK6 と SMOSI6 の‘使用する’チェックボックスがオンであることを確認してください。

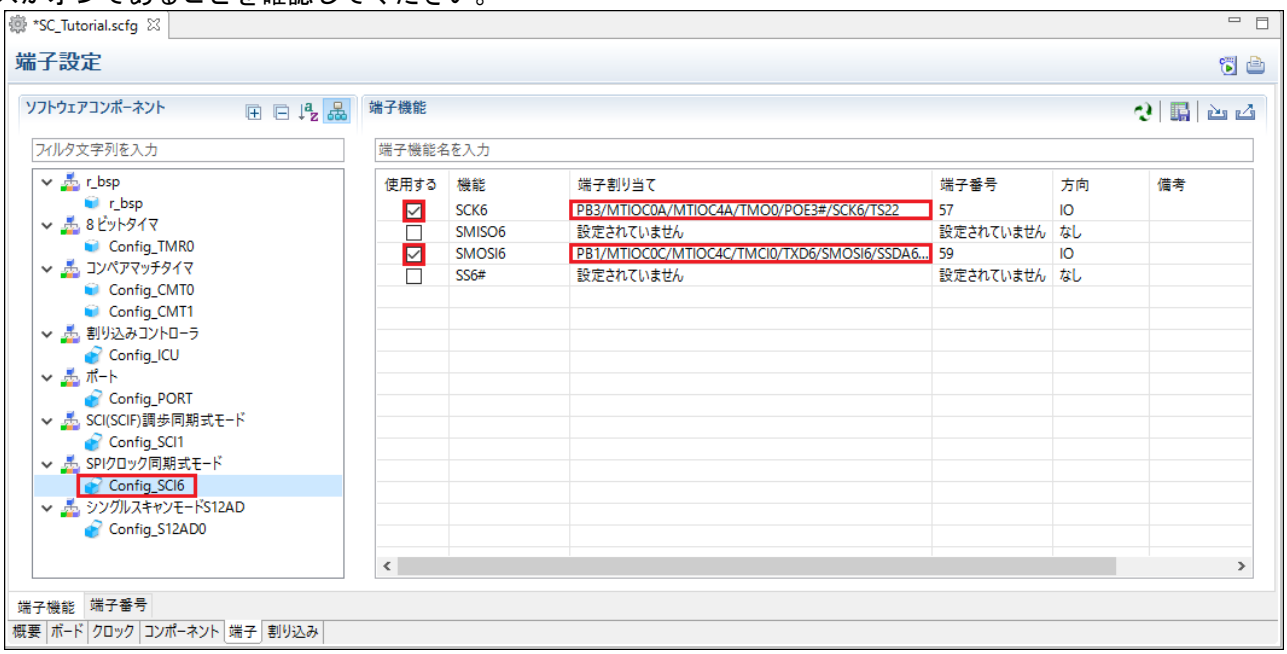


図 4-43: 端子設定 – Config_SCI6

ソフトウェアコンポーネントの Config_S12AD0 を選択します。‘端子機能’ -> ‘端子割り当て’で、AN000 に P40、ADTRG0#に P16 を設定してください。**図 4-44** に示すように、ADTRG0#、AN000、AVCC0、AVSS0 の‘使用する’チェックボックスがオンであることを確認してください。

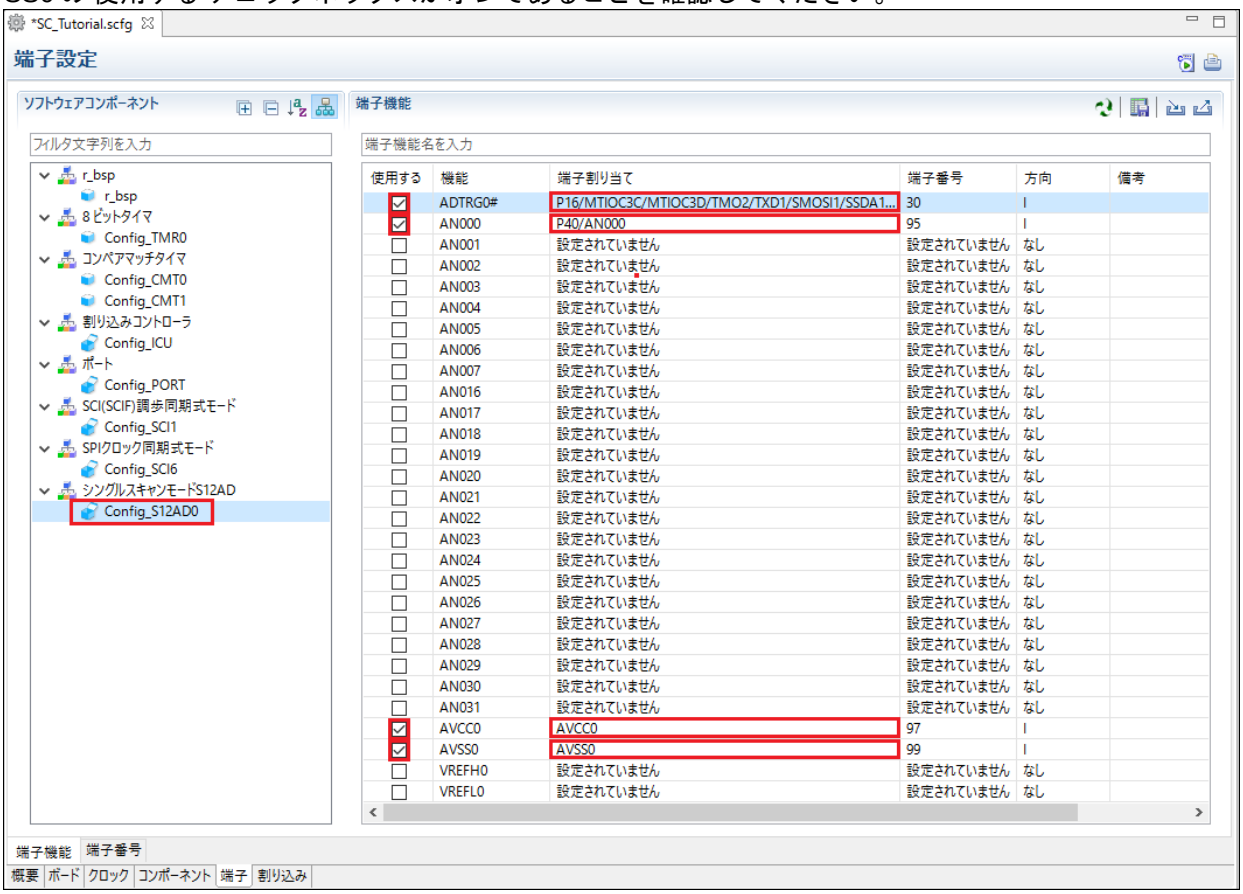


図 4-44: 端子設定 – Config_S12AD0

これで周辺機能の設定は全て完了しました。スマート・コンフィグレータのメニューバーの‘ファイル’から保存してください。次に**図 4-45** の位置にある、<コードの生成>ボタンをクリックして、コードを生成してください。

図 4-45: コード生成ボタン

図 4-46 に示すようにコードが生成されます。

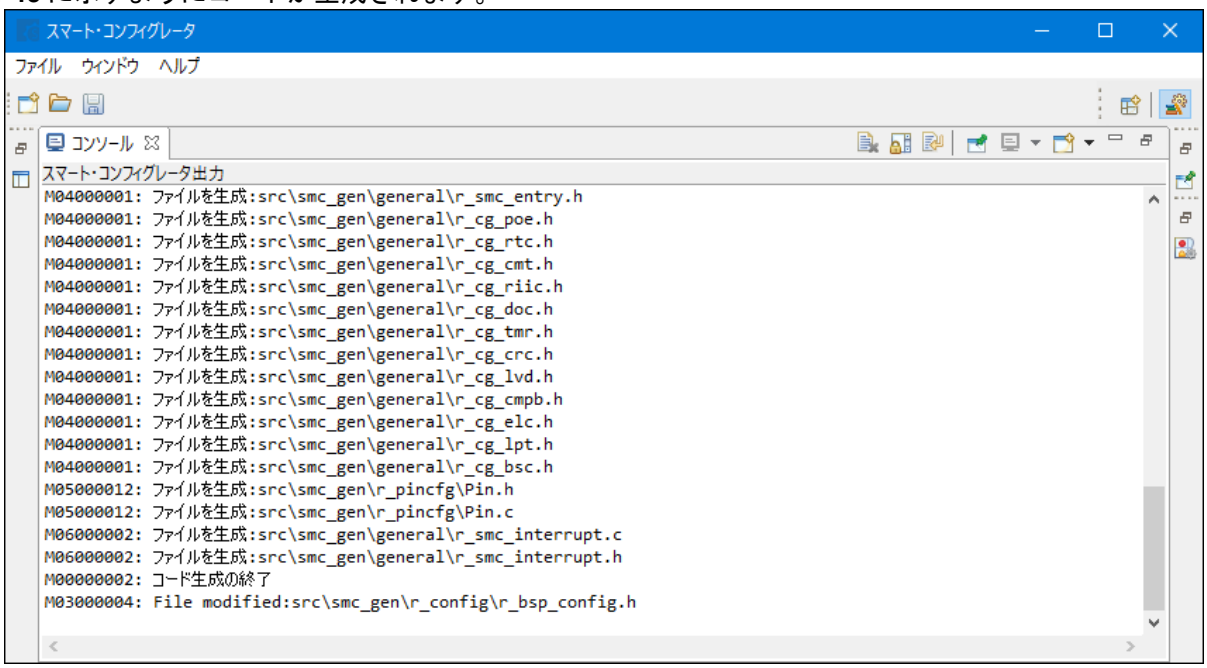


図 4-46: コード生成

コード生成を実行すると、CS+のプロジェクト作成時に生成されたスタートアップ関連のファイルが、スマート・コンフィグレータが生成したものに置き換えられます。**図 4-47** は、コード生成後のプロジェクト・ツリー中を示します。次の章ではこれらのファイルへユーザコードが追加され、新しいソースファイルがプロジェクトに追加されることで SC_Tutorial が完成します。

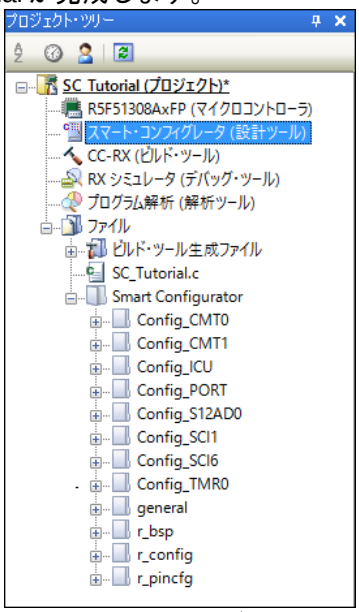
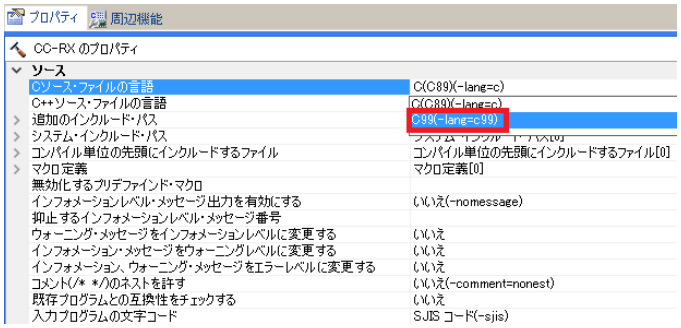
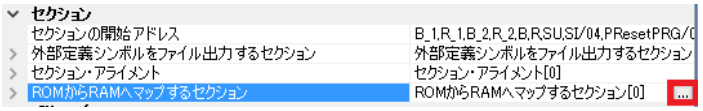
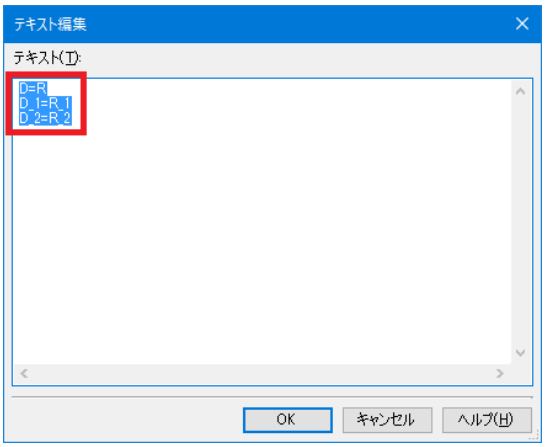


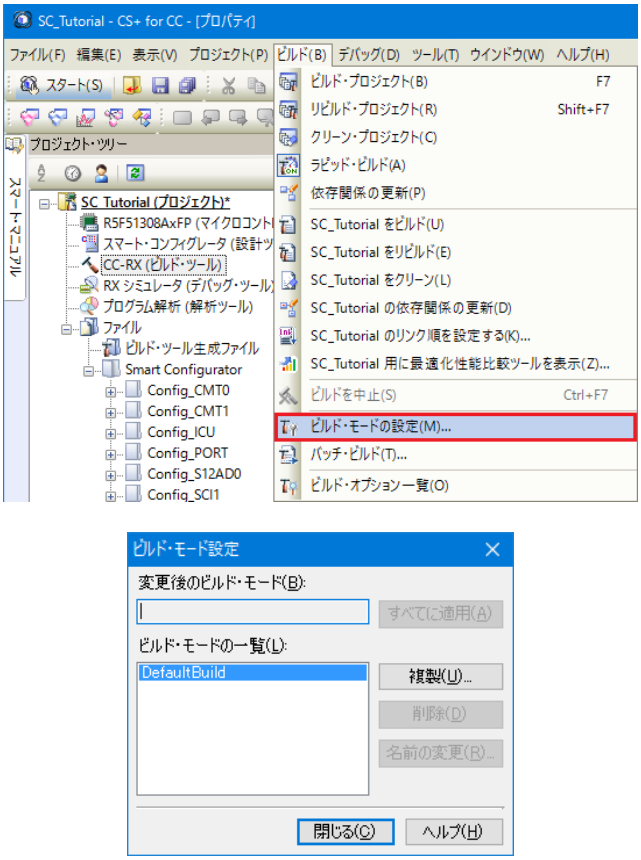
図 4-47: スマート・コンフィグレータ フォルダ構成

5. Tutorial プロジェクトの完成

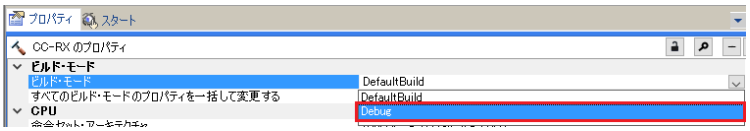
5.1 プロジェクト設定

• プロジェクト・ツリーから‘CC-RX (ビルド・ツール)’を選択すると、ビルドプロパティ画面が表示されます。 • CS+はプロジェクト用に‘DefaultBuild’と呼ばれる単一のビルドコンフィグレーションを作成します。デフォルト設定によって標準の最適化レベル 2 が設定されます。	
• プロパティ画面下にある‘コンパイル・オプション’タブを選択してください。‘C ソース・ファイルの言語’のプルダウンから‘C99(-lang=c99)’を選択してください。	
• プロパティ画面下にある‘リンク・オプション’タブを選択してください。‘ROM から RAM へマップするセクション’に 3 つのセクションを追加します。画面右端にある<...ボタンを選択してください。	
• テキスト編集ダイアログが現れます。以下のテキストを入力してください。 D=R D_1=R_1 D_2=R_2 • これはリンカが C 変数に ROM アドレスではなく RAM を割り当てることを保証します。<OK>ボタンをクリックしてください。	

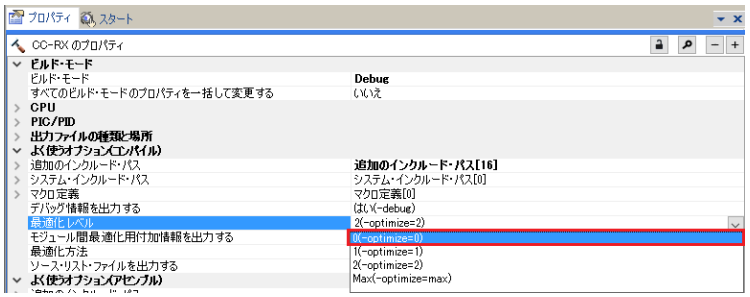
- ビルドメニューから‘ビルド・モードの設定(M)...’を選択してください。<複製>ボタンを選択して、入力フォームに‘Debug’を入力して<OK>ボタンを選択してください。
- ビルド・モードの一覧に複製した‘Debug’ビルド・モードが追加されたら、<閉じる>を選択してください。



- ビルドプロパティ画面に戻り画面下の‘共通オプション’タブをクリックしてください。‘ビルド・モード’のプルダウンから先ほど追加した‘Debug’を選択してください。



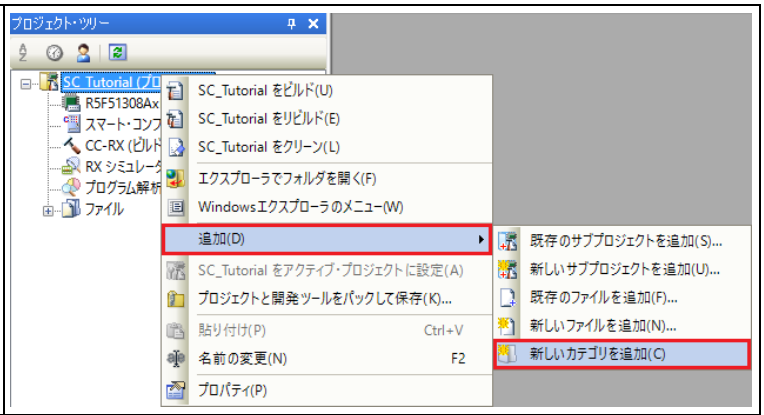
- 良く使うオプションの‘最適化レベル’のプルダウンから‘0(-optimize=0)’を選択していください。これにより、‘Debug’ビルド・モードではコードの最適化が行われません。



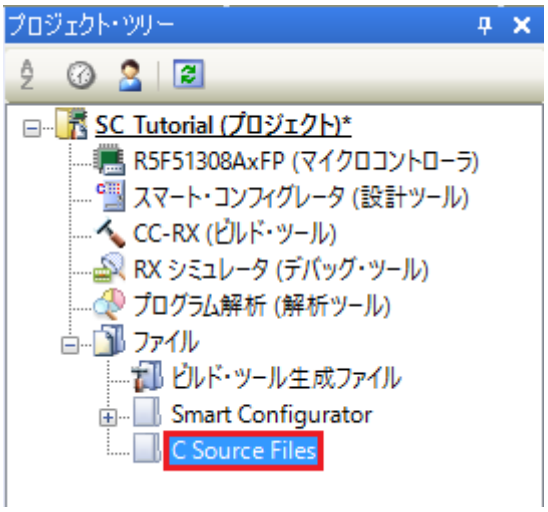
- RSK の全てのサンプルコードプロジェクトは 3 つのビルド・モード (DefaultBuild、Debug、Release)を選択できるようになっています。
- ビルド・モードに‘Debug’を追加したように、‘DefaultBuild’を複写して‘Release’ビルド・モードを追加してください。‘Release’の最適化レベルは初期設定のレベル 2 にしてください。次に、‘デバッグ情報を出力する’のプルダウンから‘いいえ(-nodebug)’を選択してください。
- ‘Release’ビルド・モードの追加、設定が完了したらビルド・モードを‘Debug’に選択し直してください。
- メニューバーの‘ファイル(F)’から‘すべてを保存(L)’を選択して、プロジェクトを保存してください。

5.2 フォルダの追加

- 新しいソースファイルをプロジェクトに追加する前に、プロジェクト・ツリーにカテゴリフォルダを 2 つ作成します。
- プロジェクト・ツリー内の SC_Tutorial プロジェクトを右クリックして、‘追加’->‘新しいカテゴリを追加’を選択してください。



- 新しく追加したフォルダ名を、‘C Source Files’に変更してください。
- 同様の手順でカテゴリフォルダを作成して、フォルダ名を‘Dependencies’に変更してください。



5.3 スマート・コンフィグレータ使用上の注意事項

スマート・コンフィグレータを使用してビルドを実行した際に、**図 5-1** の警告文が表示される場合があります。

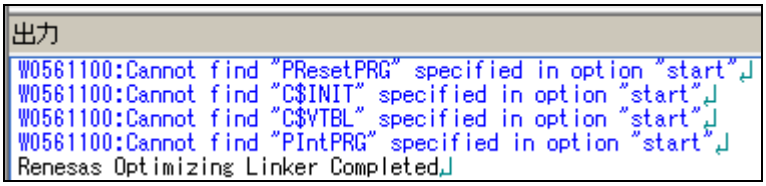
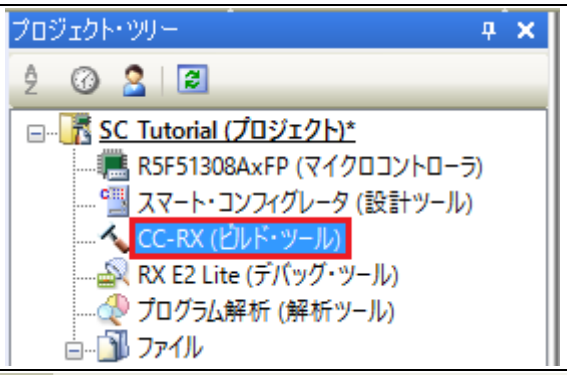
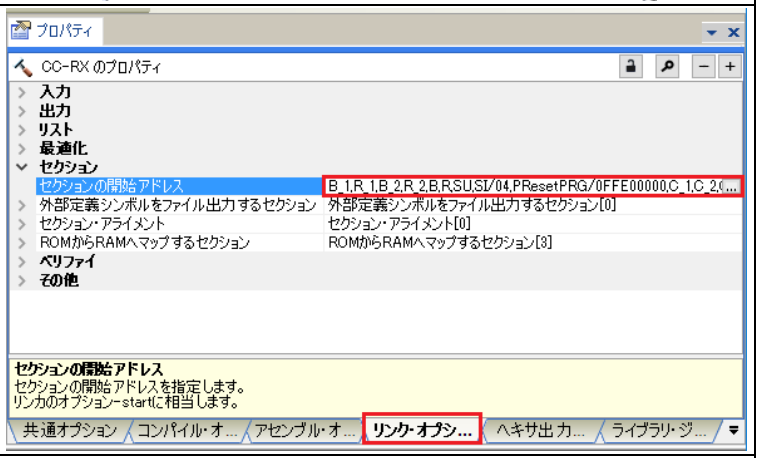
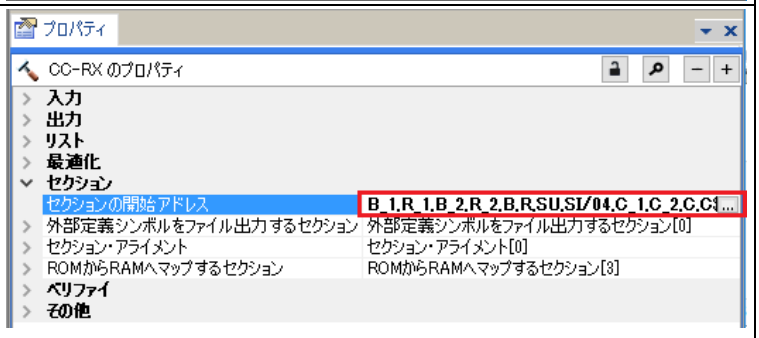


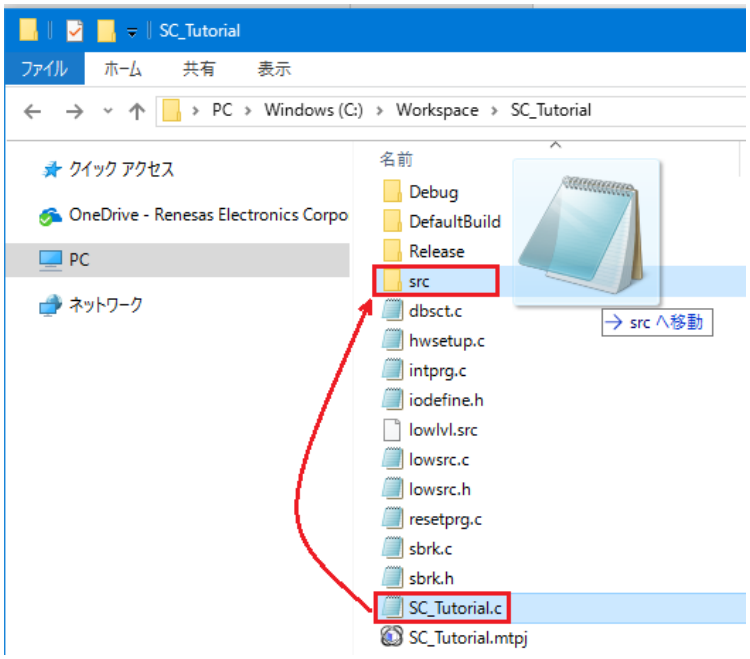
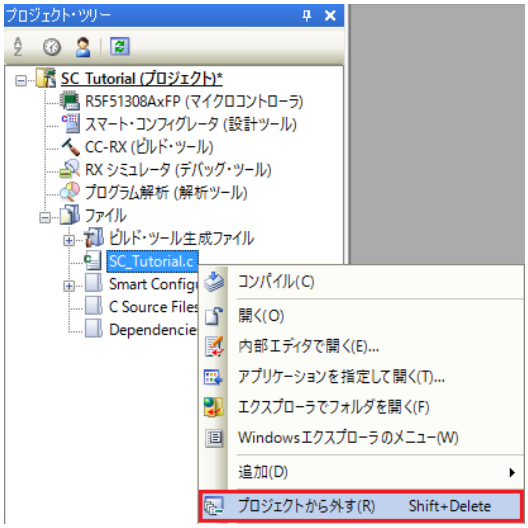
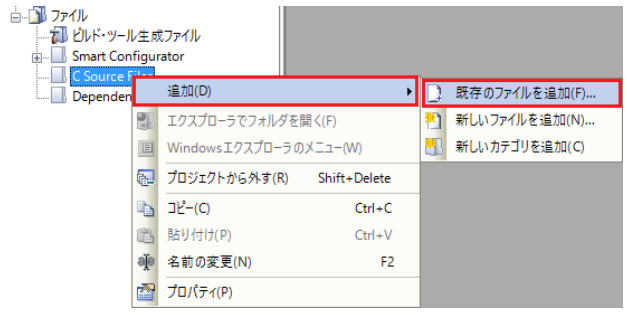
図 5-1: CS+出力ウィンドウ

使用されないセクションが残っているために表示される警告文であり、本マニュアルで作成する SC_Tutorial において動作上問題はありません。警告文を表示させないようにする場合は、以下の手順に従って設定してください。

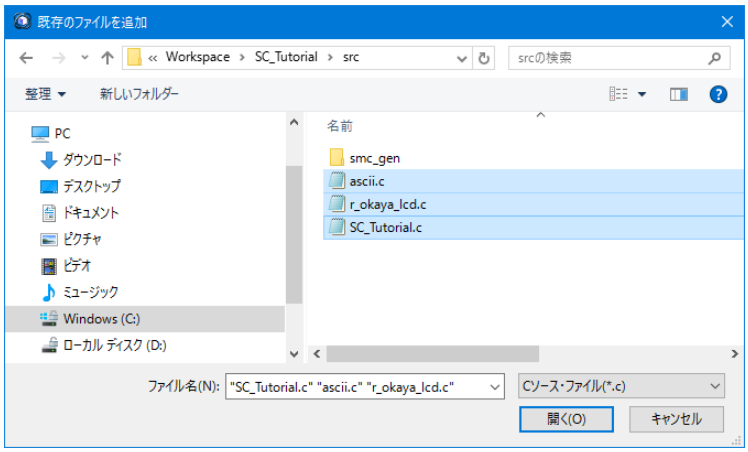
• プロジェクト・ツリーの CC-RX(ビルド・ツール)をダブルクリックしてください。	
• プロパティ画面下にある'リンク・オプション'タブを選択してください。 • 'セクションの開始アドレス'にセクションを設定します。	
• セクションの開始アドレスを以下で上書きしてください。 B_1,R_1,B_2,R_2,B,R,SU,SI/04,C_1,C_2,C,C$*,D*,W*,L,P*/0FFF80000,FIXEDVECT/0FFFFFF80	

5.4 LCD パネルコードの統合

Pmod LCD の API 機能は、RSK とともに提供されます。クイックスタートガイドの手順に従って作成した Tutorial プロジェクトのフォルダを参照してください。フォルダ内の‘ascii.h’、‘r_okaya_lcd.h’、‘ascii.c’、‘r_okaya_lcd.c’を C:¥Workspace¥SC_Tutorial¥src にコピーしてください。
次に、以下の手順に従って‘SC_Turorial.c’ファイルを移動および CS+で‘C Source Files’フォルダに‘ascii.c’、‘r_okaya_lcd.c’を追加、‘Dependencies’フォルダに‘ascii.h’、‘r_okaya_lcd.h’を追加してください。

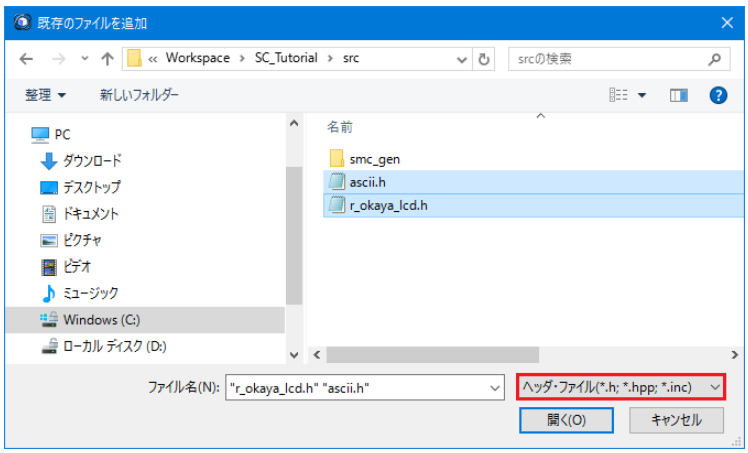
• ‘SC_Tutorial.c’ファイルを‘C:¥Workspace¥SC_Tutorial’から‘C:¥Workspace¥SC_Tutorial¥src’へ移動してください。	
• プロジェクト・ツリーの‘SC_Tutorial.c’を右クリックして、‘プロジェクトから外す(R)’をクリックしてください。	
• ‘C Source Files’フォルダを右クリックして、‘追加’ -> ‘既存のファイルを追加’を選択してください。 •	

- 追加するファイル（ascii.c、r_okaya_lcd.c、SC_Tutorial）を、C:¥Workspace¥SC_Tutorial¥src から選択してください。

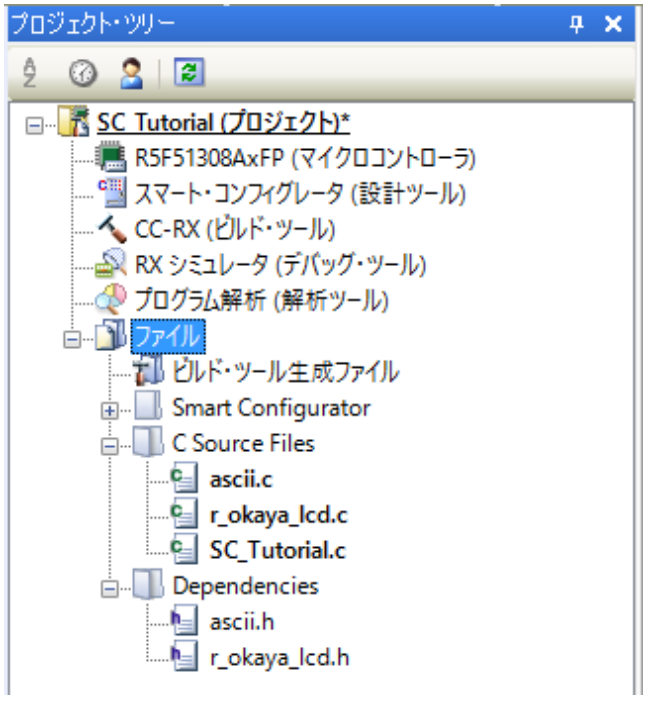


- 同様に、‘Dependencies’フォルダに ascii.h と r_okaya_lcd.h を追加してください。

注：ヘッダ・ファイル(*.h; *hpp; *.inc)を選択してください。



- プロジェクト・ツリーがスクリーンショットと同じになっていることを確認してください。



カスタムコードを加える場合、以下に示すコメント文の間にカスタムコードを加えてください。

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

‘_xxxx_’は特定のエリアで異なります。たとえば、インクルードファイルを定義する箇所では‘include’、ユーザコード記載箇所では‘function’、グローバル変数を定義する箇所では‘global’と記述されています。
コメント文の間にカスタムコードを加えることにより、コード生成による上書きから保護できます。

CS+ プロジェクトのプロジェクト・ツリーの‘Smart Configurator/general’フォルダを展開して、‘r_cg_userdefine.h’をダブルクリックして開いてください。次に、#define をコメント文の間に以下のように追加してください。

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

#define TRUE            (1)
#define FALSE           (0)

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

CS+プロジェクトのプロジェクト・ツリーの‘C Source Files’フォルダの‘SC_Tutorial.c’をダブルクリックして開いてください。次に、ファイルの main 関数の上に以下を追加してください。

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_cg_userdefine.h"
```

main 関数までスクロールしてください。ハイライトで明示した箇所を main 関数のユーザコードエリアコメント文の間に以下のようにコードを追加してください。

```
void main(void)
{
    /* Initialize the debug LCD */
    R_LCD_Init();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX130-512KB ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");
    while (1U)
    {
        ;
    }
}
```

5.4.1 SPI コード

Pmod LCD はセクション 4.4.7 で設定した SPI マスタ送信によって制御されます。CS+プロジェクトのプロジェクト・ツリーの‘Smart Configurator/Config_SCI6’フォルダを展開して、‘Config_SCI6.h’をダブルクリックして開いてください。次に、ファイル最後尾の‘function’ユーザコードエリアコメント文の間に以下のように追加してください。

```
/* 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、‘Config_SCI6_user.c’を開いて‘global’ユーザコードエリアコメント文の間に以下のように追加してください。

```
/* 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 */
```

SCI6 の送信コールバック関数のユーザエリアコメント文の間に以下のように追加してください。

```
void r_Config_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 */
}
```

ファイル最後尾のユーザコードエリアコメント文の間に以下のように追加してください。

```
/* 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_Config_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
***********************************************************************/
```

この関数は LCD への SPI 送信でフロー制御を実行するために送信完了コールバック関数を使い、LCD パネルコード中で API として使用します。

5.4.2 TMR コード

セクション 4.4.2 で設定した TMR0 は LCD 制御においてタイミングを合わせるためのディレイタイマとして使用されます。CS+プロジェクトのプロジェクト・ツリーの‘Smart Configurator/Config_TMR0’フォルダを展開して、‘Config_TMR0.h’ファイルをダブルクリックして開いて、ユーザエリアコメント文の間に以下のように追加してください。

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

void R_TMR_MsDelay(const uint16_t millisec);

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

‘Config_TMR0_user.c’ファイルを開いてファイル先頭付近の‘global’ユーザコードエリアコメント文の間に以下のように追加してください。

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

static volatile uint8_t one_ms_delay_complete = FALSE;

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

画面をスクロールして r_Config_TMR0_cmia0_interrupt 関数のユーザコードエリアコメント文の間に以下のように追加してください。

```
static void r_Config_TMR0_cmia0_interrupt(void)
{
    /* Start user code for r_Config_TMR0_cmia0_interrupt. Do not edit comment generated here */

    one_ms_delay_complete = TRUE;

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

次に、ファイルの最後尾のユーザコードエリアコメント文の間に以下のように追加してください。

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

/***********************************************************************
* Function Name: R_TMR_MsDelay
* Description  : Uses TMR0 to wait for a specified number of milliseconds
* Arguments    : uint16_t millisecs, number of milliseconds to wait
* Return Value : None
***********************************************************************/
void R_TMR_MsDelay (const uint16_t millisec)
{
    uint16_t ms_count = 0;

    do
    {
        R_Config_TMR0_Start();
        while (FALSE == one_ms_delay_complete)
        {
            /* Wait */
        }
        R_Config_TMR0_Stop ();
        one_ms_delay_complete = FALSE;
        ms_count++;
    } while (ms_count < millisec);
}
/***********************************************************************
End of function R_TMR_MsDelay
***********************************************************************/
```

メニューバーの‘ビルド’から‘ビルド・プロジェクト’または‘F7’キーを押してエラーがないことを確認してください。
6 章のデバッガ設定を行っていれば次の動作を確認できます。Pmod LCD の 1 行目に‘RSKRX130-512KB’、2 行目に‘Tutorial’、3 行目に‘Press Any Switch’が表示されます。

5.5 スイッチコードの統合

スイッチの API 機能は、RSK とともに提供されます。クイックスタートガイドの手順に従って作成した Tutorial プロジェクトのフォルダを参照してください。フォルダ内の‘rskrx130_512kbdef.h’、 ‘r_rsk_switch.h’、‘r_rsk_switch.c’を C:¥Workspace¥SC_Tutorial¥src にコピーしてください。
次に、コピーしたファイルをセクション 5.4 のようにプロジェクト・ツリーの‘C Source Files’フォルダに‘r_rsk_switch.c’を追加、‘Dependencies’フォルダに‘rskrx130_512kbdef.h’と‘r_rsk_switch.h’を追加してください。

スイッチコードでは、スイッチの ON/OFF を検知する割り込み機能とデバウンス用のタイマ機能を使用します。そのため、セクション 4.4.3 およびセクション 4.4.4 で設定されたファイル‘Config_ICU.h’、‘Config_ICU.c’ 、 ‘Config_ICU_user.c’ 、 ‘Config_CMT0.h’ 、 ‘Config_CMT0.c’ 、 ‘Config_CMT0_user.c’ 、‘Config_CMT1.h’、 ‘Config_CMT1.c’、 ‘Config_CMT1_user.c’が必要となります。

5.5.1 割り込みコード

CS+ プロジェクトのプロジェクト・ツリーの‘Smart Configurator/Config_ICU’フォルダを展開して、‘Config_ICU.h’をダブルクリックして開いてください。次に、ファイル最後尾のユーザコードエリアコメント文の間に以下のように追加してください。

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

/* Function prototypes for detecting and setting the edge trigger of ICU_IRQ */
uint8_t R_ICU_IRQIsFallingEdge(const uint8_t irq_no);
void R_ICU_IRQSetFallingEdge(const uint8_t irq_no, const uint8_t set_f_edge);
void R_ICU_IRQSetRisingEdge(const uint8_t irq_no, const uint8_t set_r_edge);

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

次に、‘Config_ICU.c’を開いてファイル最後尾のユーザコメントエリアコメント文の間に以下のように追加してください。

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

/***********************************************************************
* Function Name: R_ICU_IRQIsFallingEdge
* Description  : This function returns 1 if the specified ICU_IRQ is set to
*                falling edge triggered, otherwise 0.
* Arguments    : uint8_t irq_no
* Return Value : 1 if falling edge triggered, 0 if not
***********************************************************************/
uint8_t R_ICU_IRQIsFallingEdge (const uint8_t irq_no)
{
    uint8_t falling_edge_trig = 0x0;

    if (ICU.IRQCR[irq_no].BYTE & _04_ICU_IRQ_EDGE_FALLING)
    {
        falling_edge_trig = 1;
    }

    return (falling_edge_trig);

}

/***********************************************************************
* End of function R_ICU_IRQIsFallingEdge
***********************************************************************/

/***********************************************************************
* Function Name: R_ICU_IRQSetFallingEdge
* Description  : This function sets/clears the falling edge trigger for the
*                specified ICU_IRQ.
* Arguments    : uint8_t irq_no
*                uint8_t set_f_edge, 1 if setting falling edge triggered, 0 if
*                clearing
* Return Value : None
***********************************************************************/
void R_ICU_IRQSetFallingEdge (const uint8_t irq_no, const uint8_t set_f_edge)
{
    if (1 == set_f_edge)
    {
        ICU.IRQCR[irq_no].BYTE |= _04_ICU_IRQ_EDGE_FALLING;
    }
    else
    {
        ICU.IRQCR[irq_no].BYTE &= (uint8_t) ~_04_ICU_IRQ_EDGE_FALLING;
    }
}

/***********************************************************************
* End of function R_ICU_IRQSetFallingEdge
***********************************************************************/

/***********************************************************************
* Function Name: R_ICU_IRQSetRisingEdge
* Description  : This function sets/clear the rising edge trigger for the
*                specified ICU_IRQ.
* Arguments    : uint8_t irq_no
*                uint8_t set_r_edge, 1 if setting rising edge triggered, 0 if
*                clearing
* Return Value : None
***********************************************************************/
void R_ICU_IRQSetRisingEdge (const uint8_t irq_no, const uint8_t set_r_edge)
{
    if (1 == set_r_edge)
    {
        ICU.IRQCR[irq_no].BYTE |= _08_ICU_IRQ_EDGE_RISING;
    }
    else
    {
        ICU.IRQCR[irq_no].BYTE &= (uint8_t) ~_08_ICU_IRQ_EDGE_RISING;
    }
}

/***********************************************************************
* End of function R_ICU_IRQSetRisingEdge
***********************************************************************/

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

次に、‘Config_ICU_user.c’を開いて‘include’ユーザコードエリアコメント文の間に以下のように追加してください。

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

/* Defines switch callback functions required by interrupt handlers */
#include "r_rsk_switch.h"

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

同ファイルの r_Config_ICU_irq1_interrupt 関数内のユーザコードエリアコメント文の間に以下のように追加してください。

```
    /* Start user code for r_Config_ICU_irq1_interrupt. Do not edit comment generated here */

    /* Switch 1 callback handler */
    R_SWITCH_IsrCallback1();

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

同ファイルの r_Config_ICU_irq2_interrupt 関数内のユーザコードエリアコメント文の間に以下のように追加してください。

```
    /* Start user code for r_Config_ICU_irq2_interrupt. Do not edit comment generated here */

    /* Switch 2 callback handler */
    R_SWITCH_IsrCallback2();

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

5.5.2 デバウンス用タイマコード

‘Smart Configurator/Config_CMT0’フォルダを展開して、‘Config_CMT0_user.c’を開いて‘include’ユーザコードエリアコメント文の間に以下のように追加してください。

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

/* Defines switch callback functions required by interrupt handlers */
#include "r_rsk_switch.h"

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

同ファイルの r_Config_CMT0_cmi0_interrupt 関数内のユーザコードエリアコメント文の間に以下のように追加してください。

```
    /* Start user code for r_Config_CMT0_cmi0_interrupt. Do not edit comment generated here */

    /* Stop this timer - we start it again in the de-bounce routines */
    R_Config_CMT0_Stop();

    /* Call the de-bounce call back routine */
    R_SWITCH_DebounceIsrCallback();

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

‘Smart Configurator/Config_CMT1’フォルダを展開して、‘Config_CMT1_user.c’を開いて‘include’ユーザコードエリアコメント文の間に以下のように追加してください。

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

/* Defines switch callback functions required by interrupt handlers */
#include "r_rsk_switch.h"

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

同ファイルの r_Config_CMT1_cmi1_interrupt 関数内のユーザコードエリアコメント文の間に以下のように追加してください。

```
    /* Start user code for r_Config_CMT1_cmi1_interrupt. Do not edit comment generated here */

    /* Stop this timer - we start it again in the de-bounce routines */
    R_Config_CMT1_Stop();

    /* Call the de-bounce call back routine */
    R_SWITCH_DebounceIsrCallback();

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

5.5.3 A/D コンバータコードとメインスイッチコード

チュートリアルコードでは、スイッチを押すことで A/D 変換が行われ、A/D 変換の結果を LCD に表示します。A/D 変換開始にセクション 4.4.8 で設定した A/D 変換開始トリガ(ADTRG0#入力端子)が使用され、CPU ボード上の SW3 に接続されています。また、SW1 と SW2 はソフトウェアトリガとして機能します。
CS+ プロジェクトのプロジェクト・ツリーの‘Smart Configurator/general’フォルダを展開して、‘r_cg_userdefine.h’をダブルクリックして開いてください。次に、ユーザコードエリアコメント文の間に以下のように追加してください。

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

#define TRUE            (1)
#define FALSE           (0)

extern volatile uint8_t g_adc_trigger;

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

‘C Source Files’フォルダの‘SC_Tutorial.c’をダブルクリックして開いてください。次に、ファイルの main 関数の上にハイライト表示されているコードを追加してください。

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_rsk_switch.h"
#include "Config_S12AD0.h"
#include "r_cg_userdefine.h"

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

/* 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);
```

main 関数内にハイライト表示されているスイッチモジュールコールバック機能関数と while 文の中にコードを以下のように追加してください。

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

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

    /* Initialize the debug LCD */
    R_LCD_Init();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX130-512KB ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");

    /* Start the A/D converter */
    R_Config_S12AD0_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();

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Reset the flag */
            g_adc_trigger = FALSE;
        }

        /* SW3 is directly wired into the ADTRG0n pin so will
           cause the interrupt to fire */
        else if (TRUE == g_adc_complete)
        {
            /* Get the result of the A/D conversion */
            R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

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

続けて、以下の cb_switch_press 関数、get_adc 関数、lcd_display_adc 関数をファイル最後尾に追加してください。

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

/***********************************************************************
* Function Name : get_adc
* Description   : Reads the ADC result, converts it to a string and displays
*                 it on the LCD panel.
* Argument      : none
* Return value  : uint16_t adc value
***********************************************************************/
static uint16_t get_adc (void)
{
    /* A variable to retrieve the adc result */
    uint16_t adc_result;

    /* Stop the A/D converter being triggered from the pin ADTRG0n  */
    R_Config_S12AD0_Stop();

    /* Start a conversion */
    R_S12AD0_SWTriggerStart();

    /* Wait for the A/D conversion to complete */
    while (FALSE == g_adc_complete)
    {
        /* Wait */
    }

    /* Stop conversion */
    R_S12AD0_SWTriggerStop();

    /* Clear ADC flag */
    g_adc_complete = FALSE;

    R_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

    /* Set AD conversion start trigger source back to ADTRG0n pin */
    R_Config_S12AD0_Start();

    return (adc_result);
}
/***********************************************************************
* End of function get_adc
***********************************************************************/
```

```
/***********************************************************************
* Function Name : lcd_display_adc
* Description   : Converts adc result to a string and displays
*                 it on the LCD panel.
* Argument      : uint16_t adc result
* Return value  : none
***********************************************************************/
static void lcd_display_adc (const uint16_t adc_result)
{
    /* Declare a temporary variable */
    uint8_t a;

    /* Declare temporary character string */
    char    lcd_buffer[11] = " ADC: XXXH";

    /* Convert ADC result into a character string, and store in the local.
       Casting to ensure use of correct data type. */
    a = (uint8_t)((adc_result & 0x0F00) >> 8);
    lcd_buffer[6] = (char)((a < 0x0A) ? (a + 0x30) : (a + 0x37));
    a = (uint8_t)((adc_result & 0x00F0) >> 4);
    lcd_buffer[7] = (char)((a < 0x0A) ? (a + 0x30) : (a + 0x37));
    a = (uint8_t)(adc_result & 0x000F);
    lcd_buffer[8] = (char)((a < 0x0A) ? (a + 0x30) : (a + 0x37));

    /* Display the contents of the local string lcd_buffer */
    R_LCD_Display(3, (uint8_t *)lcd_buffer);

}
/***********************************************************************
* End of function lcd_display_adc
***********************************************************************/
```

‘Smart Configurator/Config_S12AD0’フォルダを展開して、‘Config_S12AD0.h’を開いて‘function’ユーザコードエリアコメント文の間に以下のように追加してください。

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

/* Flag indicates when A/D conversion is complete */
extern volatile uint8_t g_adc_complete;

/* Functions for starting and stopping software triggered A/D conversion */
void R_S12AD0_SWTriggerStart(void);
void R_S12AD0_SWTriggerStop(void);

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

‘Config_S12AD0.c’を開いてファイル最後尾のユーザコードエリアコメント文の間に以下を追加してください。

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

/***********************************************************************
* Function Name: R_S12AD0_SWTriggerStart
* Description  : This function starts the AD converter.
* Arguments    : None
* Return Value : None
***********************************************************************/
void R_S12AD0_SWTriggerStart(void)
{
    IR(S12AD, S12ADI0) = 0U;
    IEN(S12AD, S12ADI0) = 1U;
    S12AD.ADCSR.BIT.ADST = 1U;
}
/***********************************************************************
End of function R_S12AD0_SWTriggerStart
***********************************************************************/
```

```
/***********************************************************************
* Function Name: R_S12AD0_SWTriggerStop
* Description  : This function stops the AD converter.
* Arguments    : None
* Return Value : None
***********************************************************************/
void R_S12AD0_SWTriggerStop(void)
{
    S12AD.ADCSR.BIT.ADST = 0U;
    IEN(S12AD, S12ADI0) = 0U;
    IR(S12AD, S12ADI0) = 0U;
}
/***********************************************************************
End of function R_S12AD0_SWTriggerStop
***********************************************************************/

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

‘Config_S12AD0_user.c’を開いて‘global’ユーザコードエリアコメント文の間に以下のように追加してください。

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

/* Flag indicates when A/D conversion is complete */
volatile uint8_t g_adc_complete;

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

r_Config_S12AD0_interrupt 関数内のユーザコードエリアコメント文の間に以下のように追加してください。

```
static void r_Config_S12AD0_interrupt(void)
{
    /* Start user code for r_Config_S12AD0_interrupt. Do not edit comment generated here */

    g_adc_complete = TRUE;

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

メニューバーの‘ビルド’から‘ビルド・プロジェクト’または‘F7’キーを押してエラーがないことを確認してください。

6 章のデバッガ設定を行っていれば次の動作を確認できるようになります。いずれかのスイッチを押すことで、ポテンショメータから入力される電圧の A/D 変換値を LCD に表示します。
SCI ユーザコードを追加する場合は、再度ここから読み進めてください。

5.6 デバッグコードの統合

シリアルポートを介したデバッグトレースの API 機能は、RSK とともに提供されます。クイックスタートガイドの手順に従って作成した Tutorial プロジェクトのフォルダを参照してください。フォルダ内の‘r_rsk_debug.h’と‘r_rsk_debug.c’を C:¥Workspace¥SC_Tutorial¥src にコピーしてください。
次に、コピーしたファイルをセクション 5.4 のように‘C Source Files’フォルダに‘r_rsk_debug.c’を追加、‘Dependencies’フォルダに r_rsk_debug.h を追加してください。

‘r_rsk_debug.h’を開いて以下の記述があるか確認してください。

```
/* Macro for definition of serial debug transmit function - user edits this */
#define SERIAL_DEBUG_WRITE (R_SCI1_AsyncTransmit)
```

このマクロは‘r_rsk_debug.c’の中で参照されます。また、異なるデバッグインタフェースが使用される場合にデバッグ出力の再指定を容易にします。

5.7 UART コードの統合

5.7.1 SCI コード

CS+プロジェクトのプロジェクト・ツリーの‘Smart Configurator/Config_SCI1’フォルダを展開して、‘Config_SCI1.h’をダブルクリックして開いてください。次に、ファイル最後尾の‘function’ユーザコードエリアコメント文の間に以下のように追加してください。

```
/* 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_SCI1_AsyncTransmit(uint8_t * const tx_buf, const uint16_t tx_num);

/* Character is used to receive key presses from PC terminal */
extern uint8_t g_rx_char;

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

‘Config_SCI1_user.c’を開いて‘global’ユーザコードエリアコメント文の間に以下のように追加してください。

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

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

/* Flag used locally to detect transmission complete */
static volatile uint1_t sci1_txdone;

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

r_Config_SCI1_callback_transmitend 関数にユーザコードエリアコメント文の間に以下のように追加してください。

```
staic void r_Config_SCI1_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI1_callback_transmitend. Do not edit comment generated here */

    sci1_txdone = TRUE;

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

続けて、r_Config_sci1_callback_receiveend 関数にユーザコードエリアコメント文の間に以下のように追加してください。

```
void r_sci1_callback_receiveend(void)
{
    /* Start user code for r_Config_SCI1_callback_receiveend. Do not edit comment generated here */

    /* Check the contents of g_rx_char */
    if (('c' == g_rx_char) || ('C' == g_rx_char))
    {
        g_adc_trigger = TRUE;
    }

    /* Set up SCI1 receive buffer and callback function again */
    R_Config_SCI1_Serial_Receive((uint8_t *)&g_rx_char, 1);

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

ファイル最後尾の‘function’ユーザコードエリアコメント文の間に以下のように追加してください。

```
/***********************************************************************
* Function Name: R_SCI1_AsyncTransmit
* Description  : This function sends SCI1 data and waits for the transmit end flag.
* Arguments    : tx_buf -
*                    transfer buffer pointer
*                tx_num -
*                    buffer size
* Return Value : status -
*                    MD_OK or MD_ARGERROR
***********************************************************************/
MD_STATUS R_SCI1_AsyncTransmit (uint8_t * const tx_buf, const uint16_t tx_num)
{
    MD_STATUS status = MD_OK;

    /* clear the flag before initiating a new transmission */
    sci1_txdone = FALSE;

    /* Send the data using the API */
    status = R_Config_SCI1_Serial_Send(tx_buf, tx_num);

    /* Wait for the transmit end flag */
    while (FALSE == sci1_txdone)
    {
        /* Wait */
    }
    return (status);
}

/***********************************************************************
* End of function R_SCI1_AsyncTransmit
***********************************************************************/
```

5.7.2 メイン UART コード

‘C Source Files’フォルダの‘SC_Tutorial.c’をダブルクリックして開いてください。次に、ファイルの main 関数の上にハイライト表示されているコードを追加してください。

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "Config_S12AD0.h"
#include "Config_SCI1.h"
#include "r_cg_userdefine.h"

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

/* 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;
```

main 関数内に以下を追加してください。

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

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

    /* Initialize the debug LCD */
    R_LCD_Init();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX130-512KB ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");

    /* Start the A/D converter */
    R_Config_S12AD0_Start();

    /* Set up SCI1 receive buffer and callback function */
    R_Config_SCI1_Serial_Receive((uint8_t *)&g_rx_char, 1);

    /* Enable SCI1 operations */
    R_Config_SCI1_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();

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Increment the adc_count */
            if (16 == (++adc_count))
            {
                adc_count = 0;
            }
```

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

            /* Reset the flag */
            g_adc_trigger = FALSE;
        }
        /* SW3 is directly wired into the ADTRG0n pin so will
           cause the interrupt to fire */
        else if (TRUE == g_adc_complete)
        {
            /* Get the result of the A/D conversion */
            R_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

            /* Display the result on the LCD */
            lcd_display_adc(adc_result);

            /* Increment the adc_count */
            if (16 == (++adc_count))
            {
                adc_count = 0;
            }

            /* 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 */
}
```

次に、ファイル最後尾に以下のように追加してください。

```
/***********************************************************************
* Function Name : uart_display_adc
* Description   : Converts adc result to a string and sends it to the UART1.
* Argument      : uint8_t : adc_count
*                 uint16_t: adc result
* Return value  : none
***********************************************************************/
static void uart_display_adc (const uint8_t adc_count, const uint16_t adc_result)
{
    /* Declare a temporary variable */
    char a;

    /* Declare temporary character string */
    static char uart_buffer[] = "ADC xH Value: xxxH\r\n";

    /* Convert ADC result into a character string, and store in the local.
       Casting to ensure use of correct data type. */
    a = (char)(adc_count & 0x000F);
    uart_buffer[4] = (char)((a < 0x0A) ? (a + 0x30) : (a + 0x37));
    a = (char)((adc_result & 0x0F00) >> 8);
    uart_buffer[14] = (char)((a < 0x0A) ? (a + 0x30) : (a + 0x37));
    a = (char)((adc_result & 0x00F0) >> 4);
    uart_buffer[15] = (char)((a < 0x0A) ? (a + 0x30) : (a + 0x37));
    a = (char)(adc_result & 0x000F);
    uart_buffer[16] = (char)((a < 0x0A) ? (a + 0x30) : (a + 0x37));

    /* Send the string to the UART */
    R_DEBUG_Print(uart_buffer);

}

/***********************************************************************
* End of function uart_display_adc
***********************************************************************/
```

メニューバーの‘ビルド’から‘ビルド・プロジェクト’または‘F7’キーを押してエラーがないことを確認してください。

動作を確認する前に、コンピュータの USB ポートと CPU ボード上の USB シリアルポート（シルク印字‘G1CUSB0’）を USB ケーブルで接続する必要があります。はじめて接続した場合、コンピュータの画面にドライバのインストールメッセージが表示され、自動的にデバイスドライバはインストールされます。デバイスマネージャ上のポート(COM と LPT)に‘RSK USB Serial Port (COMx)’が現れますので、COM ポート番号を確認し、ターミナルソフトを起動して確認した COM ポート番号の設定を行ってください。

ターミナルソフトの設定は SCI1 と同じ設定にしてください(セクション 4.4.6)。6 章のデバッガ設定を行っていれば次の動作を確認できます。プログラム動作開始後、いずれかのスイッチを押すかターミナルソフト画面でキーボードの‘c’を押すことで、ポテンショメータから入力される電圧の A/D 変換値を LCD とターミナルソフト画面に表示します。
LED ユーザコードを追加する場合は、再度ここから読み進めてください。

5.8 LED コードの統合

‘C Source Files’フォルダの‘SC_Tutorial.c’をダブルクリックして開いてください。次に、ファイルの main 関数の上にハイライト表示されているコードを追加してください。

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "rskrx130_512kbdef.h"
#include "Config_S12AD0.h"
#include "Config_SCI1.h"
#include "r_cg_userdefine.h"

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

/* 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);
```

main 関数内のユーザコードエリアコメント文の間に以下を追加してください。

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

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

    /* Initialize the debug LCD */
    R_LCD_Init();

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX130-512KB ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");

    /* Start the A/D converter */
    R_Config_S12AD0_Start();
```

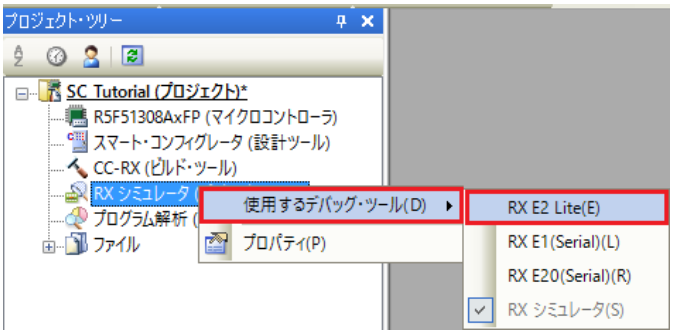
```
    /* Set up SCI1 receive buffer and callback function */
    R_Config_SCI1_Serial_Receive((uint8_t *)&g_rx_char, 1);
    /* Enable SCI1 operations */
    R_Config_SCI1_Start();
    while (1U)
    {
        uint16_t adc_result;
        /* Wait for user requested A/D conversion flag to be set (SW1 or SW2) */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an A/D conversion */
            adc_result = get_adc();
            /* Display the result on the LCD */
            lcd_display_adc(adc_result);
            /* Increment the adc_count and display using the LEDs */
            if (16 == (++adc_count))
            {
                adc_count = 0;
            }
            led_display_count(adc_count);
            /* Send the result to the UART */
            uart_display_adc(adc_count, adc_result);
            /* Reset the flag */
            g_adc_trigger = FALSE;
        }
        /* SW3 is directly wired into the ADTRG0n pin so will
           cause the interrupt to fire */
        else if (TRUE == g_adc_complete)
        {
            /* Get the result of the A/D conversion */
            R_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);
            /* Display the result on the LCD */
            lcd_display_adc(adc_result);
            /* Increment the adc_count and display using the LEDs */
            if (16 == (++adc_count))
            {
                adc_count = 0;
            }
            led_display_count(adc_count);
            /* Send the result to the UART */
            uart_display_adc(adc_count, adc_result);
            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
}
```

次に、ファイル最後尾に以下のように追加してください。

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

メニューバーの‘ビルド’から‘ビルド・プロジェクト’または‘F7’キーを押してエラーがないことを確認してください。
6 章のデバッガ設定を行っていれば次の動作を確認できます。基本動作はセクション 5.7.2 までの内容と同じですが、adc_count（A/D 変換カウント）をユーザ LED でバイナリ形式表示します。

6. プロジェクトのデバッグ設定

• RX シミュレータ（デバッグ・ツール）を右クリックし、RX E2 Lite を選択してください。	
• RX E2 Lite を右クリックし、プロパティを選択してください。 • 接続用設定タブをクリックしてメイン・クロック周波数を設定してください。 メイン・クロック周波数[MHz]：8.0000 動作周波数[MHz]：32.0000 • エミュレータから電源供給をする(最大 200mA)：はい	
• E2 Lite をコンピュータの USB ポートに接続してください。Pmod LCD を PMOD1 コネクタに接続してください。 • ツールバーの‘ダウンロード’ボタンをクリックしてください。	

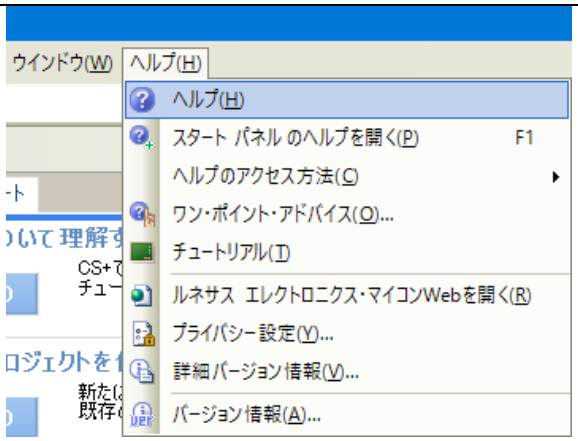
7. チュートリアルコードの実行

7.1 コードの実行

プログラムが CPU ボード上のマイクロコントローラにダウンロードされると、プログラムを実行できます。現在のプログラムカウンタ位置からプログラムを始めるため'実行'ボタンをクリックしてください。	

8. 追加情報

サポート

CS+の使用方法等の詳細情報は、CS+のヘルプメニューを参照してください。	

RX130 マイクロコントローラに関する詳細情報は、RX130 ユーザーズマニュアルハードウェア編を参照してください。

アセンブリ言語に関する詳細情報は、RX ファミリユーザーズマニュアルソフトウェア編を参照してください。

オンラインの技術サポート、情報等は https://www.renesas.com/rskrx130-512kb より入手可能です。

オンライン技術サポート

技術関連の問合せは、https://www.renesas.com/support/contact.html を通じてお願いいたします。

ルネサスのマイクロコントローラに関する総合情報は、https://www.renesas.com/より入手可能です。

商標

本書で使用する商標名または製品名は、各々の企業、組織の商標または登録商標です。

著作権

改訂記録	RSKRX130-512KB スマート・コンフィグレータチュートリアルマニュアル(CS+)

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2017.06.30	ー	初版発行

RSKRX130-512KB
スマート・コンフィグレータチュートリアルマニュアル(CS+)

発行年月日　2017 年 06 月 30 日　　Rev.1.00

発行　　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）

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/

RX130 グループ

R20UT3924JG0100

ルネサス エレクトロニクス株式会社