User's Manual

32

RX140 グループ

Renesas Starter Kit for RX140
スマート・コンフィグレータ　チュートリアルマニュアル
（e^2 studio）

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　2022.01

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。

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。

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 e^2 studio 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。

このマニュアルには、コードを生成して e^2 studio にインポートするための手順が段階的に明示していますが、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。
RX140 マイクロコントローラの操作に関する詳細は、「RX140 グループ ユーザーズマニュアル ハードウェア編」およびサンプルコード内に記載されています。また、RSK Web インストーラのセットアップ手順については「クイックスタートガイド」に記載されています。

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。

RSKRX140 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	Renesas Starter Kit for RX140 ユーザーズマニュアル	R20UT5026JG
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	Renesas Starter Kit for RX140 チュートリアルマニュアル	R20UT5030JG
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	Renesas Starter Kit for RX140 クイックスタートガイド	R20UT5031JG
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	Renesas Starter Kit for RX140 スマート・コンフィグレータ チュートリアルマニュアル	R20UT5032JG (本マニュアル)
回路図	CPU ボードの回路図	Renesas Starter Kit for RX140 CPU ボード回路図	R20UT5025EG
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	RX140 グループ　ユーザーズマニュアル　ハードウェア編	R01UH0905JJ

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
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
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
TFT	Thin Film Transistor	薄膜トランジスタ
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

この 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 e^2 studio 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ます。

1.2 特徴

本 RSK は以下の特徴を含みます：

- e^2 studio による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。
- 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 e^2 studio 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
- カスタムコードとの統合
- e^2 studio プロジェクトの構築

プロジェクトジェネレータは、選択可能な 2 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。:

- 'HardwareDebug'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、デバッグ情報出力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。

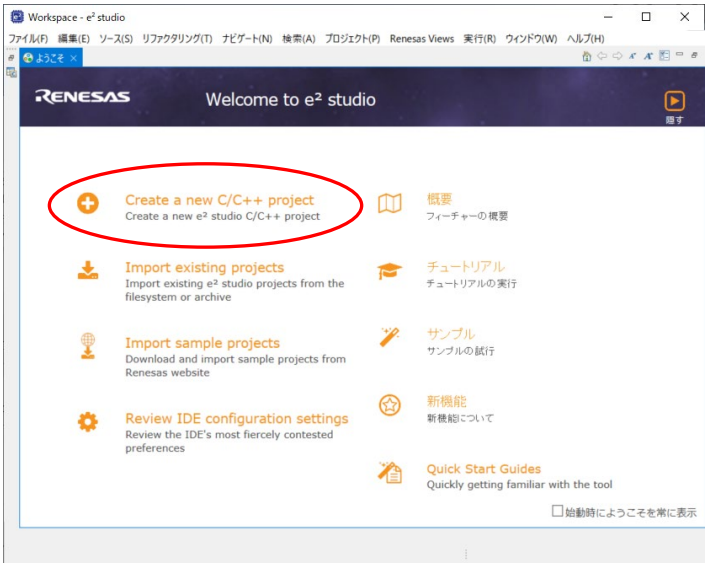
これらのチュートリアルは、RSK の使用方法の説明を目的とするものであり、e^2 studio、コンパイラまたは E2 エミュレータ Lite の入門書ではありません。これらに関する詳細情報は各関連マニュアルを参照してください。

3. e² studio プロジェクトの作成

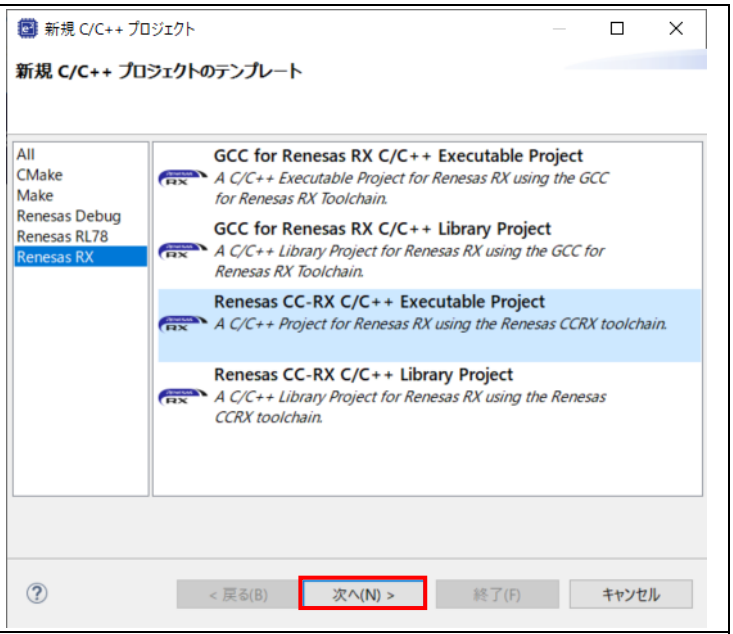
3.1 はじめに

この章では、RX140 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手順を説明し、スマート・コンフィグレータを使用して周辺機器ドライバコードを生成する準備を行います。このプロジェクト生成手順は、マイクロコントローラ特有のソース、プロジェクトおよび、デバッグファイルを作成するために必要です。

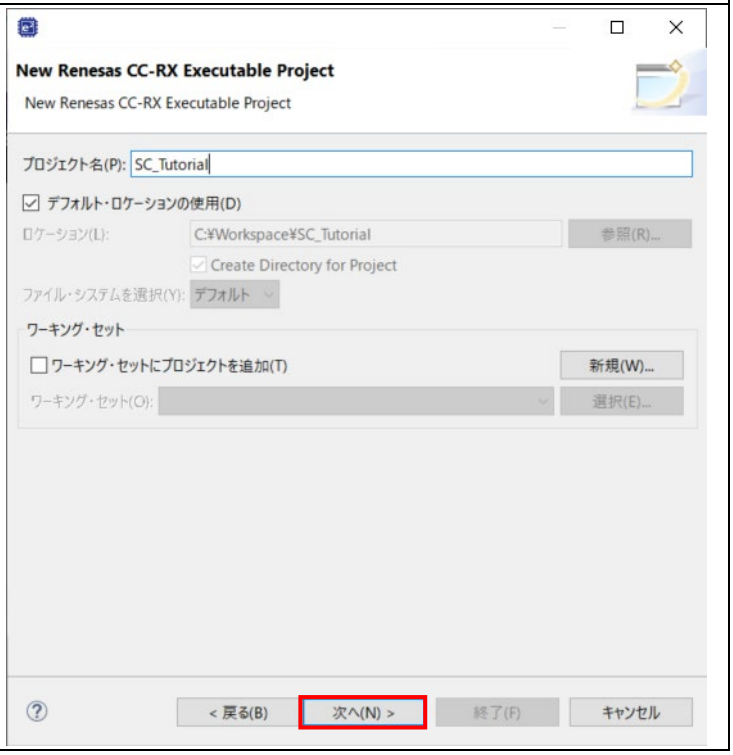
3.2 プロジェクトの作成

• e² studio を起動し、プロジェクトワークスペースに適したディレクトリを選択します。	
• Welcome to e² studio ページで、'Create a new C/C++ project'をクリックします。 （Welcome to e² studio ページは、'ヘルプ' -> 'ようこそ'からも開けます。）	

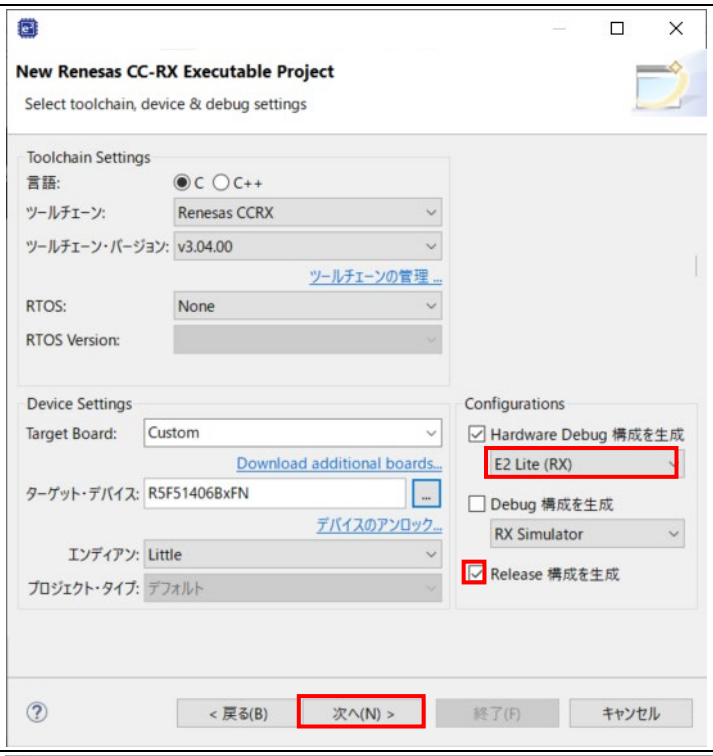
- ‘Templates for New C/C++ Project’ダイアログで、‘Renesas RX’ -> ‘Renesas CC-RX C/C++ Executable Project’を選択します。
- ‘次へ(N)’をクリックします。



- プロジェクト名’SC_Tutorial’を入力します。
  ‘次へ(N)’をクリックします。

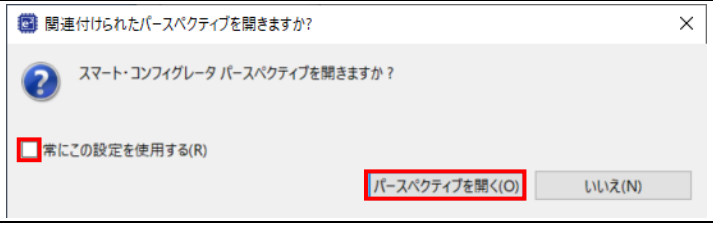


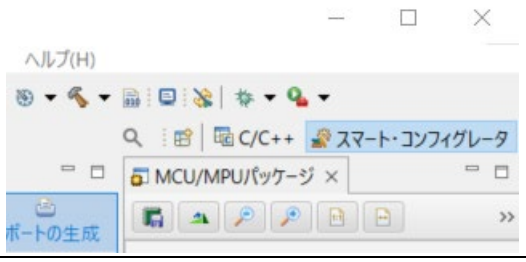
- ‘Select toolchain, device & debug settings’ ダイアログにて右図にあるオプションを選択します。
- ‘ツールチェイン:’にて‘Renesas CCRX’を選択します。
- ターゲット・デバイスはプルダウンから RX100 > RX140 > RX140 - 80 pin >の順で R5F51406BxFN を選択します。
- 使用するエミュレータはプルダウンから’E2 Lite(RX)’を選択、’Release 構成を生成’のチェックボックスをオンにします。
- ‘次へ(N)’をクリックします。



- ‘コーディング・アシストツールの選択’ダイアログにて、‘Use Smart Configurator’を選択します。
- ‘次へ(N)’をクリックします。

- 右図のポップアップメッセージを今後スキップするには、’常にこの設定を使用する’をオンにして、’パースペクティブを開く(O)’をクリックします。



• スマート・コンフィグレータが起動して、パースペクティブがスマート・コンフィグレータに切り替わります。	

4. 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めため GUI ツールです。スマート・コンフィグレータはターゲット・デバイス、クロック、ソフトウェアコンポーネント、ハードウェアリソース、およびプロジェクトの割り込みを視覚的に設定できます。

スマート・コンフィグレータによって生成されるコードは、特定の周辺機能ごとに 3 つのコードを生成します（「Config_xxx.h」、「Config_xxx.c」、「Config_xxx_user.c」）。例えば A/D コンバータの場合、周辺を表す xxx は'S12AD'と名付けられます。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。ただしこれらのファイルでは、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コードを追加する必要があります。

```
/* 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。
注）上記のカスタムコード領域外にコードを追加した場合、スマート・コンフィグレータでコード生成を再実行すると、コードが消失しますのでご注意ください。

本書の手順を踏むことで、ユーザは SC_Tutorial という e^2 studio 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、RSK Web インストーラ（https://www.renesas.com/rskrx140/install/e2）に含まれており、クイックスタートガイドの手順に従えば、e^2 studio にインポートできます。本書はスマート・コンフィグレータを使用して、オリジナルの e^2 studio プロジェクトを作成したいユーザのためのチュートリアルマニュアルです。

SC_Tutorial プロジェクトは、スイッチによる割り込み、A/D コンバータ、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）などのモジュールを使用し、仮想 COM ポートを介してターミナルソフトと RSK の Pmod LCD に A/D 変換結果を表示します。

このセクションでは、タブページ（ボード、クロック、コンポーネント、端子）のスマート・コンフィグレータの主要なユーザーインターフェイス機能のツアーおよび、プロジェクトのビルド」のデモにつづいて各周辺機能構成ページをガイドします。その後、スマート・コンフィグレータが提供するユーザコード領域に独自のコードを追加する過程など、テンプレート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

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、RX スマート・コンフィグレータ ユーザーガイド: e^2 studio 編を参照ください。
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・コンフィグレータの初期画面が**図 4-1** のように表示されます。

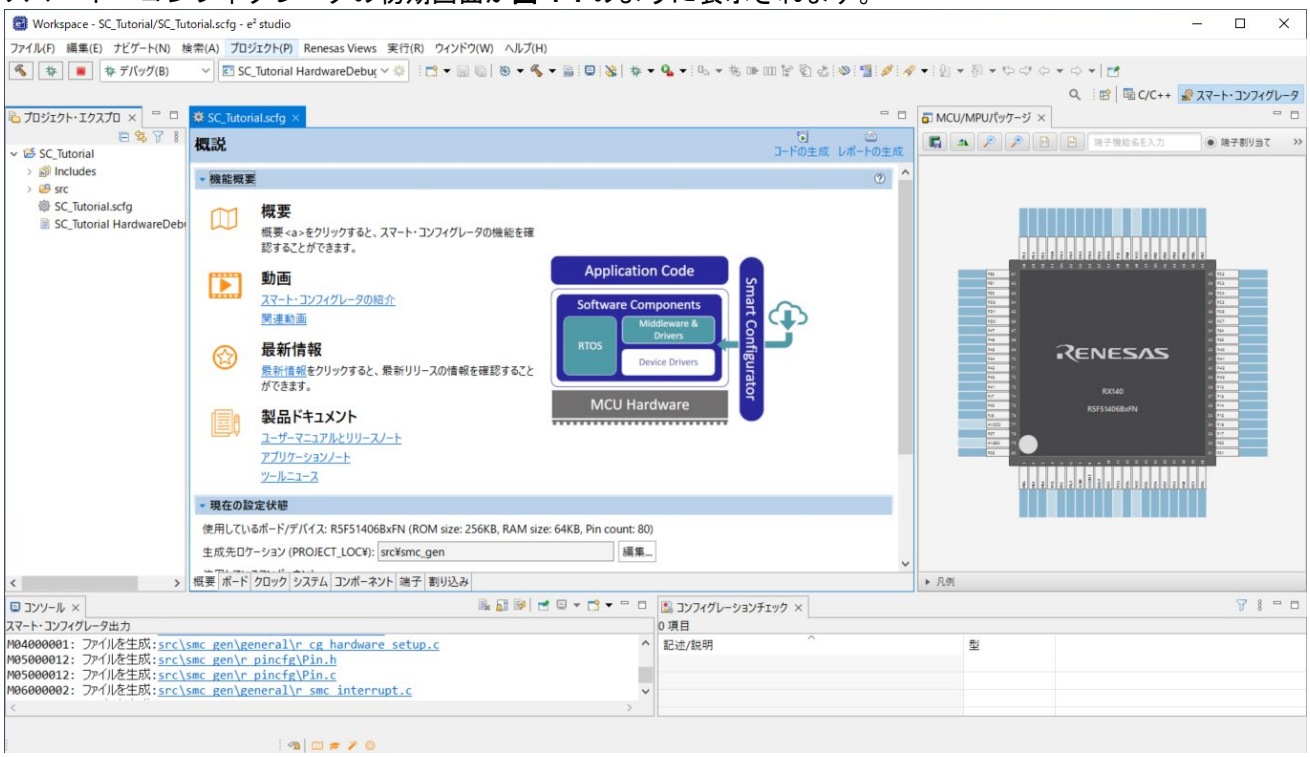


図 4-1 概要ページ

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、'コードの生成'ボタンをクリックすると、GUI で設定した内容のソースコードが生成されます。

4.3 ボードページ

ボードページでは、ボードおよびデバイスの種類を設定します。
‘ボード’タブをクリックすると、**図 4-2** のように表示されます。

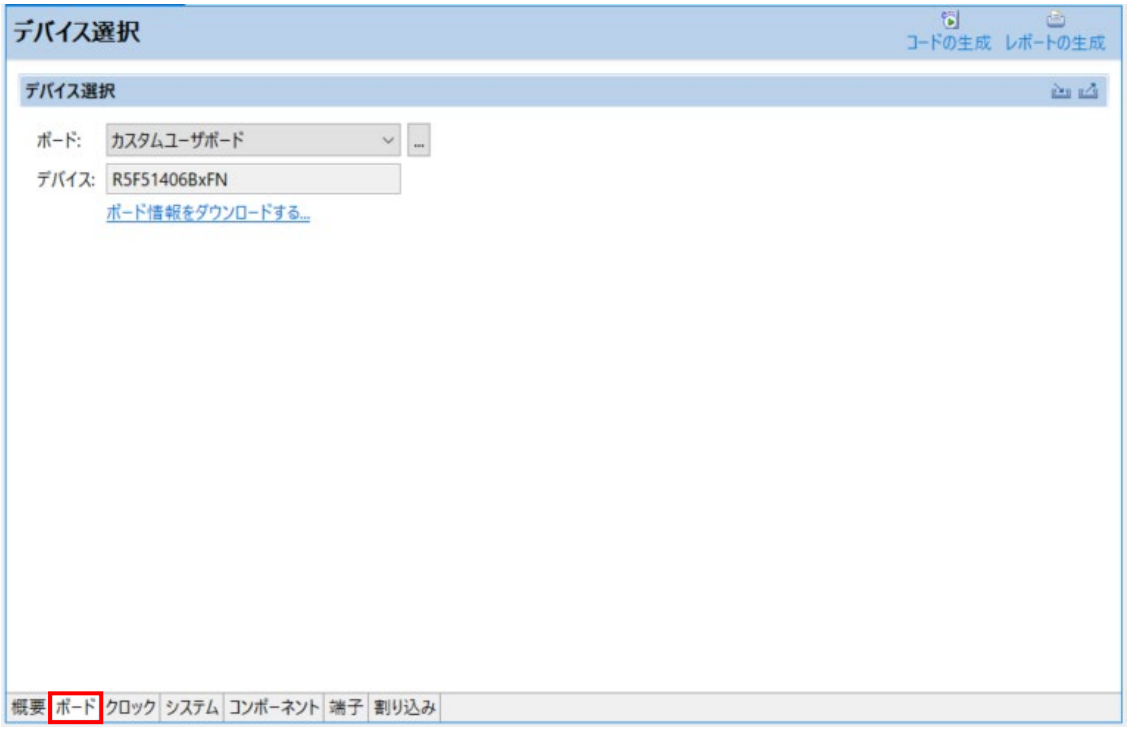


図 4-2 ボードページ

4.3.1 ボード設定

‘ボード：’は‘カスタムユーザボード’を選択していることを確認します。

図 4-3 ボードの選択

4.4 クロックページ

クロックページでは、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。クロック設定は、プロジェクト・ツリーの “\src\smc_gen\r_config”の”r_bsp_config.h”ファイルに反映されます。

4.4.1 クロック設定

スマート・コンフィグレータのクロック設定を**図 4-4** に示します。‘クロック’タブをクリックしてください。図のようにシステムクロックを設定します。チュートリアルでは、メイン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。

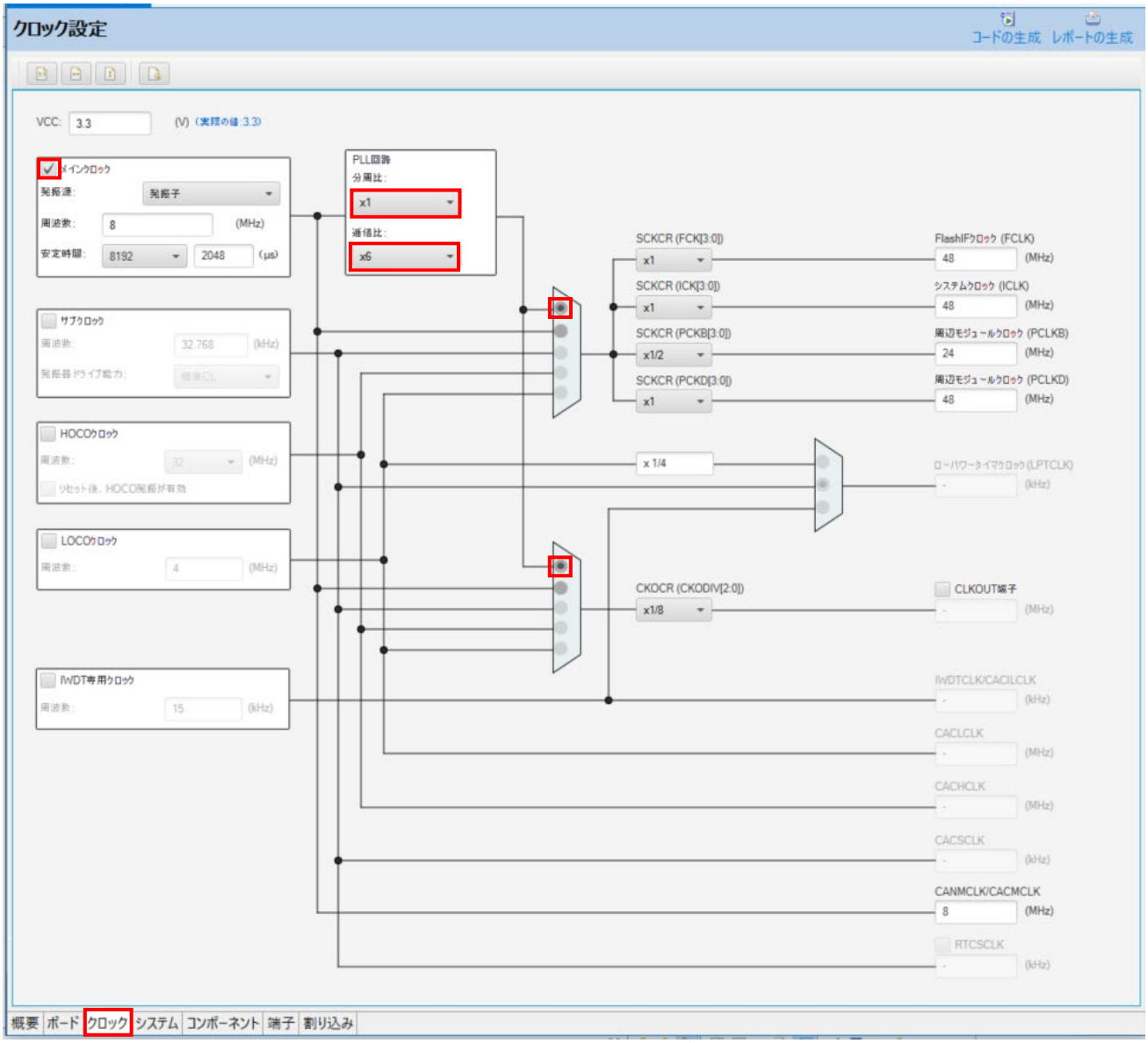


図 4-4 クロックページ

4.5 システム設定ページ

システムページでは、オンチップ・デバッグモードを設定します。

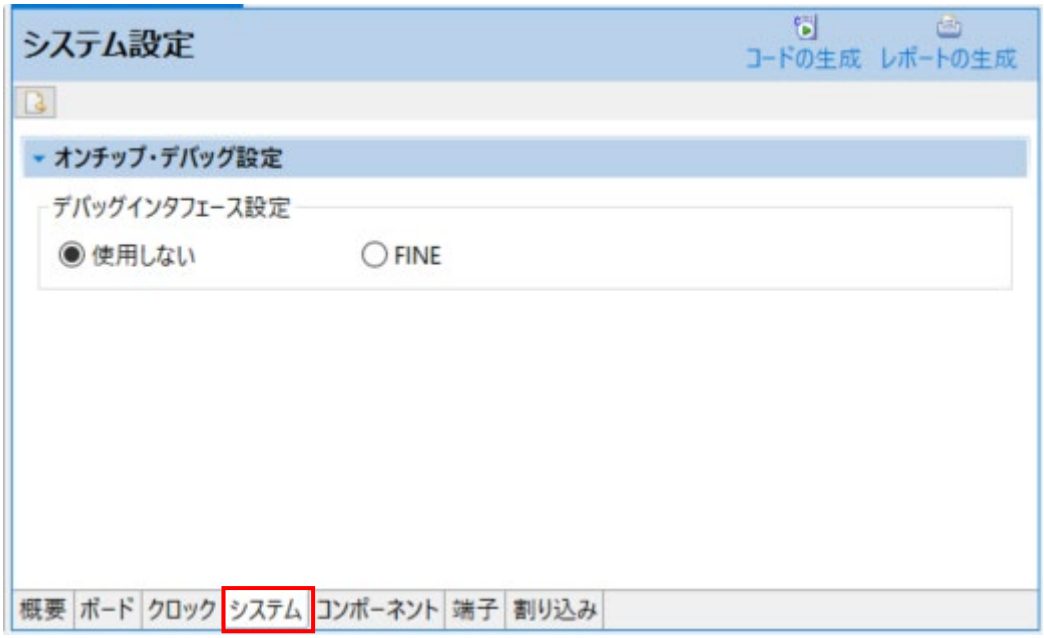


図 4-5 システムページ

4.5.1 オンチップ・デバッグ設定

オンチップ・デバッグ設定では、デバッグで使用するインタフェースを設定します。RSKRX140 の CPU ボードでは、**図 4-6** のように FINE を選択してください。

図 4-6 デバッグインタフェース設定

4.6 コンポーネントページ

ドライバとミドルウェアは Smart Configurator のソフトウェアコンポーネントとして処理されます。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。

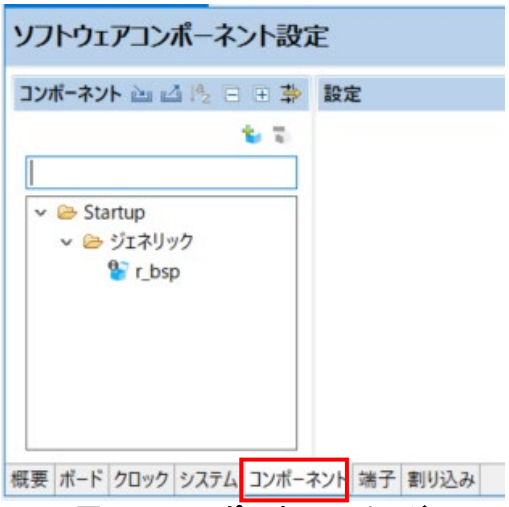


図 4-7 コンポーネントページ

4.6.1 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、Startup、Drivers、Middleware、Application、そして RTOS の 5 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。以下のサブセクションでは、Drivers のコンポーネントによるスイッチ入力の割り込み、タイマ、ADC、および SCI 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。

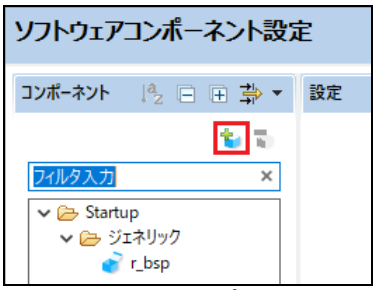


図 4-8 ソフトウェアコンポーネントの追加 (1)

‘ソフトウェアコンポーネントの選択’ダイアログ -> タイプは‘Drivers’を選択します。

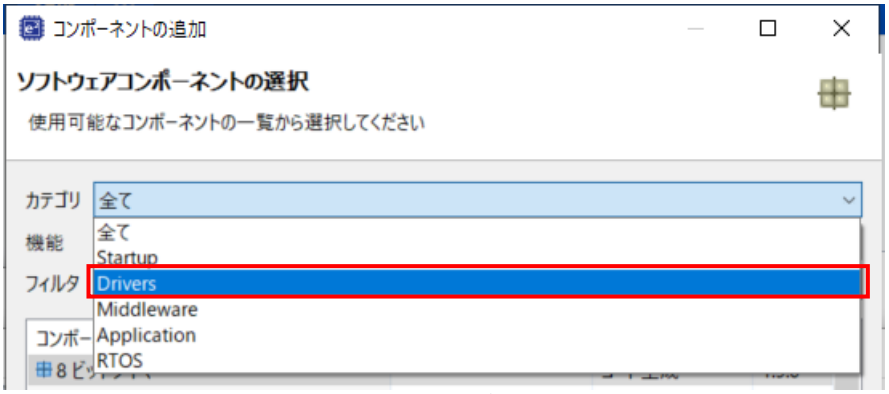


図 4-9 ソフトウェアコンポーネントの追加 (2)

4.6.2 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。**図 4-10** 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 4-10 8 ビットタイマ選択

‘選択したコンポーネントの新しい構成を追加します。**図 4-11** のように“リソース”は、“TMR0”を選択します。

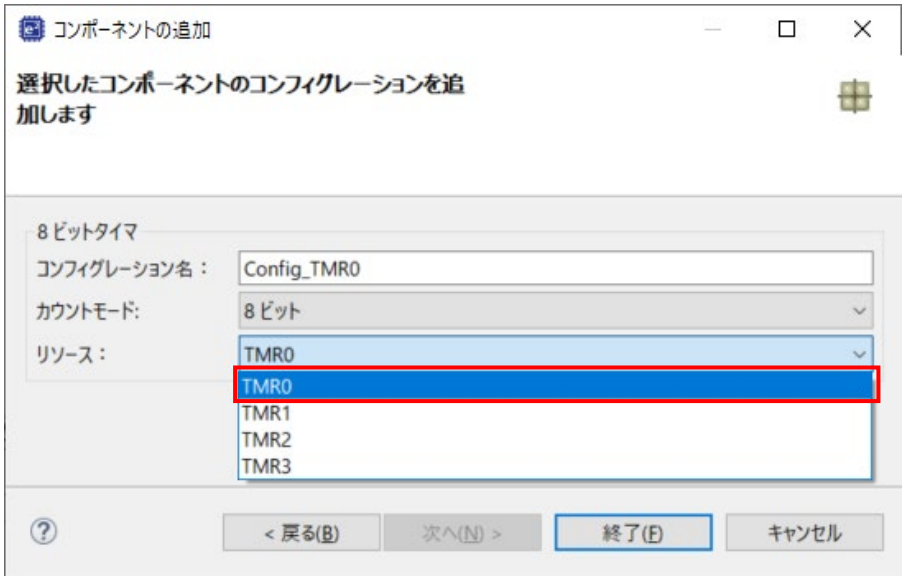


図 4-11 リソース選択 - TMR0

図 4-12 のように、コンフィグレーション名が“Config_TMR0”であることを確認し、’終了(F)’をクリックします。

図 4-12 コンフィグレーション名確認 - TMR0

図 4-13 のように TMR0 を設定してください。TMR0 は 1ms 周期に割り込みを発生するように設定しています。チュートリアルでは、この割り込みをアプリケーションのディレイタイマとして使用します。

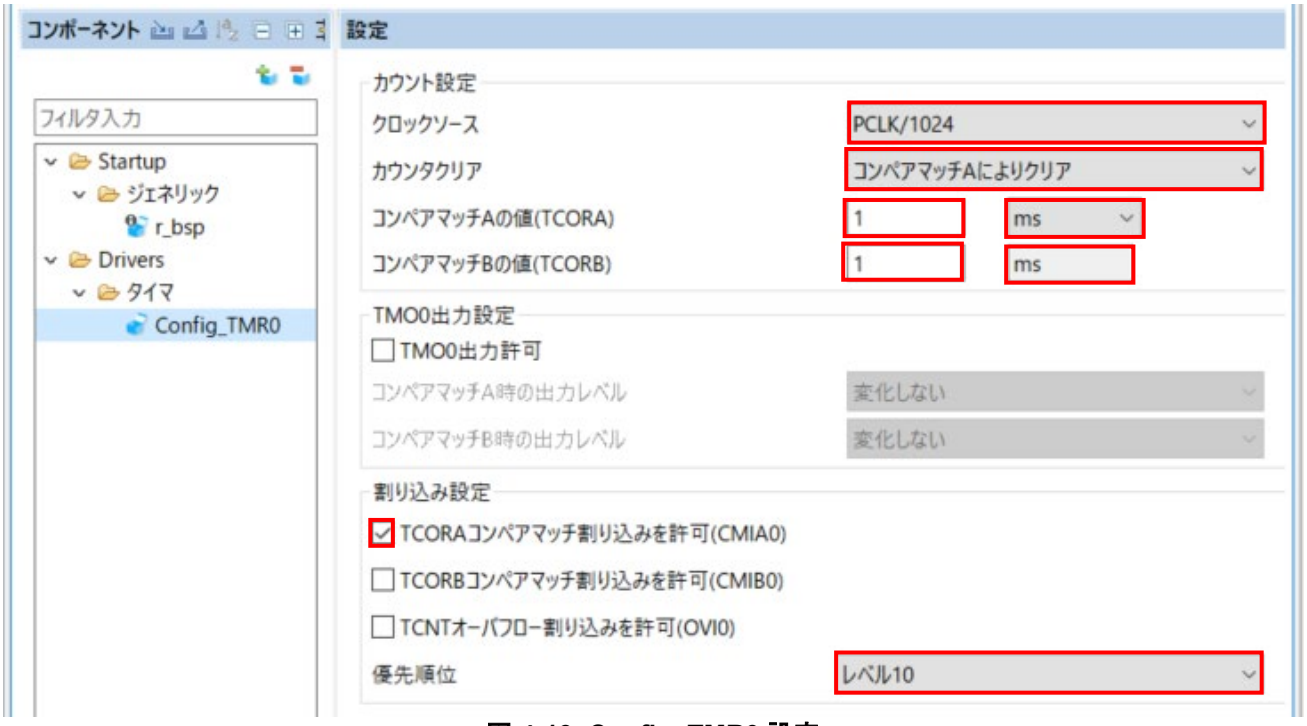


図 4-13 Config_TMR0 設定

4.6.3 コンペアマッチタイマ

CMT0 と 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 -> タイプは‘Drivers’を選択します。**図 4-14** のように '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 4-14 コンペアマッチタイマ選択

‘選択したコンポーネントの新しい構成を追加します。**図 4-15** のように“リソース”は、“CMT0”を選択します。

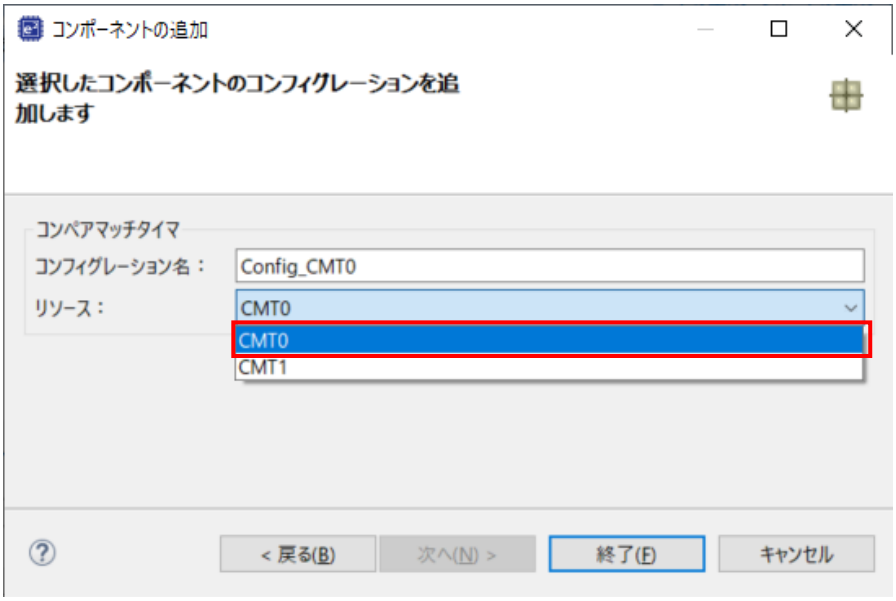


図 4-15 リソース選択 - CMT0

図 4-16 のように、コンフィグレーション名が“Config_CMT0”であることを確認し、’終了(F)’をクリックします。

図 4-16 コンフィグレーション名確認 - CMT0

図 4-17 のように CMT0 を設定してください。CMT0 は 20ms 周期に割り込みを発生するように設定しています。チュートリアルでは、この割り込みをスイッチのデバウンス用として使用します。

図 4-17 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 -> タイプは'Drivers'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 4-18 のように"リソース"は、"CMT1"を選択します。

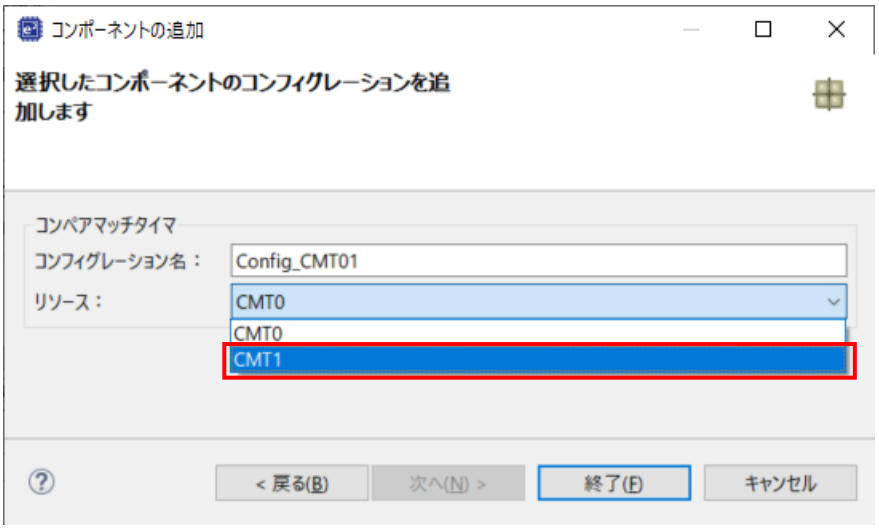


図 4-18 リソース選択 – CMT1

図 4-19 のように、コンフィグレーション名が“Config_CMT1”であることを確認し、’終了(F)’をクリックします。

図 4-19 コンフィグレーション名確認 - CMT1

図 4-20 のように CMT1 を設定してください。CMT1 は、200ms 周期に割り込みを生成するように設定しています。チュートリアルでは、この割り込みをスイッチのデバウンス用として使用します。

図 4-20 Config_CMT1 設定

4.6.4 割り込みコントローラ

RSKRX140 の CPU ボードは、SW1 に IRQ1（P31）、SW2 に IRQ2（P32）、SW3 に ADTRG0n（P16）が接続されています。ADTRG0n はセクション 4.6.8 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 -> タイプは‘Drivers’を選択します。**図 4-21** のように‘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。

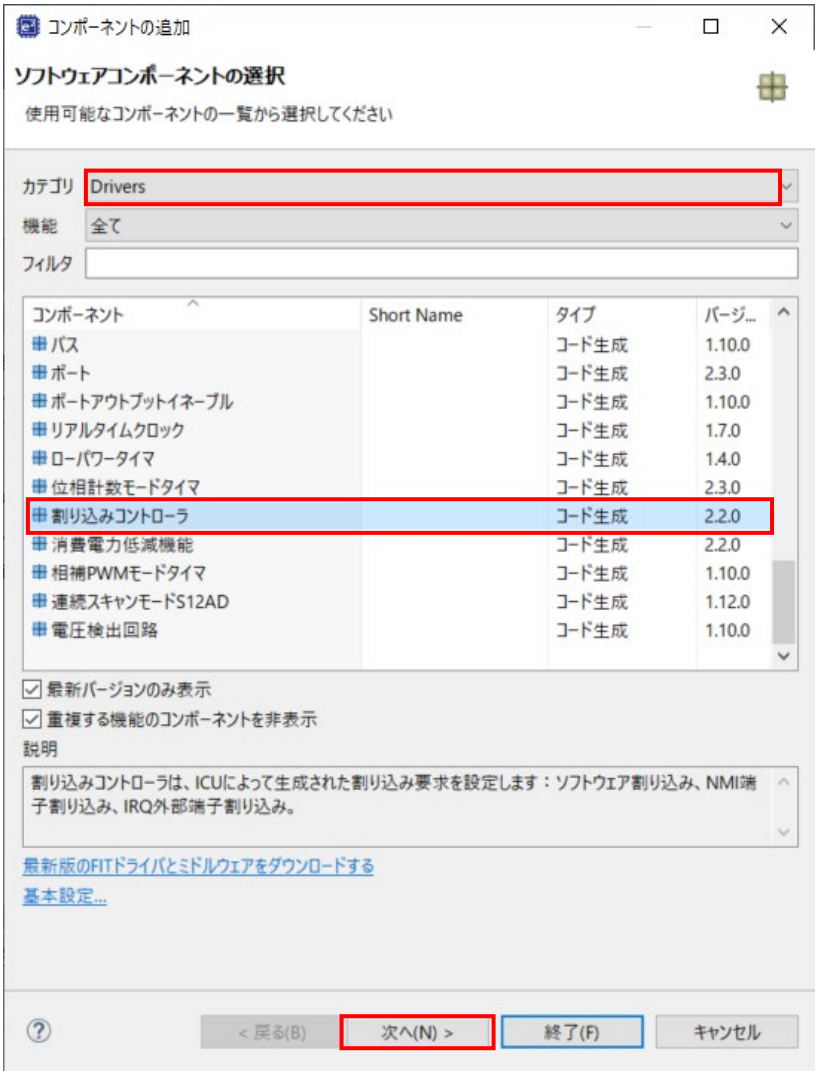


図 4-21 割り込みコントローラ選択

‘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。**図 4-22** のように“リソース”は、“ICU”を選択し、’終了(F)’をクリックします。

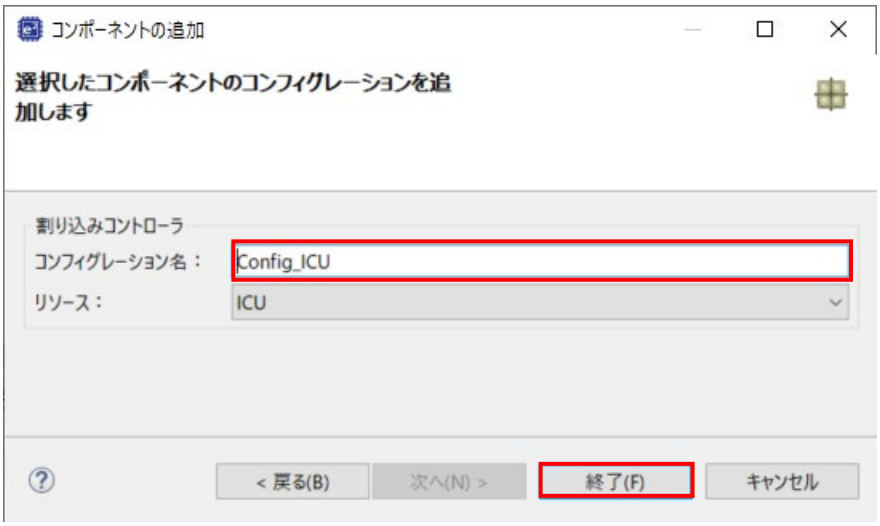


図 4-22 リソース選択 – ICU

’Config_ICU’で IRQ1、IRQ2 を**図 4-23** のように設定してください。

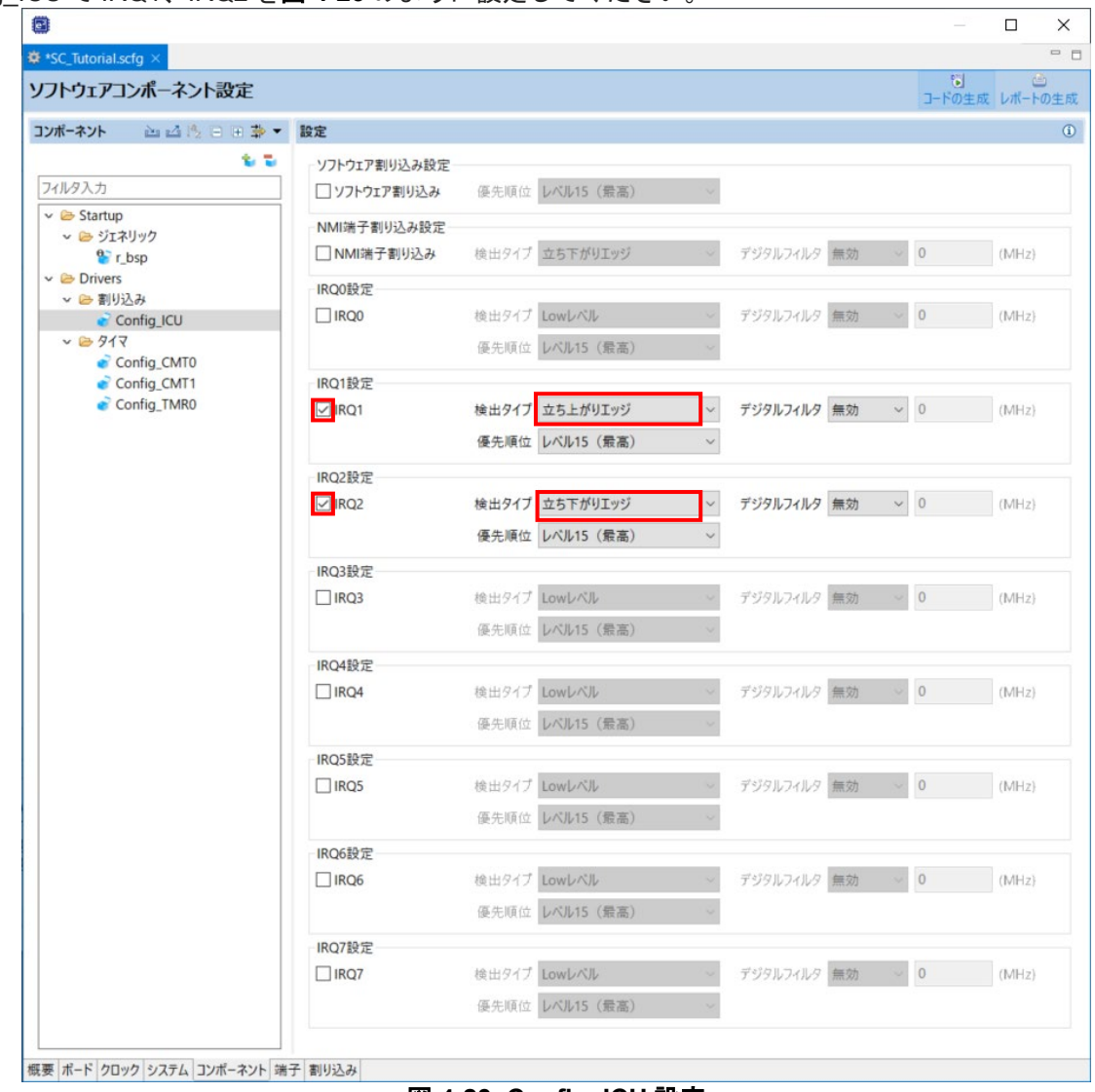


図 4-23 Config_ICU 設定

4.6.5 ポート

CPU ボードは、LED0 に P21、LED1 に P04、LED2 に P06、LED3 に P07 が接続されます。また、Pmod LCD に PB2、PE4、PC7、PC6 が接続されています。

‘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 -> タイプは‘Drivers’を選択します。**図 4-24** のように‘ポート’を選択し、‘次へ(N)’をクリックします。

図 4-24 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。**図 4-25** のように“リソース”は、“PORT”を選択し、“終了(F)”をクリックします。

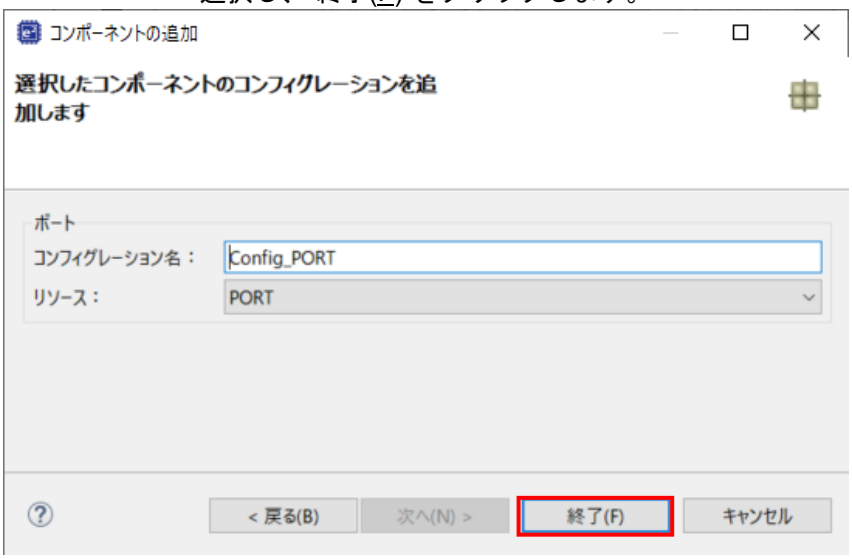


図 4-25 リソース選択 – PORT

図 4-26 のように PORT0、PORT2、PORTB、PORTC そして PORTE のチェックボックスをオンにしてください。

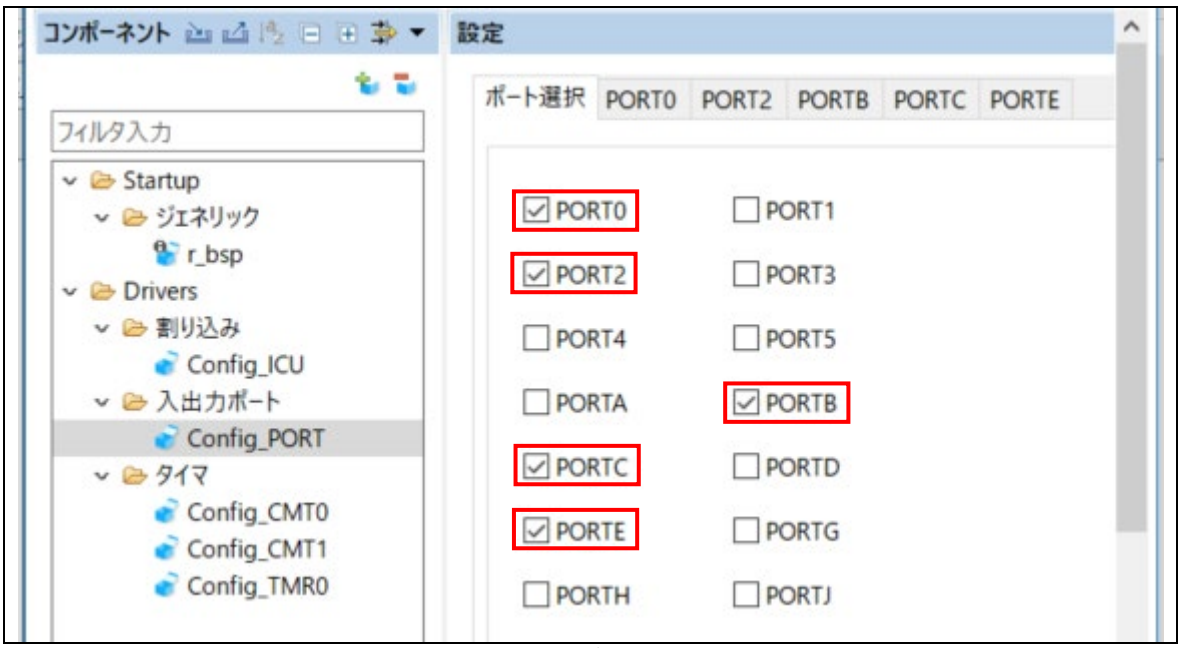


図 4-26 ポート選択

‘PORTx’タブをナビゲートし、**図 4-27**、**図 4-28**、**図 4-29**、**図 4-30**、そして**図 4-31** に示すように I/O ポートと LCD 制御ポートを構成します。‘出力’のチェックボックスおよび、PORTC タブの下の PC6 以外は ‘1 を出力’ チェックボックスをオンにしてください。まずは ‘PORT0’ タブより開始します。

図 4-27 PORT0 タブ選択

PORT2 タブを選択してください。

図 4-28 PORT2 タブ選択

PORTB タブを選択してください。

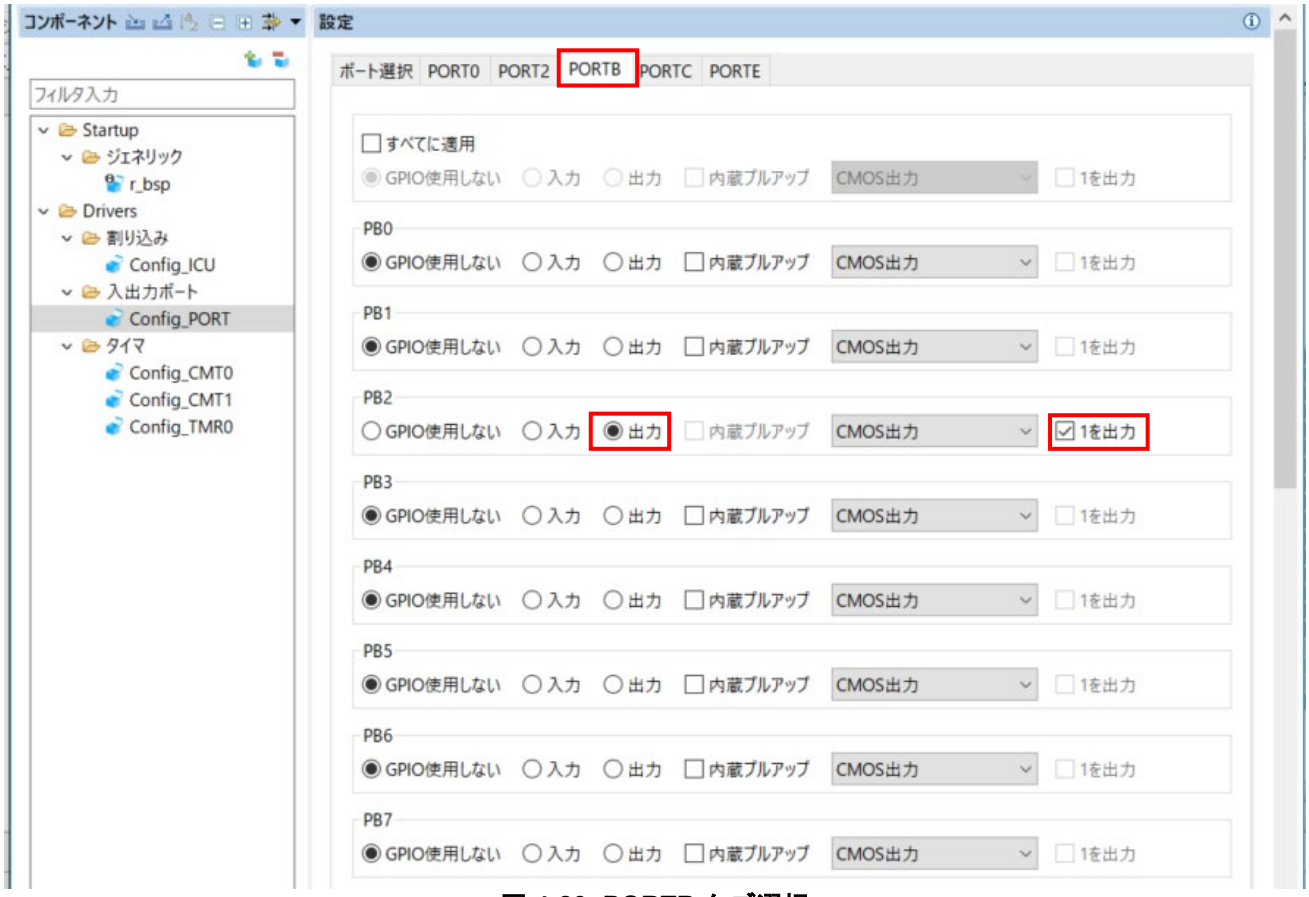


図 4-29 PORTB タブ選択

PORTC タブを選択してください。

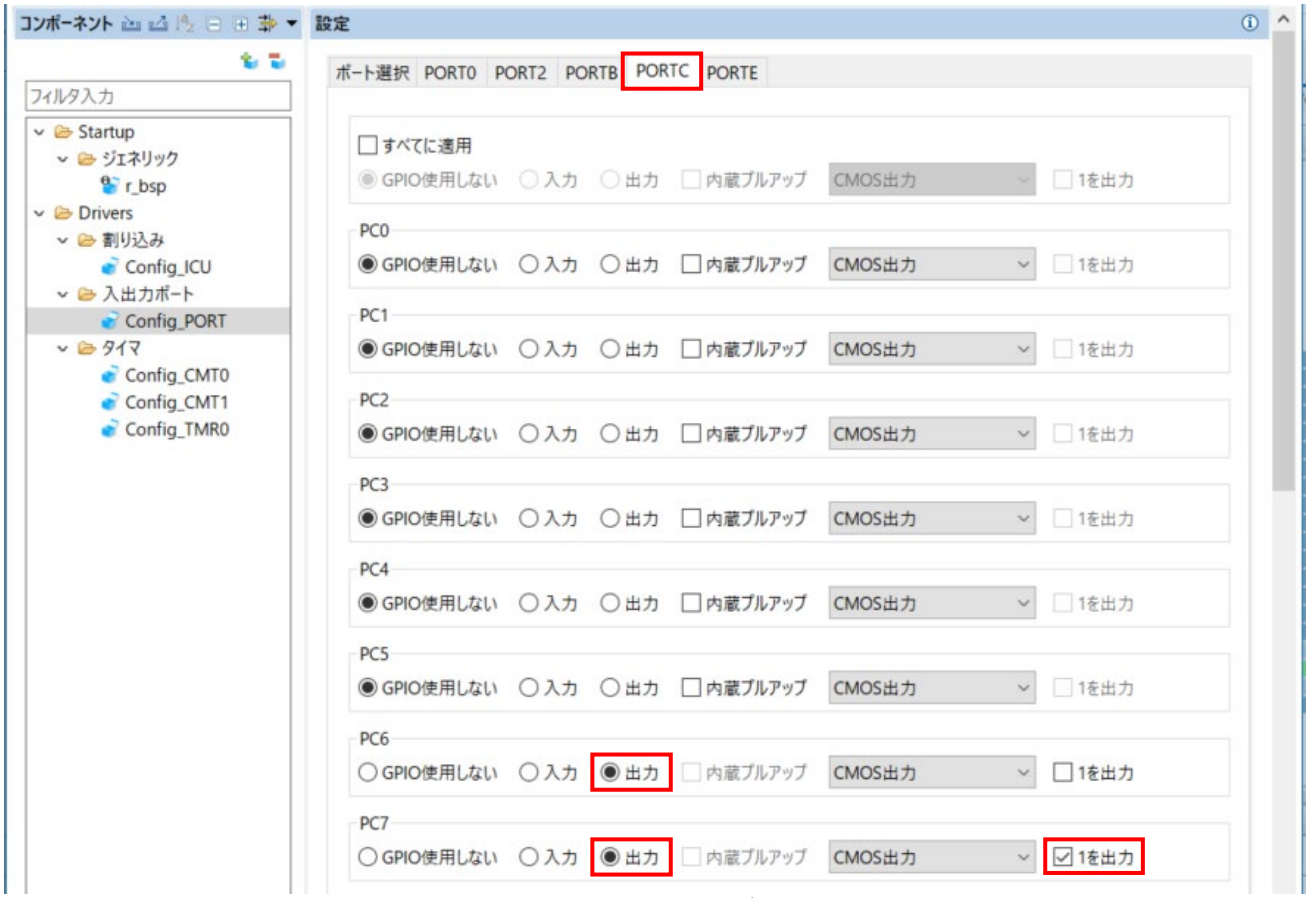


図 4-30 PORTC タブ選択

PORTE タブを選択してください。

図 4-31 PORTE タブ選択

4.6.6 SCI(SCIF)調歩同期式モード

RSKRX140 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 -> タイプは‘Drivers’を選択します。**図 4-32** のように‘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。

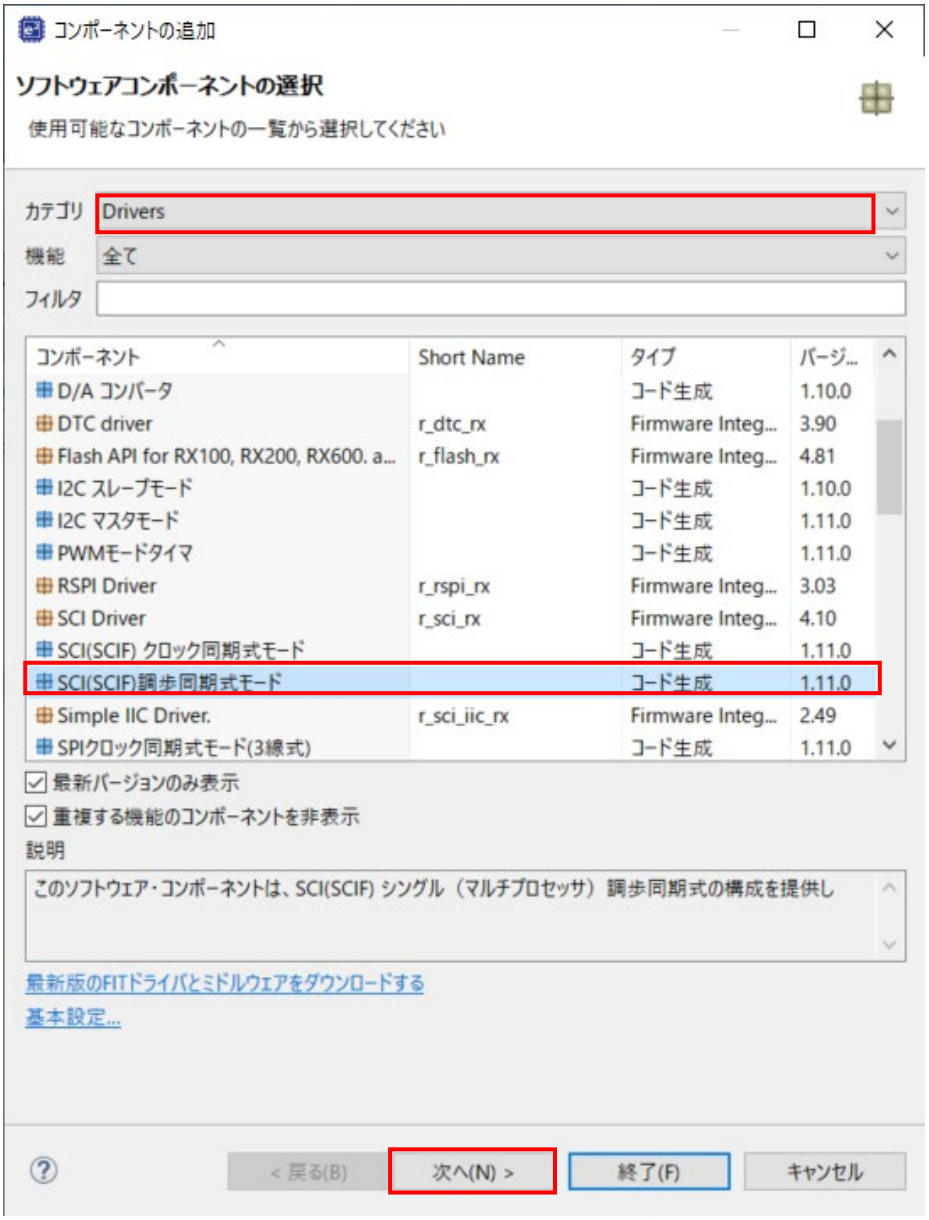


図 4-32 SCI(SCIF)調歩同期式モード選択

‘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 4-33** のように“送信/受信”を選択します。

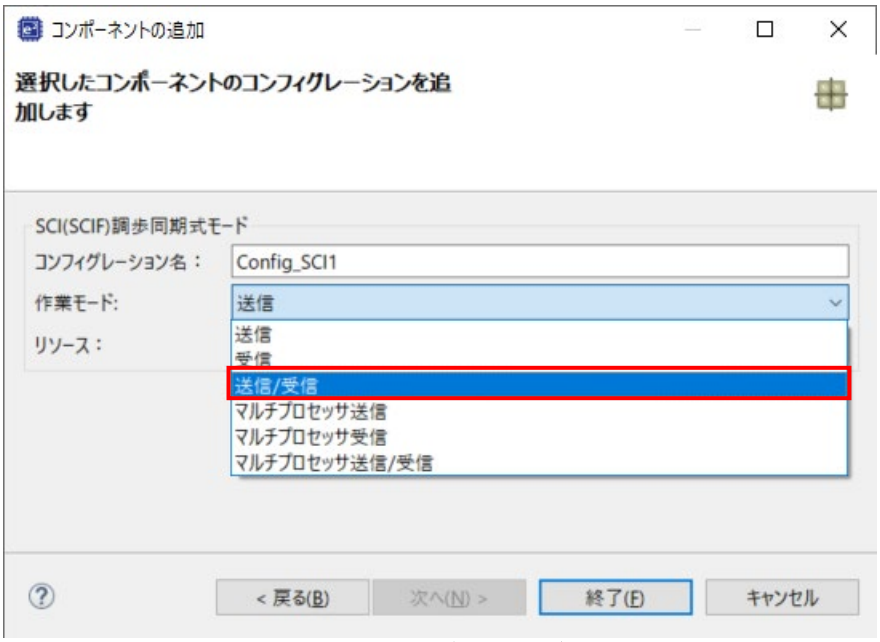


図 4-33 作業モード選択 – 送信/受信

図 4-34 のように“リソース”は、“SCI1”を選択してください。

図 4-34 リソース選択 – SCI1

図 4-35 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。

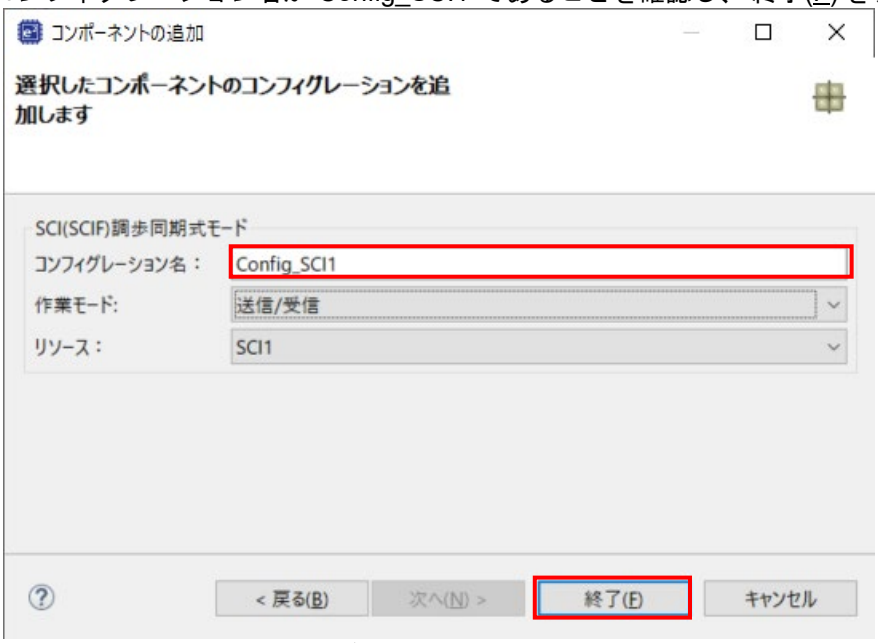


図 4-35 コンフィグレーション名確認 - Config_SCI1

図 4-36 のようにスタートビット検出設定を'RXD1 端子の立下りエッジ'、ビットレートを 19200 (bps)に設定してください。その 他の設定は初期設定のままです。

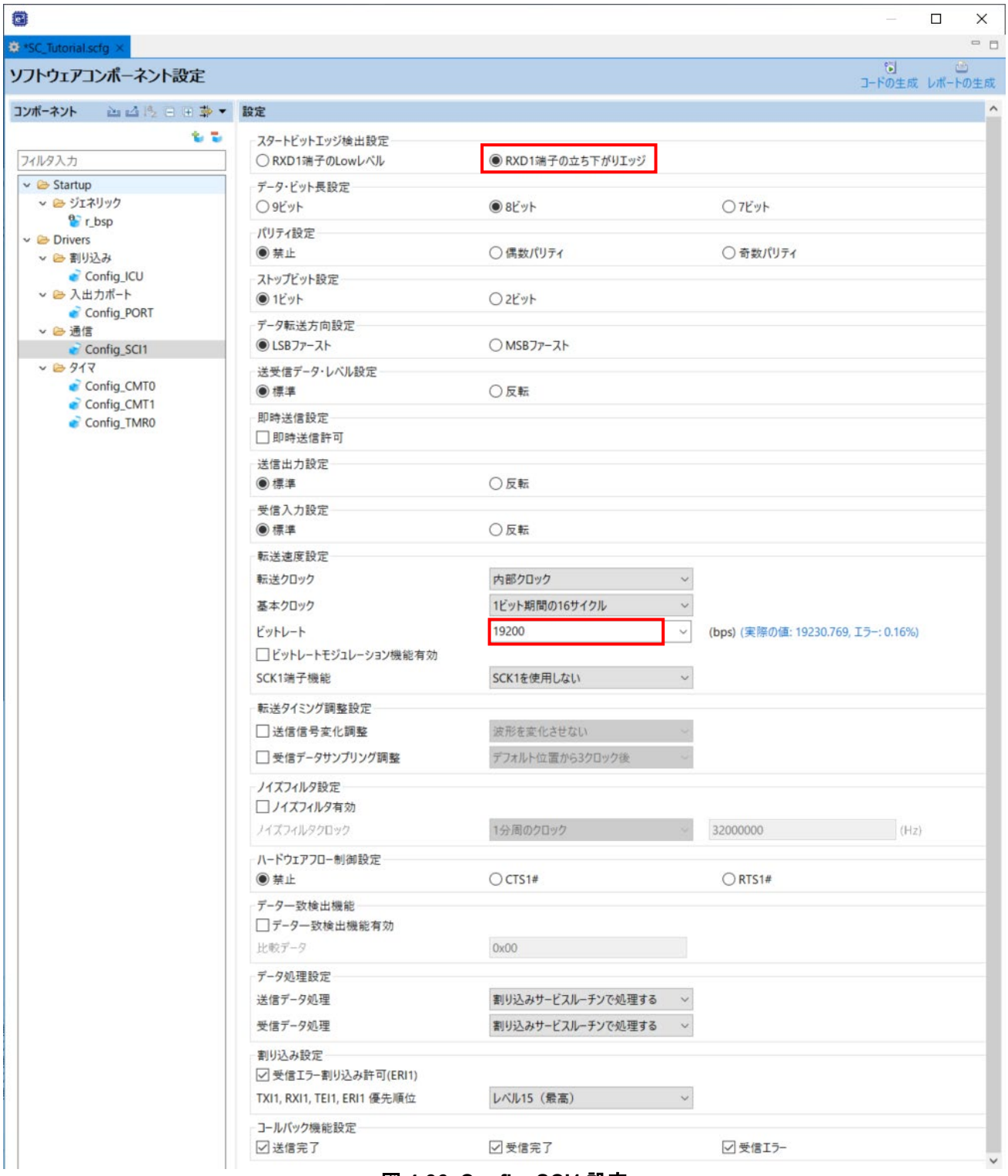


図 4-36 Config_SCI1 設定

4.6.7 SPI クロック同期式モード

RSKRX140 の CPU ボードは、SCI6 を Pmod LCD の SPI マスタとして使用します。

‘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 -> タイプは‘Drivers’を選択します。**図 4-37** のように‘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 4-37 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 4-38** のように’マスタ送信機能’を選択します。

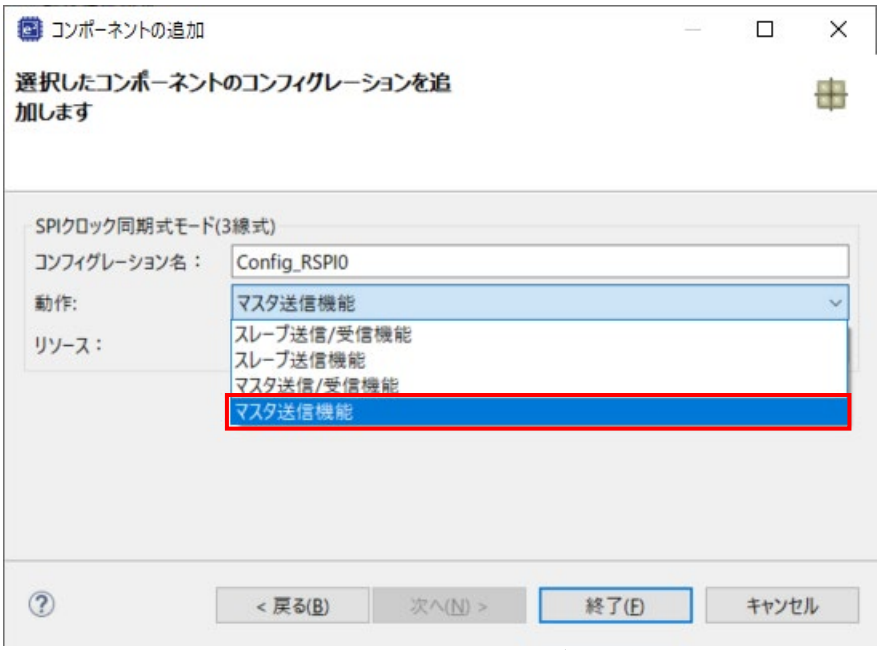


図 4-38 動作選択 – マスタ送信機能

図 4-39 のように“リソース”は、“SCI6”を選択してください。

図 4-39 リソース選択 – SCI6

図 4-40 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。

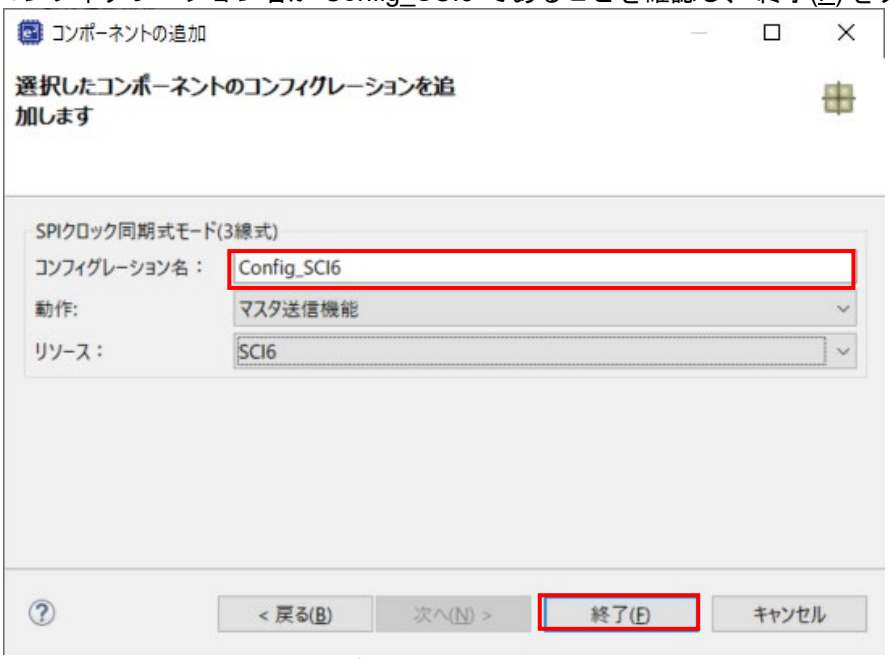


図 4-40 コンフィグレーション名確認 - Config_SCI6

図 4-41 のように SCI6 を設定します。データ転送方向設定を’MSB ファースト’に、ビットレートを 6000 (bps)に設定してください。その他の設定は初期設定のままです。

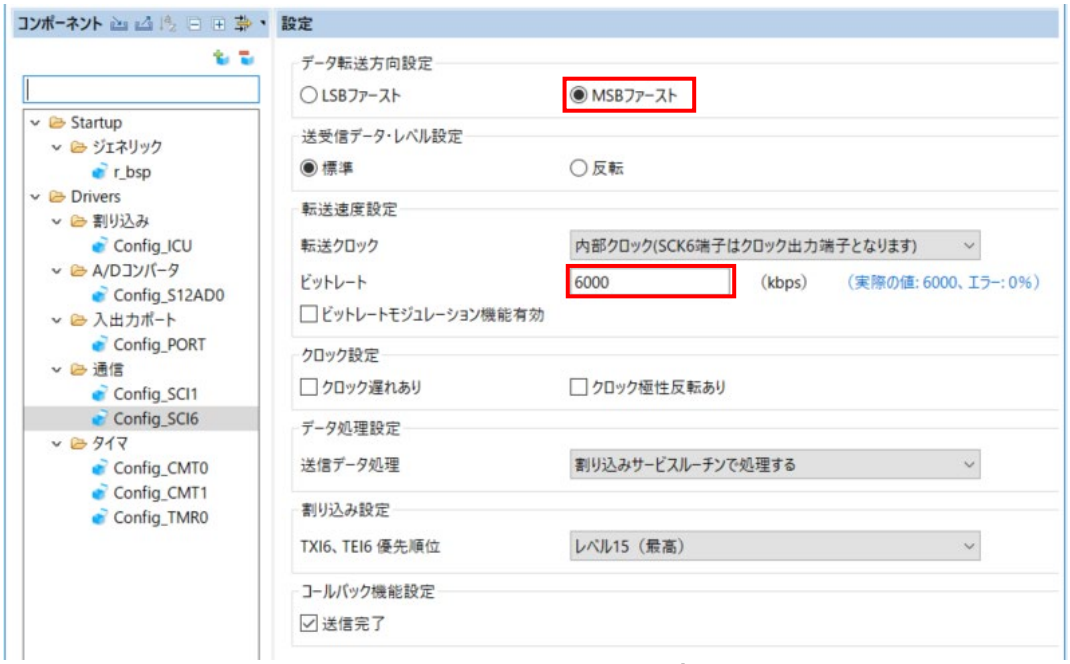


図 4-41 Config_SCI6 設定

4.6.8 シングルスキャンモード S12AD

RSKRX140 の 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 -> タイプは‘Drivers’を選択します。**図 4-42** 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 4-42 シングルスキャンモード S12AD 選択

図 4-43 のようにコンフィグレーション名が“Config_S12AD0”であることを確認し、’終了(F)’をクリックします。

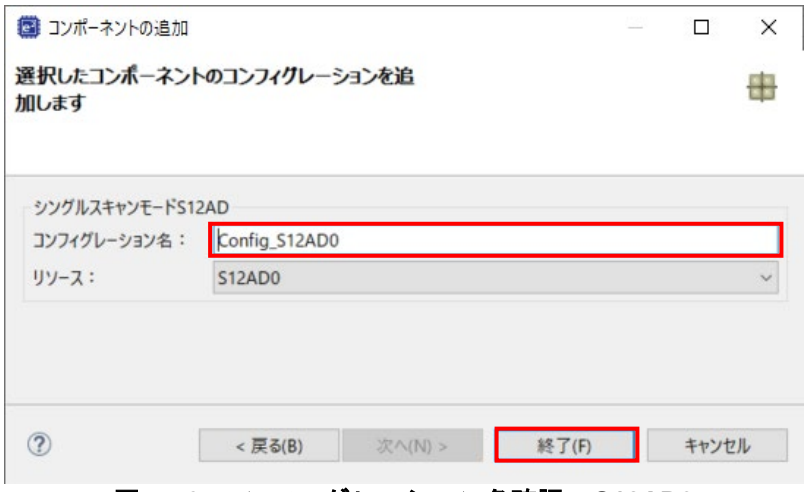


図 4-43 コンフィグレーション名確認 – S12AD0

図 4-44、図 4-45、図 4-46 のように S12AD0 を設定します。AN000 の アナログ入力チャンネル設定の AN000 チェックボックスをオン、 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。その他の設定は初期設定のままです。

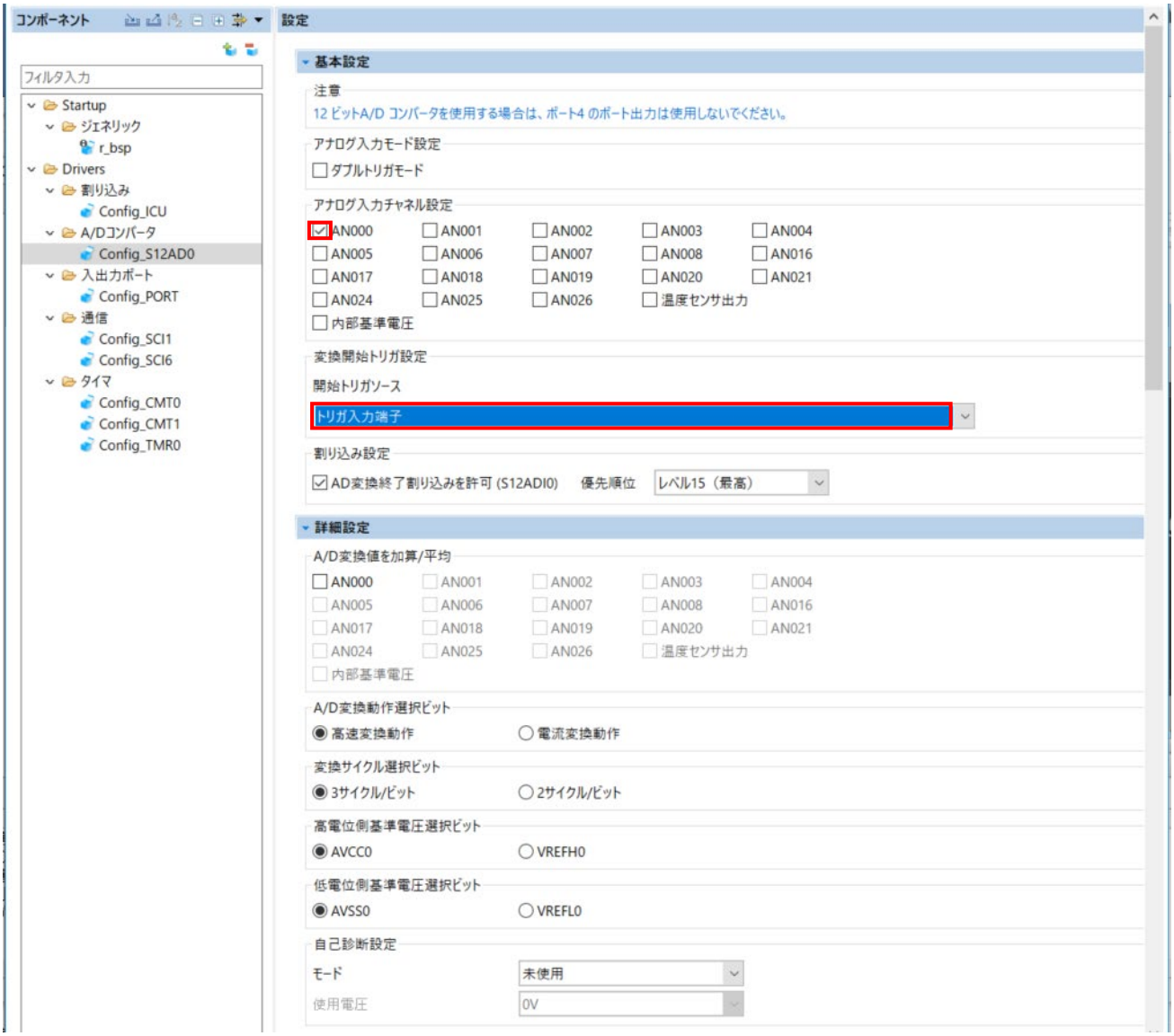


図 4-44 Config_S12AD0 設定(1)

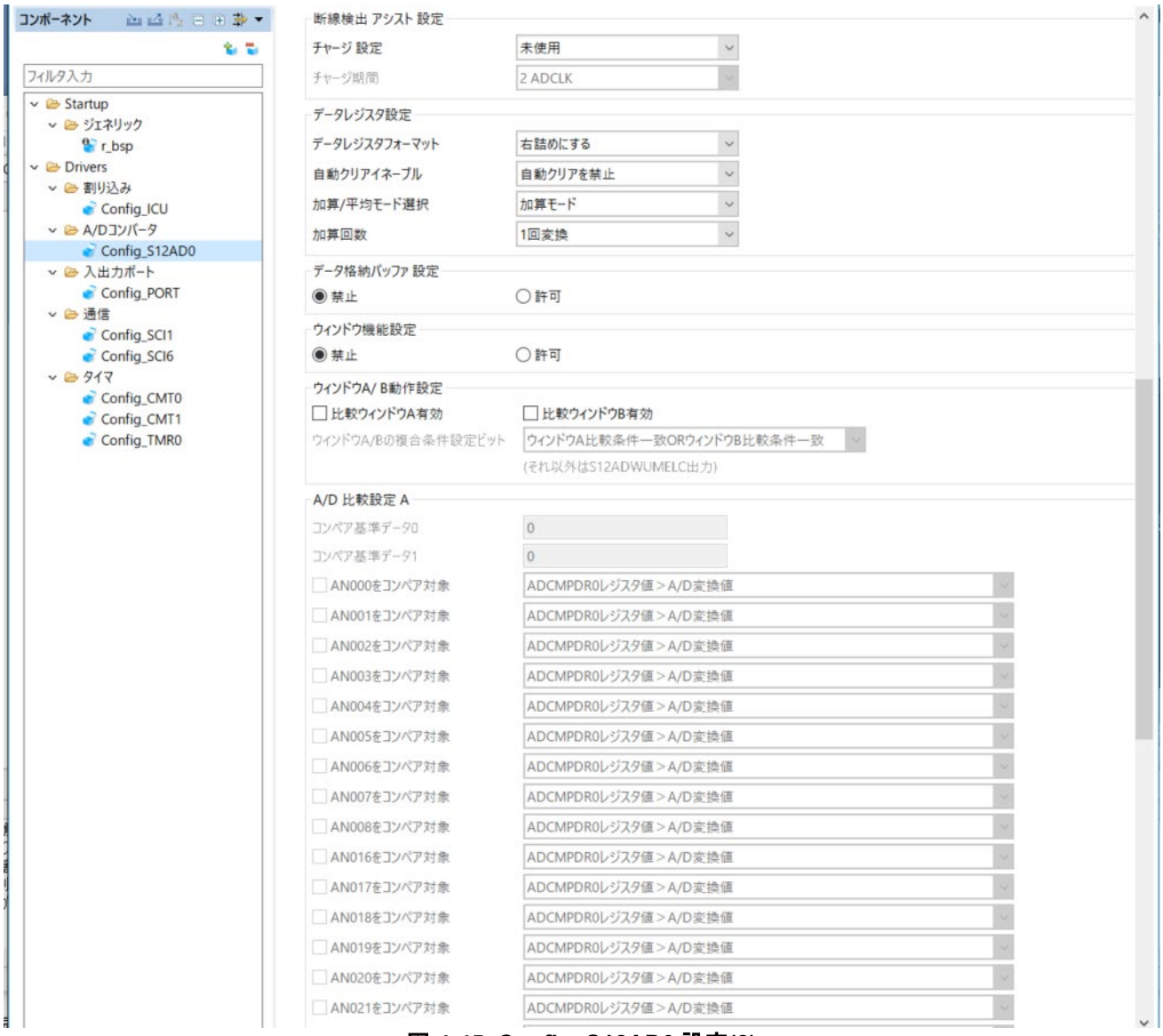


図 4-45 Config_S12AD0 設定(2)

図 4-46 Config_S12AD0 設定(3)

4.7 端子ページ

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、端子ページで変更できます。

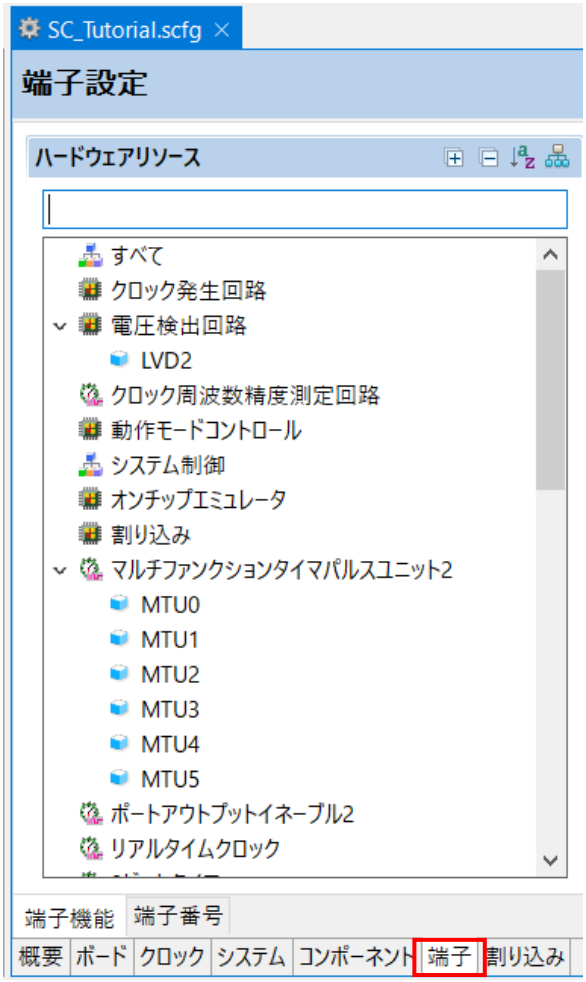


図 4-47 端子ページ

4.7.1 ソフトウェアコンポーネントのピン設定変更

ピン機能一覧で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。  をクリックして、ソフトウェアコンポーネントで表示するビューから変更します。

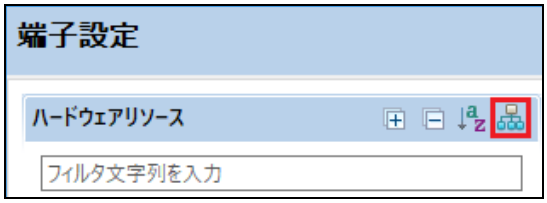


図 4-48 ハードウェアリソース表示への切り替え

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 に設定されていることを確認してください。**図 4-49** のように、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。

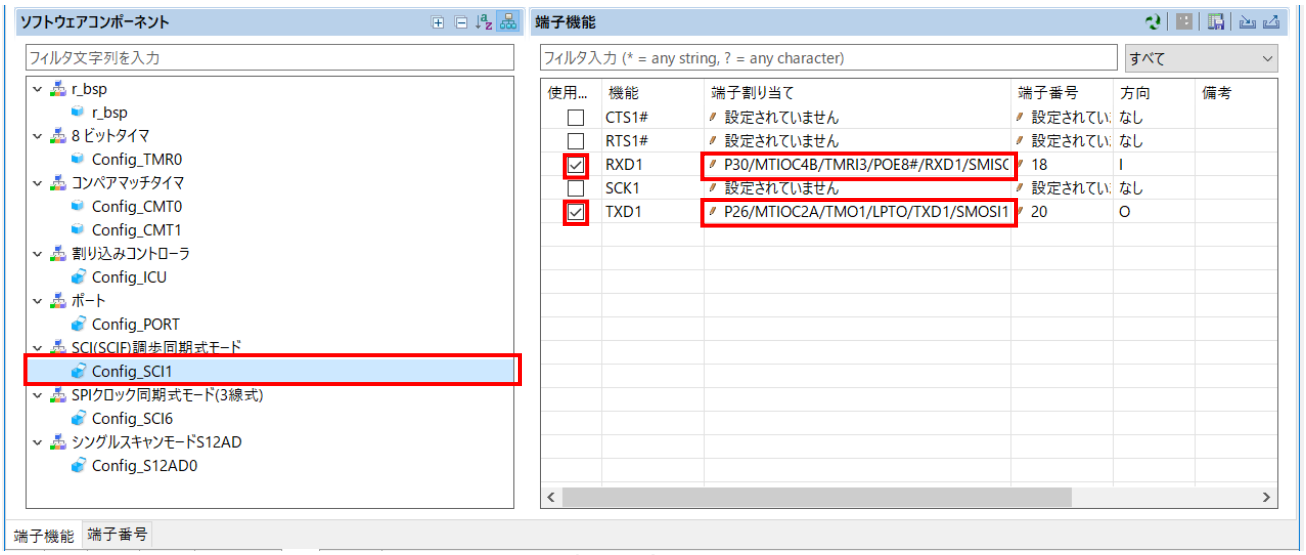


図 4-49 端子設定 - Config_SCI1

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 に設定されていることを確認してください。**図 4-50** のように、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。

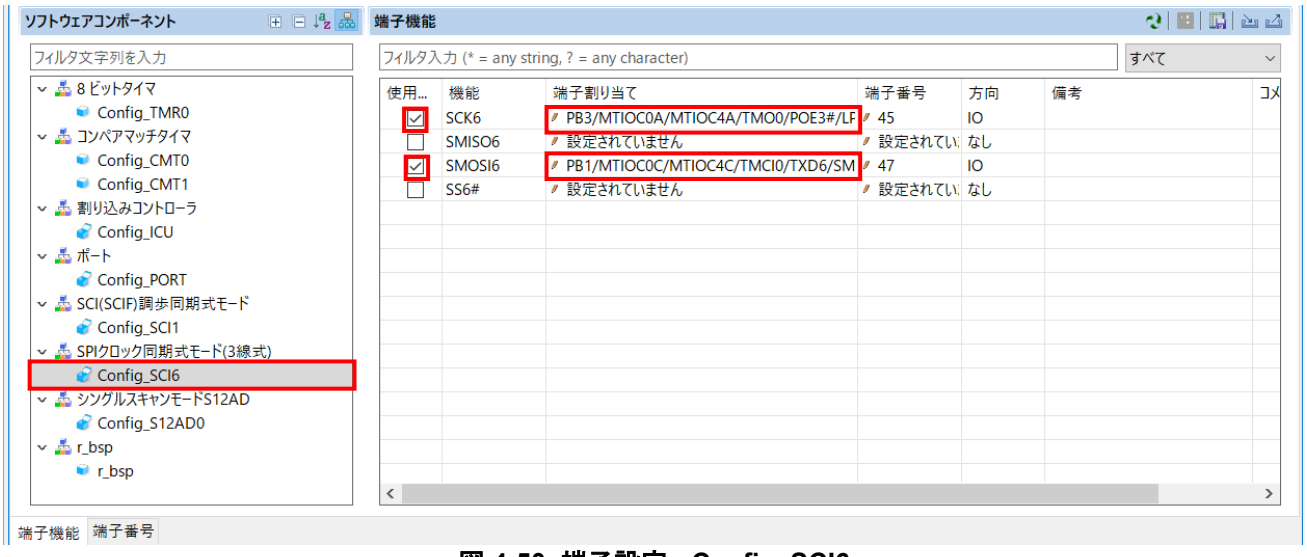


図 4-50 端子設定 - Config_SCI6

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 に設定されていることを確認してください。**図 4-51** のように、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。

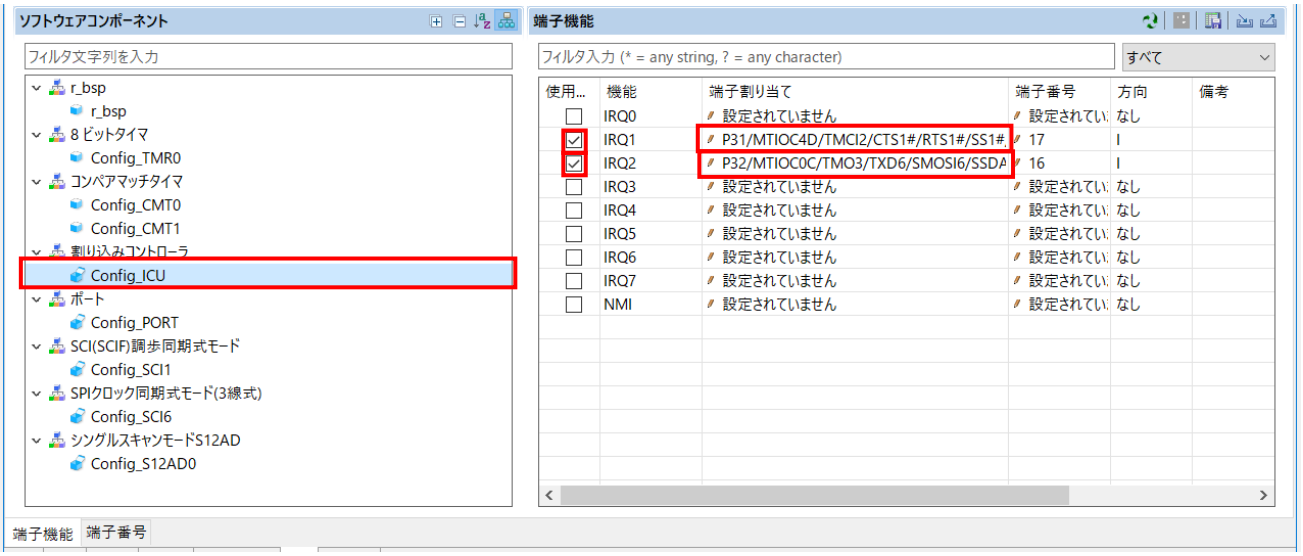


図 4-51 端子設定 - Config_ICU

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 に設定されていることを確認してください。**図 4-52** のように、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。

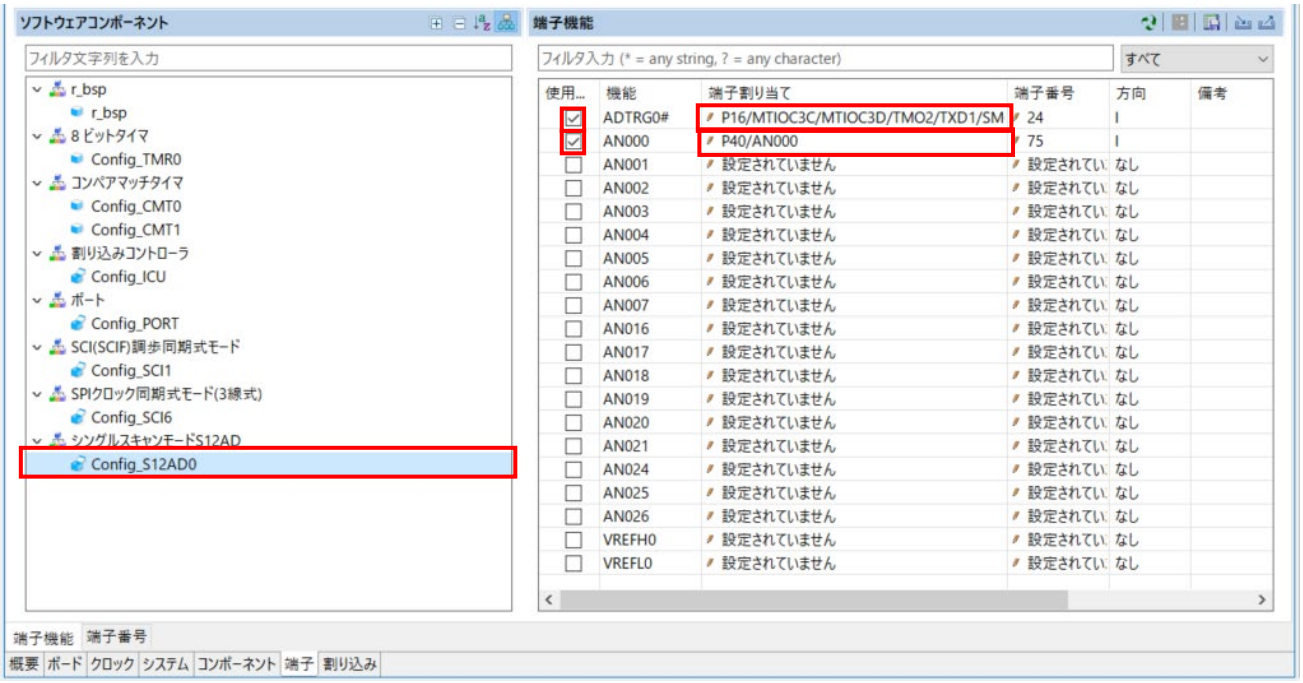


図 4-52 端子設定 - Config_S12AD0

これで周辺機能の設定は全て完了しました。ファイル(F) - >保存(S)でプロジェクトを保存してください。
次に**図 4-53** 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 4-53 コード生成ボタン

図 4-54 のように、コンソールに’コード生成の終了’と表示されます。

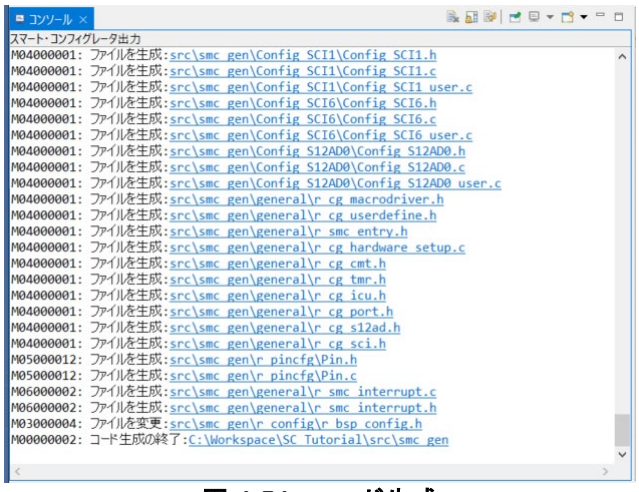


図 4-54 コード生成

4.8 プロジェクトのビルド

スマート・コンフィグレータで作成したプロジェクトテンプレートを作成できるようになりました。プロジェクト・エクスプローラーで、 'src'フォルダ、smc_gen フォルダを展開します。

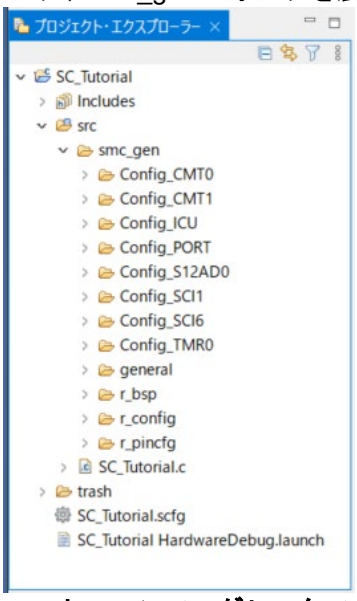


図 4-55 スマート・コンフィグレータフォルダ構成

ワークスペースの右上にある ボタンをクリックして、 'C / C ++'パースペクティブに戻ります。プロジェクト・エクスプローラーで SC_Tutorial プロジェクトを選択し、'プロジェクト'メニューの「ビルドプロジェクト」またはチュートリアルを作成する ボタンをクリックします。e^2 studio はエラーのないプロジェクトを構築します。

5. ユーザコードの統合

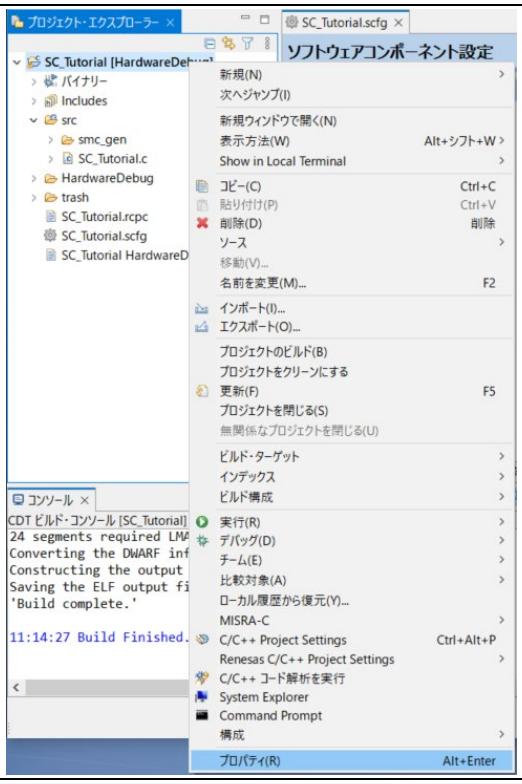
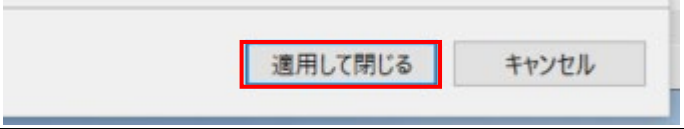
この章では、残りのアプリケーションコードをプロジェクトに追加します。ユーザは RSK Web インストーラにあるソースファイルをワークスペースへのコピー、コード生成ファイルのユーザコードエリアにコードを追加するよう指示します。

このプロジェクトのスマート・コンフィグレータが生成する複数のファイル内のユーザコードエリアにコードを挿入する必要があります。これらのユーザコードエリアは、次のようにコメントで区切られています。

```
/* 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。

5.1 プロジェクト設定

• プロジェクトを構築する前に、ビルドコンフィグレーション’HardwareDebug’の最適化レベルを変更します。SC_Tutorial プロジェクトを選択した状態で、右クリックして[プロパティ]を選択、またはショートカットキー[Alt] + [Enter]を使用して[プロパティ]を開きます。	
• C/C++ビルド -> 設定 -> Compiler -> 最適化に移動します。 • 最適化レベルのプルダウンから’レベル 0：最適化をしない’を選択します。	
• ’適用して閉じる’ボタンを押して[プロパティ]を閉じます。	

5.2 LCD パネルコードの統合

Pmod LCD の API 機能は、RSK とともに提供されています。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。Tutorial プロジェクトの src フォルダに以下のファイルがあることを確認します：

・ascii.c
・ascii.h
・r_okaya_lcd.c
・r_okaya_lcd.h

これらのファイルをワークスペースの下の src フォルダにコピーしてください。ファイルは**図 5-1** のように自動的にプロジェクトへ追加されます。

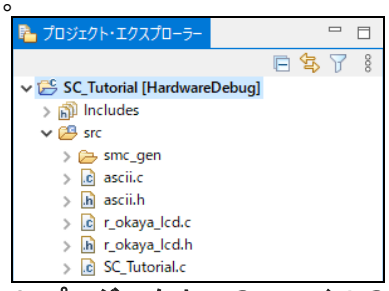


図 5-1 プロジェクトへのファイルの追加

e² studio のプロジェクト・ツリーで、 'src\smc_gen\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 を以下のように追加してください。

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

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

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

同じくファイル先頭付近 include のユーザコードエリアに以下の#include を追加してください。

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

#include "platform.h"

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

同じく typedefine ユーザコードエリアには以下のように追加してください。

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

typedef char char_t;

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

e² studio のプロジェクト・ツリーで 'src'フォルダを展開して、'SC_Tutorial.c'ファイルをダブルクリックして開いてください。次に 宣言 '#include r_smc_entry.h'の下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、以下のようにハイライトで明示した箇所を main 関数に追加してください。

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

    /* Displays the application name on the debug LCD */
    R_LCD_Display(0, (uint8_t *)" RSKRX140 ");
    R_LCD_Display(1, (uint8_t *)" Tutorial ");
    R_LCD_Display(2, (uint8_t *)" Press Any Switch ");
    while (1U)
    {
        ;
    }
}
```

このマニュアルに記載されているコードを e^2 studio のソースファイルに貼り付ける場合、インデントが失われるためご注意ください。

5.2.1 SPI コード

Pmod LCDはセクション4.6.7で設定したSPIマスタ送信によって制御されます。e² studioのプロジェクト・ツリーの'src\smc_gen\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。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コードを追加してください。

```
/* 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の間には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 s_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コードエリアには以下のように追加してください。

```
static void r_Config_SCI6_callback_transmitend(void)
{
    /* Start user code for r_Config_SCI6_callback_transmitend. Do not edit comment generated here */

    s_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には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 */
    s_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 == s_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用し、LCD モジュールの API として使用します。

5.2.2 TMR コード

Pmod LCD 制御において、タイミング要件を満たすためにディレイタイマを挿入する必要があります。セクション 4.6.2 で設定した TMR を使用して実現します。'src\smc_gen\Config_TMR0\Config_TMR0.h'を開いて、ファイルの最後尾にある'function'ユーザコードエリアには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'を開いて'global'ユーザコードエリアには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 s_one_ms_delay_complete = FALSE;

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

r_Config_TMR0_cmia0_interrupt 関数までスクロールし、以下の行をユーザコードエリアには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 */

    s_one_ms_delay_complete = TRUE;

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

次にファイル最後尾のユーザコードエリアには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 == s_one_ms_delay_complete)
        {
            /* Wait */
        }
        R_Config_TMR0_Stop();
        s_one_ms_delay_complete = FALSE;
        ms_count++;
    } while (ms_count < millisec);
}
/***********************************************************************
End of function R_TMR_MsDelay
***********************************************************************/
```

5.3 インクルードパスの追加

プロジェクトを構築する前に、コンパイラにインクルードパスを追加する必要があります。プロジェクト・エクスプローラーで SC_Tutorial プロジェクトを選択、を右クリックで'プロパティ'を選択します。
図 5-2 のように C / C ++ビルド -> 設定 -> Compiler -> ソースに移動し、'インクルード・ファイルを検索するフォルダ'の横に追加ボタン をクリックます。

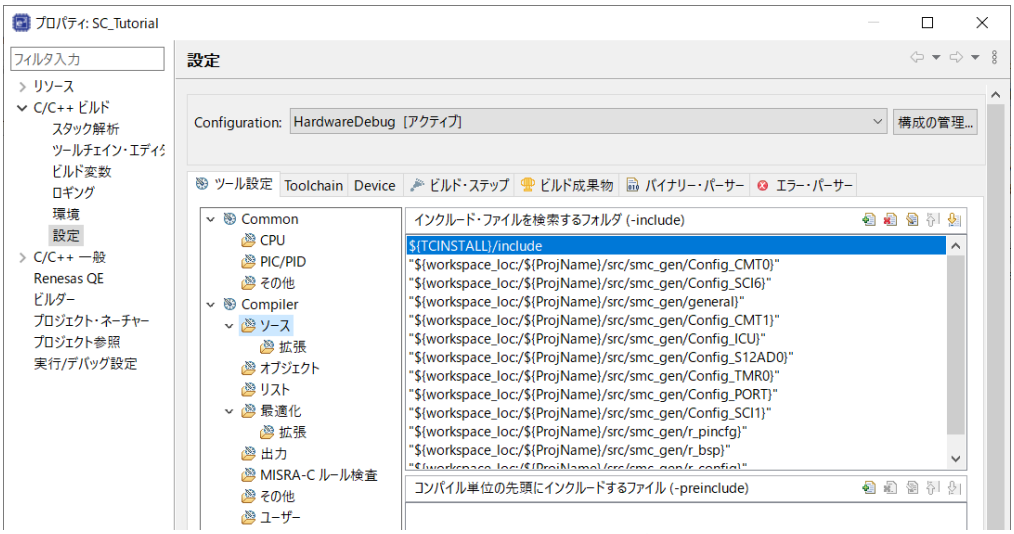


図 5-2 インクルードパス追加

'ディレクトリ・パスの追加'ダイアログで'ワークスペース...'ボタンをクリックし、'フォルダ選択'ダイアログで SC_Tutorial/src フォルダを参照します。**図 5-3** のように ディレクトリ・パスを追加できたら'OK'ボタンをクリックします。

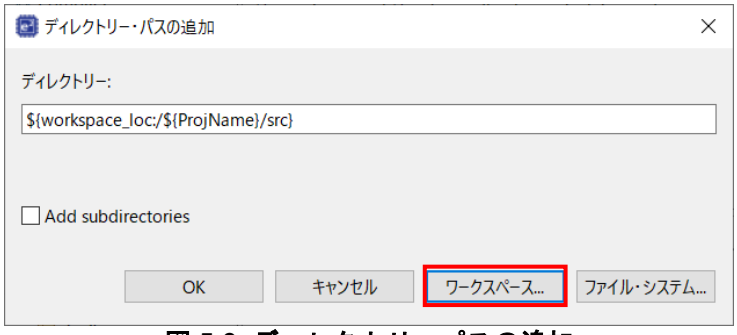


図 5-3 ディレクトリ・パスの追加

'適用して閉じる'ボタンをクリックしてプロパティを閉じ、**図 5-4** の設定ダイアログが表示されたら、'はい(Y)'をクリックして設定を終了させてください。

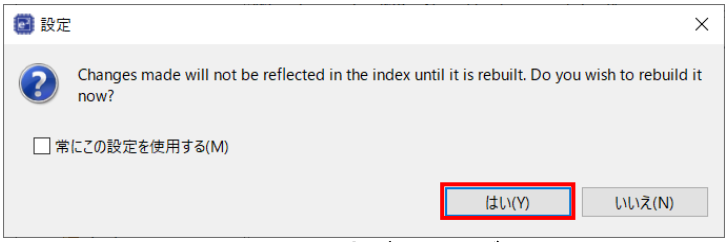


図 5-4 設定ダイアログ

'プロジェクト'メニューから 'すべてビルド'を選択するか、または ボタンをクリックしてください。
e² studio はエラーのないプロジェクトを構築します。
プロジェクトは 6 章で説明するデバッガを使用して実行できるようになりました。Pmod LCD の 1 行目に 'RSKRX140'、2 行目に'Tutorial'、3 行目に'Press Any Switch'を表示します。

5.4 スイッチコードの統合

スイッチの API 機能は、RSK とともに提供されています。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。Tutorial プロジェクトの src フォルダに以下のファイルがあることを確認します：

・rskrx140def.h’
・r_rsk_switch.c
・r_rsk_switch.h

これらのファイルをワークスペースの下の src フォルダにコピーしてください。

スイッチコードでは、セクション 4.6.4 で設定したファイル Config_ICU.h、Config_ICU.c および Config_ICU_user.c の中の割り込み機能と、セクション 4.6.3 で設定したファイル Config_CMT0.h、Config_CMT0.c、Config_CMT0_user.c、Config_CMT1.h、Config_CMT1.c および Config_CMT1_user.c のコンペアマッチタイマ機能を使用します。これらのファイルには、r_rsk_switch.c の API 関数で必要となるスイッチの ON/OFF の検出とデバウンス処理を実装するため、追加のユーザコードを提供する必要があります。

5.4.1 割り込みコード

e^2 studio のプロジェクト・ツリーで、'src\smc_gen\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。ファイル最後尾のユーザコードエリアには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ードエリアには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には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 */
```

同じファイルの r_Config_ICU_irq1_interrupt 関数内のユーザコードエリアには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 */
```

同じファイルの r_Config_ICU_irq2_interrupt 関数内のユーザコードエリアには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 */
```

5.4.2 デバウンス用タイマコード

e^2 studio のプロジェクト・ツリーで、'src\smc_gen\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には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には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 */
```

'src\smc_gen\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には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 */
```

同じファイルの r_Config_CMT1_cmi1_interrupt 関数内のユーザコードエリアには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 */
```

5.4.3 A/D コンバータコードとメインスイッチ

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、その結果を LCD に表示します。セクション 4.6.8 で設定した A/D 変換開始トリガ(ADTRG0＃入力端子)を使用するために、CPU ボード上では SW3 に接続されています。このコードでは SW1 または SW2 の入力が検出されると、直ちに ADC トリガソースを再設定することにより、ユーザスイッチ SW1 および SW2 からソフトウェアトリガ A/D 変換も実行します。
e^2 studio のプロジェクト・ツリーで、 'src\smc_gen\general'フォルダを展開して、 'r_cg_userdefine.h'ファイルを開いてください。ユーザコードエリアには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 */
extern volatile uint8_t g_adc_trigger;

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

プロジェクト・ツリーで 'src' フォルダを展開して 'SC_Tutorial.c' ファイルを開き、以下のハイライト表示されたコードを追加してください。

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

次に main 関数内に以下のハイライト表示されたコードと while 文の中にコード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 *)" RSKRX140 ");
    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、cb_switch_press 関数、get_adc 関数、lcd_display_adc 関数をファイル最後尾に以下を追加してください。

```
/***********************************************************************
* 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 */
        nop();
    }

    /* 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 */
    char_t tmp;

    /* Declare temporary character string */
    char_t 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. */
    tmp           = (char_t)((adc_result & 0x0F00) >> 8);
    lcd_buffer[6] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp           = (char_t)((adc_result & 0x00F0) >> 4);
    lcd_buffer[7] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp           = (char_t)(adc_result & 0x000F);
    lcd_buffer[8] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);

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

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

'src\smc_gen\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には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'を開いてファイルの最後尾のユーザコードエリアには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_S12AD0_SWTriggerStart
* Description  : This function starts the AD0 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 AD0 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には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には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 */
}
```

'プロジェクト'メニューから 'すべてビルド'を選択するか、または  ボタンをクリックしてください。
e^2 studio はエラーのないプロジェクトを構築します。

プロジェクトは 6 章で説明するデバッガを使用して実行できるようになりました。いずれかのスイッチを押すと、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 に表示されます。
UART ユーザコードを追加する場合は、ここからチュートリアルを読み進めてください。

5.5 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 とともに提供されています。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。Tutorial プロジェクトの src フォルダに以下のファイルがあることを確認します：

・r_rsk_debug.c
・r_rsk_debug.h

これらのファイルをワークスペースの下の src フォルダにコピーしてください。

r_rsk_debug.h ファイルで、次のマクロ定義が含まれていることを確認します。

```
/* 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'ファイルで参照され、異なるデバッグインタフェースが使用されている場合、デバッグ出力の再設定を容易にします。

5.6 UART コードの統合

5.6.1 SCI コード

e² studio プロジェクト・ツリーで'src\smc_gen\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。ファイル最後尾の'function'ユーザコードエリアには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には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 uint8_t s_sci1_txdone;

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

r_Config_SCI1_callback_transmitend 関数のユーザコードエリアには以下のように追加してください。

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

    s_sci1_txdone = TRUE;

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

r_Config_SCI1_callback_receiveend 関数内のユーザコードエリアには以下のように追加してください。

```
static void r_Config_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 */
}
```

ファイルの最後尾の'function'ユーザコードエリアには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 */
    s_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 == s_sci1_txdone)
    {
        /* Wait */
    }
    return (status);
}

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

5.6.2 メイン UART コード

ファイル'SC_Tutorial.c'を開いてください。以下のハイライト表示されたコードを追加してください。

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_cg_userdefine.h"
#include "Config_S12AD0.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "Config_SCI1.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 s_adc_count = 0;
```

main 関数に以下のハイライト表示されたコード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 *)" RSKRX140 ");
    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 s_adc_count */
            if (16 == (++s_adc_count))
            {
                s_adc_count = 0;
            }

            /* Send the result to the UART */
            uart_display_adc(s_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_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

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

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

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

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

次に、ファイルの最後尾に以下の関数を追加してください。

```
/***********************************************************************
* Function Name : uart_display_adc
* Description   : Converts adc result to a string and sends it to the UART.
* 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_t tmp;

    /* Declare temporary character string */
    char_t 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. */
    tmp             = (char_t)(adc_count & 0x000F);
    uart_buffer[4] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp             = (char_t)((adc_result & 0x0F00) >> 8);
    uart_buffer[14] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp             = (char_t)((adc_result & 0x00F0) >> 4);
    uart_buffer[15] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);
    tmp             = (char_t)(adc_result & 0x000F);
    uart_buffer[16] = (tmp < 0x0A) ? (tmp + 0x30) : (tmp + 0x37);

    /* Send the string to the UART */
    r_debug_print(uart_buffer);
}

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

'プロジェクト'メニューから 'すべてビルド'を選択するか、または  ボタンをクリックしてください。
e² studio はエラーのないプロジェクトを構築します。

プロジェクトは 6 章で説明するデバッガを使用して実行できるようになりました。
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 G1CUSB0 ポートを USB ケーブルで接続する必要があります。CPU ボードが初めて PC に接続する場合は、デバイスドライバが自動的にインストールされます。デバイスマネージャー上のポート(COM＆LPT）に 'RSK USB Serial Port (COMx)'として表示されます。(x は数字)

ハイパーターミナルなどのターミナルプログラムを、SCI1 と同じ設定にしてください（ボーレート：19200, データ長: 8, パリティビット: なし, ストップビット: 1, フロー制御: なし）。いずれかのスイッチを押すか、ターミナルソフト画面上でキーボードの'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 SCI1 を介してターミナル画面上に表示します。
LED ユーザコードを追加する場合は、再びここからチュートリアルを読み進めてください。

5.7 LED コードの統合

ファイル'SC_Tutorial.c'を開いてください。以下のハイライト表示されたコードを追加してください。

```
#include "r_smc_entry.h"
#include "r_okaya_lcd.h"
#include "r_cg_userdefine.h"
#include "Config_S12AD0.h"
#include "r_rsk_switch.h"
#include "r_rsk_debug.h"
#include "Config_SCI1.h"
#include "rskrx140def.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 s_adc_count = 0;

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

main 関数に以下のハイライト表示されたコード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 *)" RSKRX140 ");
    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 s_adc_count and display using the LEDs */
            if (16 == (++s_adc_count))
            {
                s_adc_count = 0;
            }
            led_display_count(s_adc_count);

            /* Send the result to the UART */
            uart_display_adc(s_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_Config_S12AD0_Get_ValueResult(ADCHANNEL0, &adc_result);

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

            /* Increment the s_adc_count and display using the LEDs */
            if (16 == (++s_adc_count))
            {
                s_adc_count = 0;
            }
            led_display_count(s_adc_count);

            /* Send the result to the UART */
            uart_display_adc(s_adc_count, adc_result);
            /* Reset the flag */
            g_adc_complete = FALSE;
        }
        else
        {
            /* do nothing */
        }
    }
}
```

次に、ファイルの最後尾に以下の関数を追加してください。

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

'プロジェクト'メニューから 'すべてビルド'を選択するか、または  ボタンをクリックしてください。
e² studio はエラーのないプロジェクトを構築します。
プロジェクトは 6 章で説明するデバッガを使用して実行できるようになりました。コードは同じ動作をしますが、ユーザ LED は s_adc_count の値をバイナリ形式で表示します。

6. プロジェクトのデバッグ設定

プロジェクト・エクスプローラーで、'SC_Tutorial'プロジェクトが選択されていることを確認してください。
構成を入力するにはデバッグボタンの横にある矢印 をクリックし、「デバッグの構成(B)...」を選択してください。

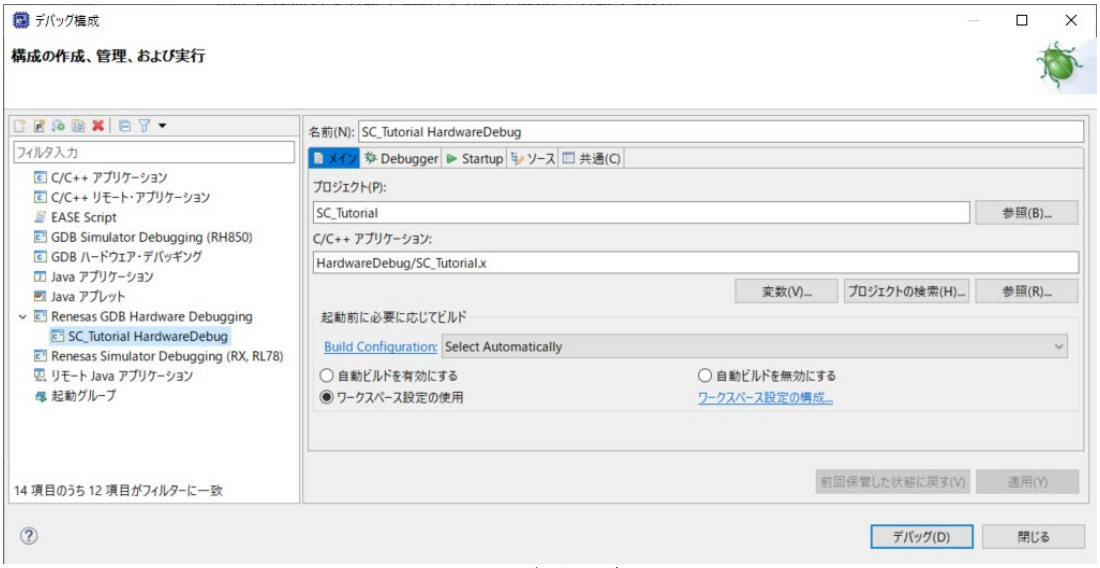


図 6-1 デバッグ構成

プロジェクトを実行するには、'Renesas GDB Hardware Debugging' -> 'SC_Tutorial HardwareDebug'->'Debugger'->'Connection Settings'にて以下の設定が必要です。
'エミュレーターから電源を供給する（MAX 200mA）'を<はい>に、 'EXTAL 周波数[MHz]'と'動作周波数[MHz]'に正しい周波数を設定してください。（値を入力した後に Enter キーを入力しないでください。）
これらは回路図から確認できます。（RSKRX140 の場合、EXTAL 周波数は 8.0000、動作周波数は 48.000）
RSKRX140 の電源投入の詳細については、ユーザーズマニュアルを参照してください。

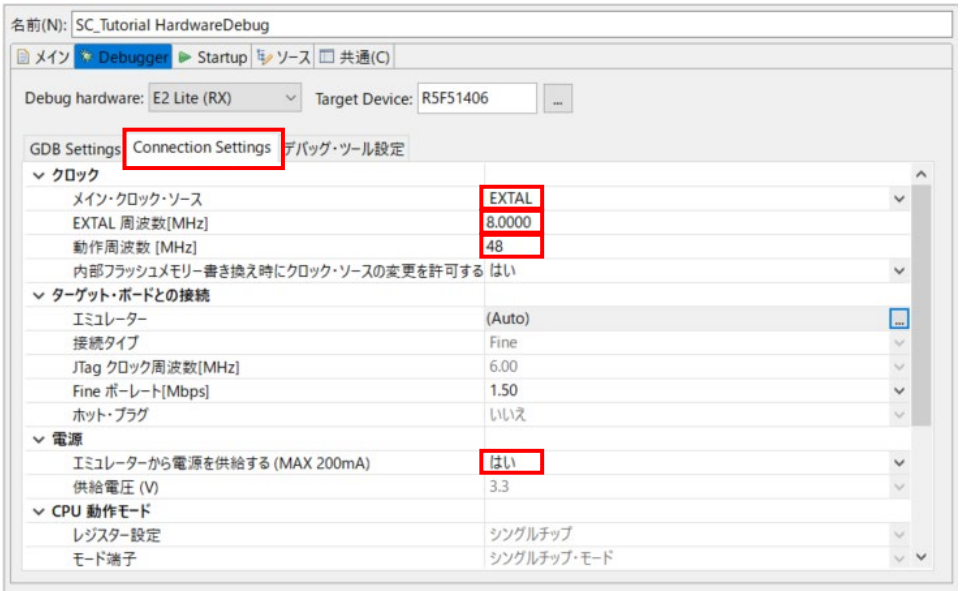


図 6-2 Connection Settings

設定が完了したら、'適用'ボタンに続けて'閉じる'ボタンを押して、デバッグ構成ウィンドウを閉じます。

CPU ボード上の RSK E2 Lite コネクタに E2 Lite を接続してください。また、PMOD1 コネクタに Pmod LCD を接続してください。
プロジェクト・エクスプローラーで、 'SC_Tutorial'プロジェクトが選択されていることを確認してください。
プロジェクトをデバッグするには、  ボタンをクリックします。**図 6-3** のダイアログが表示されます。

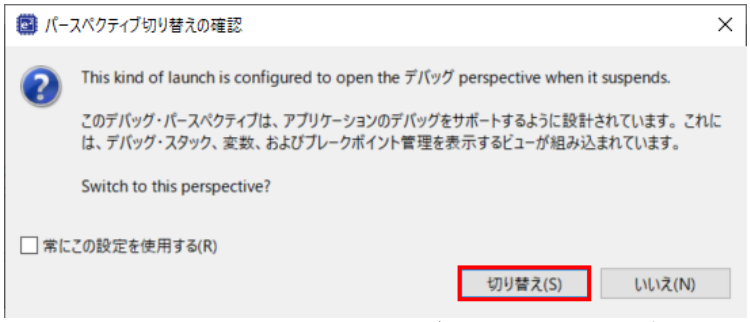


図 6-3 Perspective パースペクティブ切り替えの確認ダイアログ

このダイアログを後でスキップするには、'常にこの設定を使用する'をクリックします。[切り替え(S)]をクリックして、デバッグのパースペクティブが使用されることを確認してください。デバッガが起動すると、コードは**図 6-4** のようにスマート・コンフィグレータの PowerOn_Reset_PC 関数で停止します。

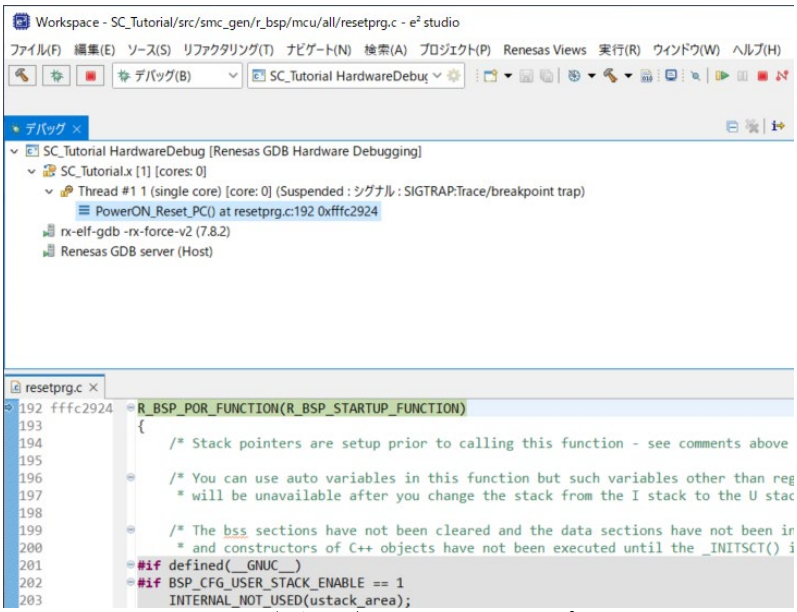
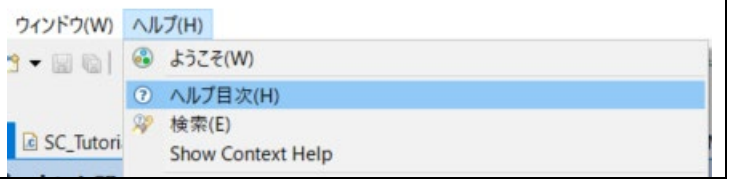


図 6-4 デバッガ・スタートアップ画面

e² stuido のデバッガの詳細についてはチュートリアルマニュアルを参照してください。コードを実行するには  ボタンをクリックしてください。デバッガは main 関数の先頭で停止します。もう一度  を押すと再度実行します。

7. 追加情報

サポート

e² studio の使用方法の詳細については、e² studio のメニューバーから’ヘルプ(H)’-> ‘ヘルプ目次(H)’を選択して、ヘルプファイルを参照してください。	

RX140 マイクロコントローラに関する情報は、RX140 グループ　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。

本製品に関する情報は、https://www.renesas.com/rskrx140 より入手可能です。

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	RX140 グループ Renesas Starter Kit for RX140 スマート・コンフィグレータ　チュートリアルマニュアル(e^2 studio)

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Jan.17.2022	―	初版発行

RX140 グループ
Renesas Starter Kit for RX140
スマート・コンフィグレータ　チュートリアルマニュアル(e^2 studio)

発行年月日　2022 年 01 月 17 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）

RX140 グループ

ルネサス エレクトロニクス株式会社

R20UT5032JG0100