User's Manual

32

# RX113 グループ

## Renesas Starter Kit
## コード生成支援ツール 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 2014.12

# 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、お客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
2. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
3. 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に関し、当社は、何らの責任を負うものではありません。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. 当社製品を改造、改変、複製等しないでください。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）、交通用信号機器、
   防災・防犯装置、各種安全装置等

   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（原子力制御システム、軍事機器等）に使用されることを意図しておらず、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お客様または第三者に損害が生じても、当社は一切その責任を負いません。なお、ご不明点がある場合は、当社営業にお問い合わせください。
6. 当社製品をご使用の際は、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他の保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障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については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、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。お客様が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. 本資料に記載されている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大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途に使用しないでください。当社製品または技術を輸出する場合は、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。
10. お客様の転売等により、本ご注意書き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、当社は何らの責任も負わず、お客様にてご負担して頂きますのでご了承ください。
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。

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。

(2012.4)

## 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RX用コード生成プラグインを使用して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。

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。

RSKRX113 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 | RSKRX113<br>ユーザーズマニュアル | R20UT2756JG |
| 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 | RSKRX113<br>チュートリアルマニュアル | R20UT2757JG |
| 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 | RSKRX113<br>クイックスタートガイド | R20UT2758JG |
| コード生成支援ツール<br>チュートリアルマニュアル | コード生成支援ツールの使用方法の説明 | RSKRX113<br>コード生成支援ツール<br>チュートリアルマニュアル | R20UT3254JG<br>(本マニュアル) |
| 回路図 | CPU ボードの回路図 | RSKRX113<br>CPU ボード回路図 | R20UT2755EG |
| ユーザーズマニュアル<br>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 | RX113 グループ<br>ユーザーズマニュアル<br>ハードウェア編 | R01UH0448JJ |

## 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 |
| 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 | 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 |
| MCU | Micro-controller Unit | マイクロコントローラユニット |
| PC | Personal Computer | パーソナルコンピュータ |
| 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。 |
| PLL | Phase-locked Loop | 位相同期回路 |
| 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 Plus | ルネサススタータキット |
| RTC | Realtime 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 |
| 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 |

# 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 RX 用コード生成プラグイン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：

- CS+プロジェクトの作成
- CS+プラグインのコード生成の使用
- 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 RX 用コード生成プラグイン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
- CS+コード生成プラグイン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と実行

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。最適化は行いません。
- ‘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 2）。

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 E1 エミュレータ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

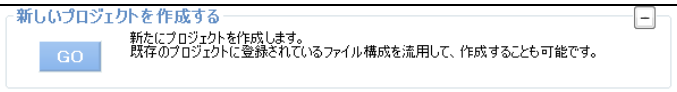
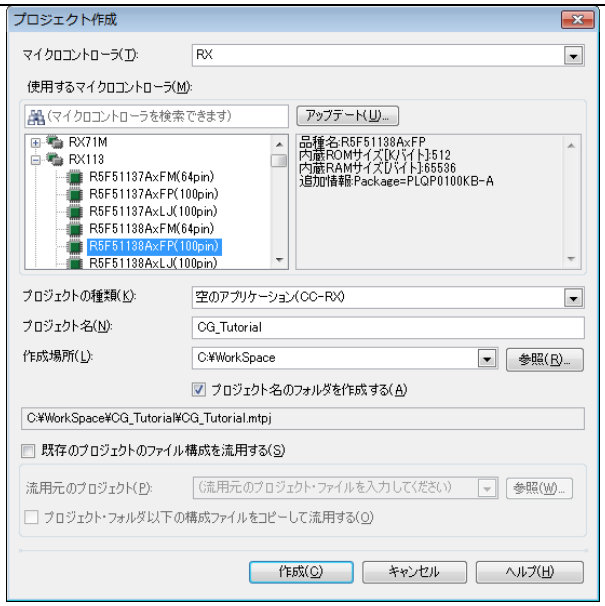
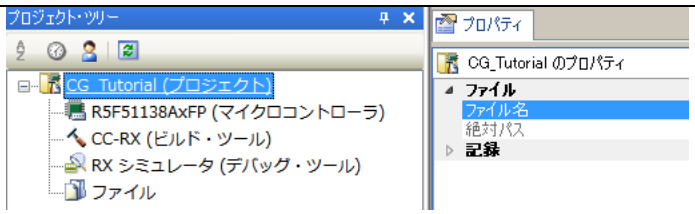
この章では RX113 マイクロコントローラのための新しい C ソースプロジェクトを作成するのに必要な手順をガイドします。
このプロジェクト作成の手順はマイクロコントローラ特有のプロジェクトを作成し、ソースをデバッグするのに必要です。

## 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™ Vista/7: スタートメニュー > すべてのプログラム > Renesas Electronics CS+ > CS+ for CC (RX, RH850)

Windows™ 8: ⓓをクリックして[アプリ]ビューを表示 > “CS+ for CC (RX, RH850)”アイコン

| | |
|---|---|
| • 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。<br>• マイクロコントローラー覧で下にスクロールします。‘RX113’の‘+’を展開してR5F51138AxFP(100pin)を選択してください。<br>• ‘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。<br>• ‘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。<br>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。<br>• ‘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・ツリーを持つ空のプロジェクトを生成します。事前にオプションでコード生成プラグインを有効にしていると、‘コード生成(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+コード生成プラグインによるコード生成

## 4.1 はじめに

コード生成は 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。コード生成は直感的な 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 CG_Tutorial と呼ばれる CS+プロジェクトを作成することができます。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、コード生成プラグイン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。

コード生成によって生成されるコードは、特定の周辺ごとに 3 つのコードを生成します（「r_cg_xxx.h」、「r_cg_xxx.c」、「r_cg_xxx_user.c」）。例えば A/D コンバータの場合、周辺を表す xxx は‘adc’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。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 */
```

コード生成の 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コード生成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。

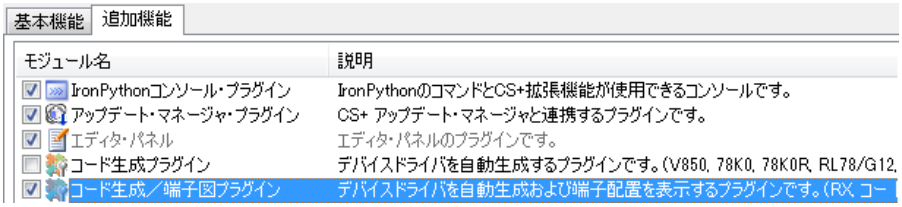
CG_Tutorial プロジェクトは、外部トリガによる A/D 変換モジュール、シリアルコミュニケーションインタフェース(SCI)と LCD ドライバを使用し、A/D 変換値をターミナルソフトや付属の LCD ディスプレイに表示します。

セクション 4.3 ではコード生成のユーザインタフェースについて、セクション 4.4 では各周辺機能ダイアログについて、6 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。

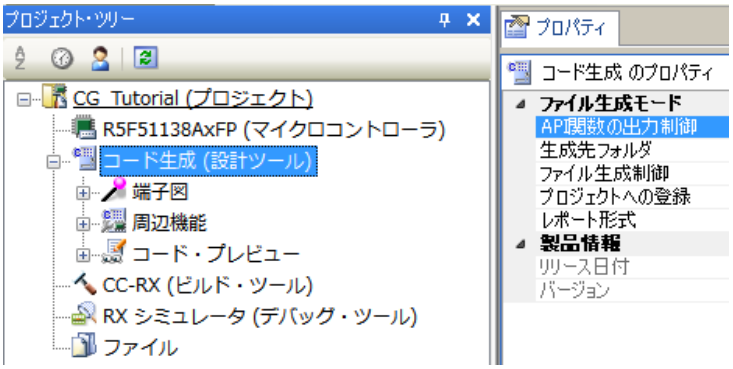
## 4.2 コード生成の有効化

CS+インストール後、コード生成プラグインを有効にする必要があります。この設定を一度だけ行えば CS+の設定情報は記憶されます。

CS+メイン画面上のツールバー「ツール」から'プラグインの管理'を選択してください。次に、'コード生成/端子図プラグイン'のチェックボックスをチェックしてください。




ダイアログ上の<OK>ボタンをクリックすると質問ダイアログが表示されるので、'はい(Y)'を選択してください。CS+は自動的に再起動し、'コード生成(設計ツール)'がプロジェクト・ツリー上に表示されます。



## 4.3 コード生成ツアー

このセクションでは、コード生成の簡単な操作方法を示しています。各操作の詳細については、Application Leading Tool 共通操作編ユーザーズマニュアル（R20UT2663)を参照してください。
Application Leading Tool は CS+にプラグインされていない独立したコード生成ツールです。Application Leading Tool 共通操作編ユーザーズマニュアルは、コード生成プラグインのマニュアルとしてもご利用いただけます。

プロジェクト・ツリーの  アイコンをクリックして‘コード生成’を展開します。同様に、  アイコンをクリックして‘周辺機能’を展開してください。次に何れかの周辺機能名をダブルクリックすることで、**図 4-1** に示すような周辺機能タブを含むメイン画面が表示されます。

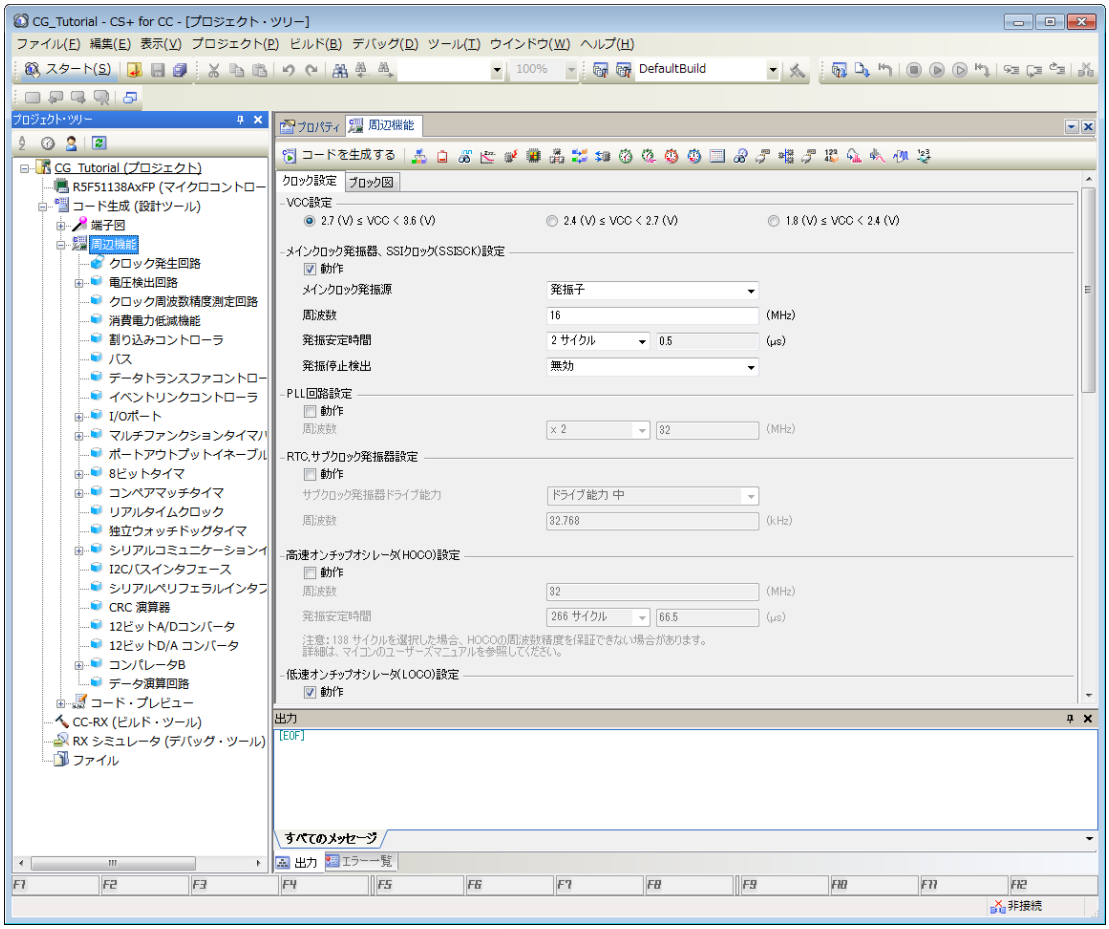



**図 4-1: 初期表示画面例**

コード生成は MCU 設定を GUI で操作することができます。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。

周辺機能はプロジェクト・ツリー内の周辺機能をダブルクリックするか、グラフィカルツールバーから周辺機能アイコンをクリックすることで設定できます。

プロジェクト・ツリー内のプロジェクトからコード・プレビューにある周辺機能をダブルクリックすることで生成されるコードをプレビューできます。

## 4.4 コード生成

このセクションでは、外部 SW トリガによる A/D 変換モジュール、シリアルコミュニケーションインタフェース（SCI）と LCD 出力するための機能を含むチュートリアルプロジェクト設定手順を示します。

### 4.4.1 クロック発生回路

クロック発生回路を**図 4-2** に示します。

チュートリアルでは、メインクロック発振源に 16MHz 水晶発振子を使用し、PLL 回路でメインクロックを逓倍します。サブクロックは、LCD ソースクロックに使用します。

プロジェクト・ツリーの‘コード生成’ -> ‘周辺機能’ -> ‘クロック発生回路’をダブルクリックしてください。
図 4-2 を参照して設定してください。

次にポートを設定します。

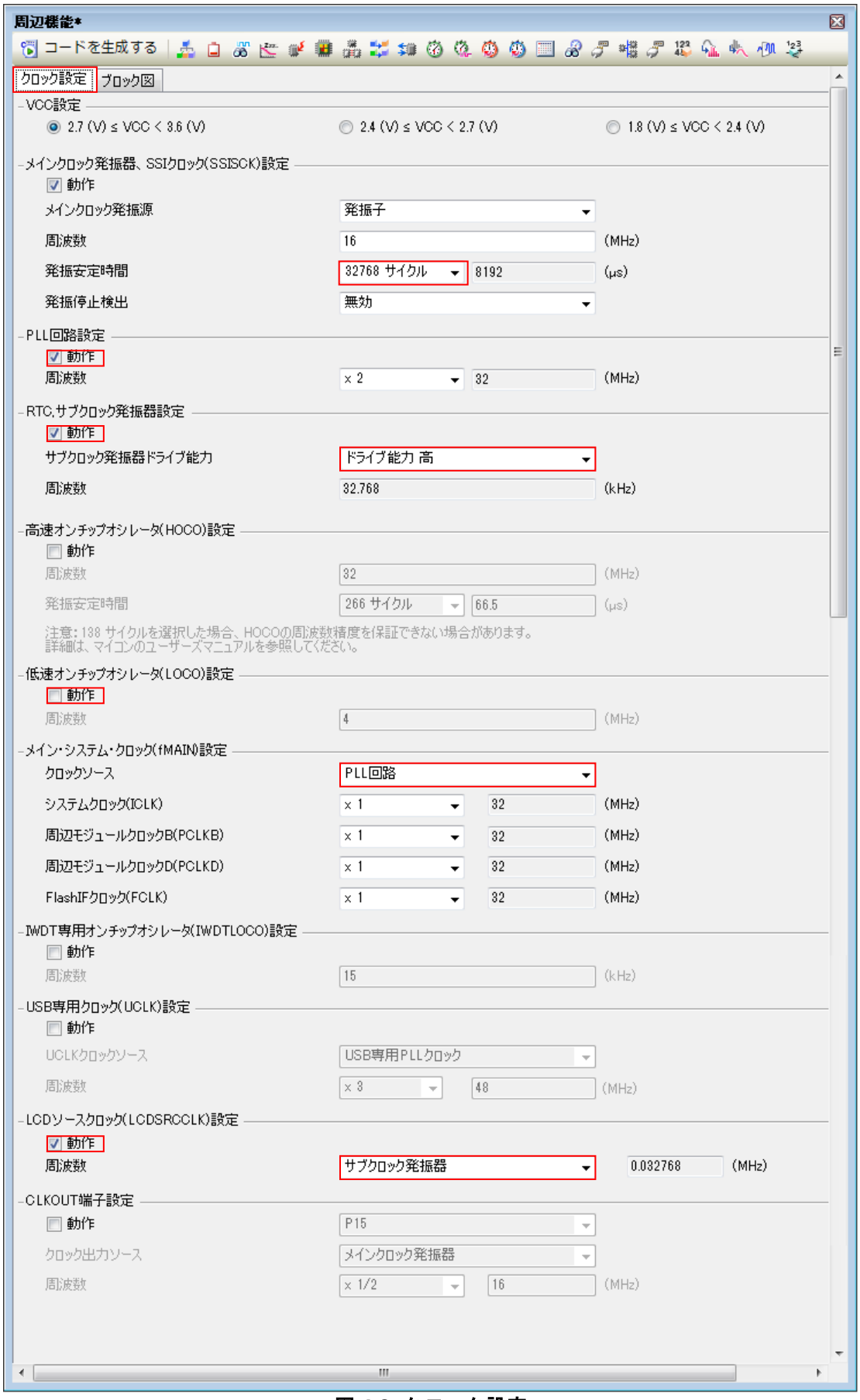

周辺機能*
コードを生成する
クロック設定
ブロック図
VCC設定
2.7 (V) ≤ VCC < 3.6 (V)
2.4 (V) ≤ VCC < 2.7 (V)
1.8 (V) ≤ VCC < 2.4 (V)
メインクロック発振器、SSIクロック(SSISCK)設定
動作
メインクロック発振源
発振子
周波数
16
(MHz)
発振安定時間
32768 サイクル
8192
(μs)
発振停止検出
無効
PLL回路設定
動作
周波数
× 2
32
(MHz)
RTC,サブクロック発振器設定
動作
サブクロック発振器ドライブ能力
ドライブ能力 高
周波数
32.768
(kHz)
高速オンチップオシレータ(HOCO)設定
動作
周波数
32
(MHz)
発振安定時間
266 サイクル
66.5
(μs)
注意：138 サイクルを選択した場合、HOCOの周波数精度を保証できない場合があります。
詳細は、マイコンのユーザーズマニュアルを参照してください。
低速オンチップオシレータ(LOCO)設定
動作
周波数
4
(MHz)
メイン・システム・クロック(fMAIN)設定
クロックソース
PLL回路
システムクロック(ICLK)
× 1
32
(MHz)
周辺モジュールクロックB(PCLKB)
× 1
32
(MHz)
周辺モジュールクロックD(PCLKD)
× 1
32
(MHz)
FlashIFクロック(FCLK)
× 1
32
(MHz)
IWDT専用オンチップオシレータ(IWDTLOCO)設定
動作
周波数
15
(kHz)
USB専用クロック(UCLK)設定
動作
UCLKクロックソース
USB専用PLLクロック
周波数
× 3
48
(MHz)
LCDソースクロック(LCDSRCCLK)設定
動作
周波数
サブクロック発振器
0.032768
(MHz)
CLKOUT端子設定
動作
P15
クロック出力ソース
メインクロック発振器
周波数
× 1/2
16
(MHz)

**図 4-2: クロック設定**

### 4.4.2 ポート設定

ポート設定では、LED ポート出力設定およびユーザスイッチ設定を行います。ユーザスイッチの SW3 は、A/D 変換開始トリガとして使用されるため、セクション 4.4.4 にて設定します。詳細については、回路図を参照してください。

プロジェクト・ツリーの‘コード生成’ -> ‘周辺機能’ -> ‘I/O ポート’をダブルクリックしてください。
Port2、Port3、PortJ を設定します。‘Port2’タブ、‘Port3’タブ、‘PortJ’タブを各々設定してください。

各ポートの設定は、**図 4-3、図 4-4、図 4-5** を参照して設定してください。

次にシリアルコミュニケーションインタフェースを設定します。

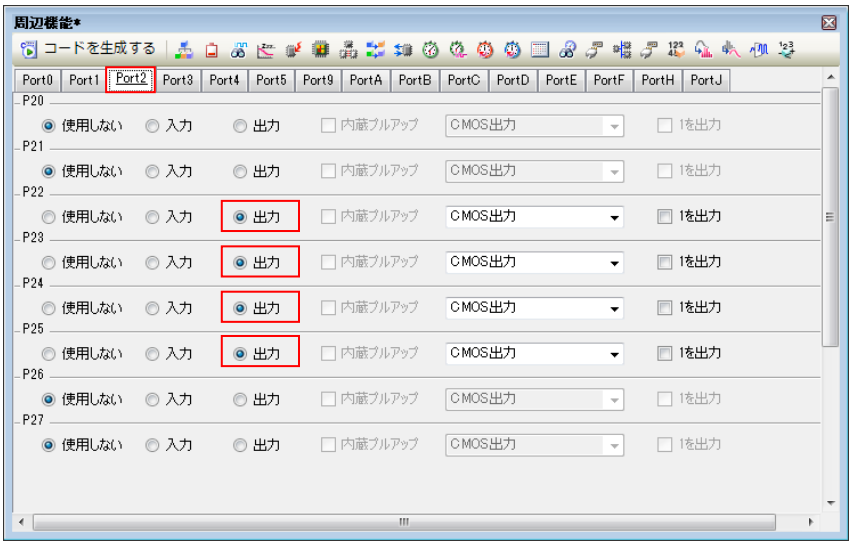



**図 4-3: Port2 設定**

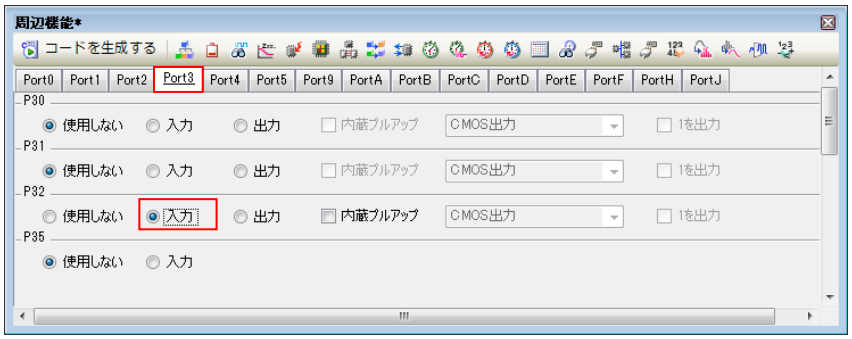



**図 4-4: Port3 設定**

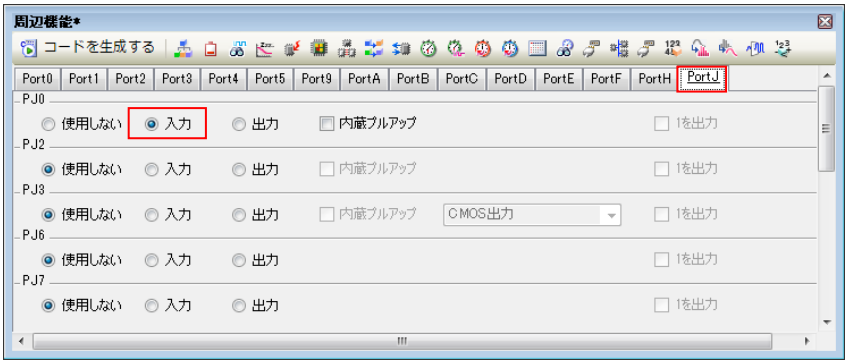



**図 4-5: PortJ 設定**

### 4.4.3 シリアルコミュニケーションインタフェース

シリアルコミュニケーションインタフェース設定では、SCI1 の設定を行います。SCI1 チャネルを使用し、USB シリアル変換および PC ターミナルへのデータ転送を実行します。

プロジェクト・ツリーの‘コード生成’ -> ‘周辺機能’ -> ‘シリアルコミュニケーションインタフェース’をダブルクリックしてください。
‘SCI1’タブの‘一般設定’および‘設定’タブを**図 4-6、図 4-7** を参照して設定してください。

SCI1 チャネルの設定は、調歩同期式: 送信/受信、データ・ビット長設定: 8 ビット、パリティ設定: パリティなし、ストップビット設定: 1 ビット、転送速度設定: ビットレートを 19200bps に設定してください。

次に 12 ビット A/D コンバータを設定します。

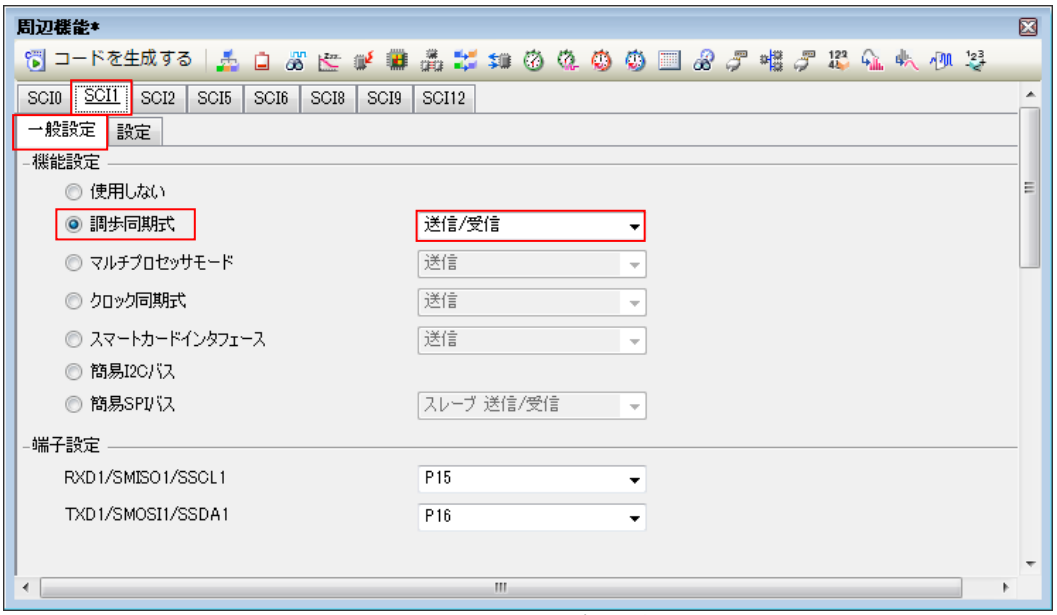



**図 4-6: SCI1 一般設定**

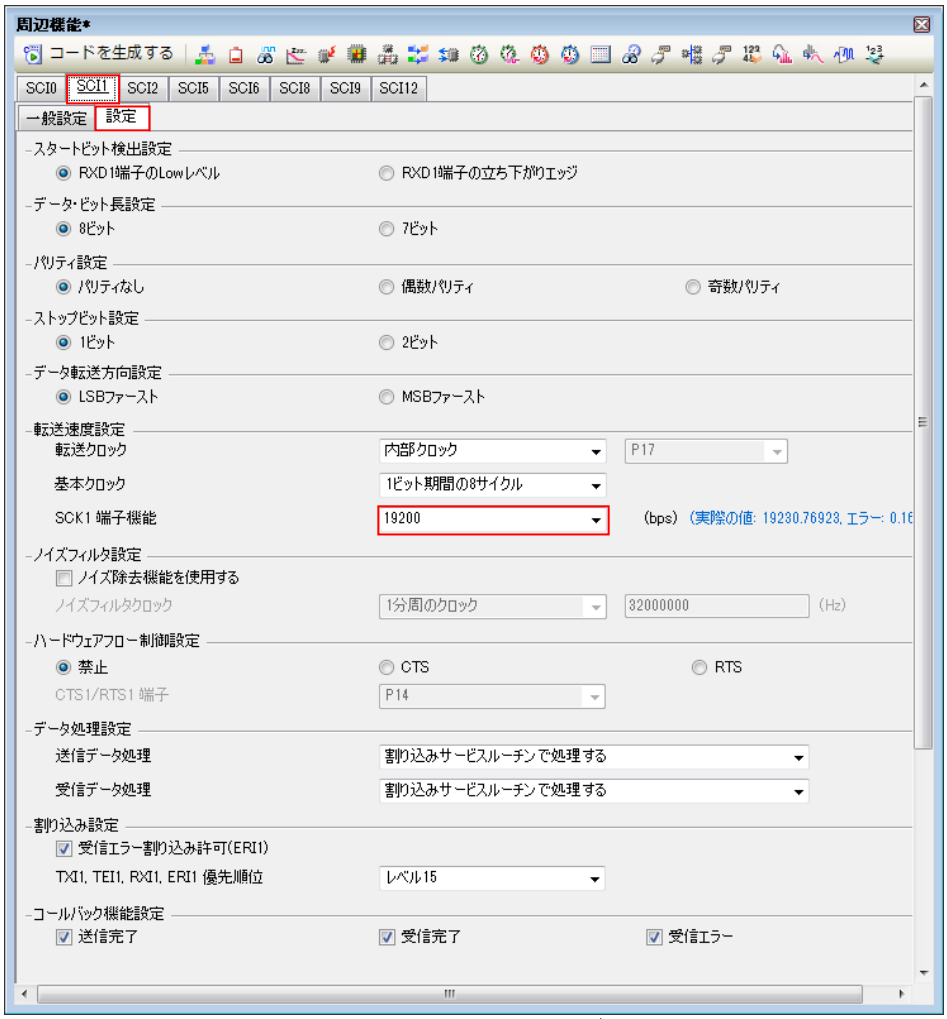



**図 4-7: SCI1 設定タブ**

### 4.4.4 12 ビット A/D コンバータ

12 ビット A/D コンバータの設定を行います。12 ビット A/D コンバータは CPU ボード上のポテンショメータ RV1 から入力される電圧を取得します。12 ビット A/D コンバータは、CPU ボード上の SW3 を押すことで A/D 変換を開始します。

プロジェクト・ツリーの‘コード生成’ -> ‘周辺機能’ -> ‘12 ビット A/D コンバータ’をダブルクリックしてください。
‘一般的な設定’および‘設定’タブを**図 4-8、図 4-9** を参照して設定してください。

これで周辺機能の設定は全て完了しました。次のセクションに進んでください。

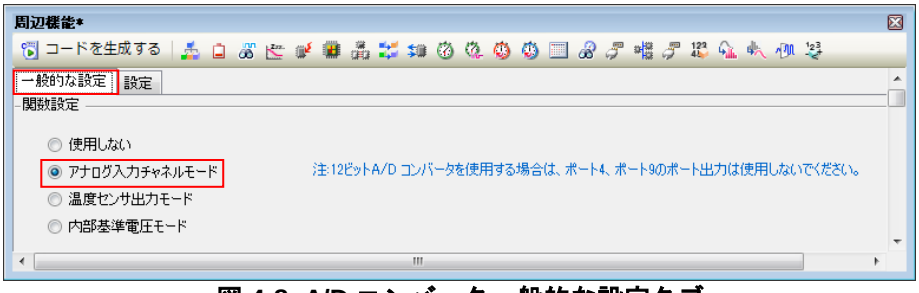



**図 4-8: A/D コンバーター般的な設定タブ**

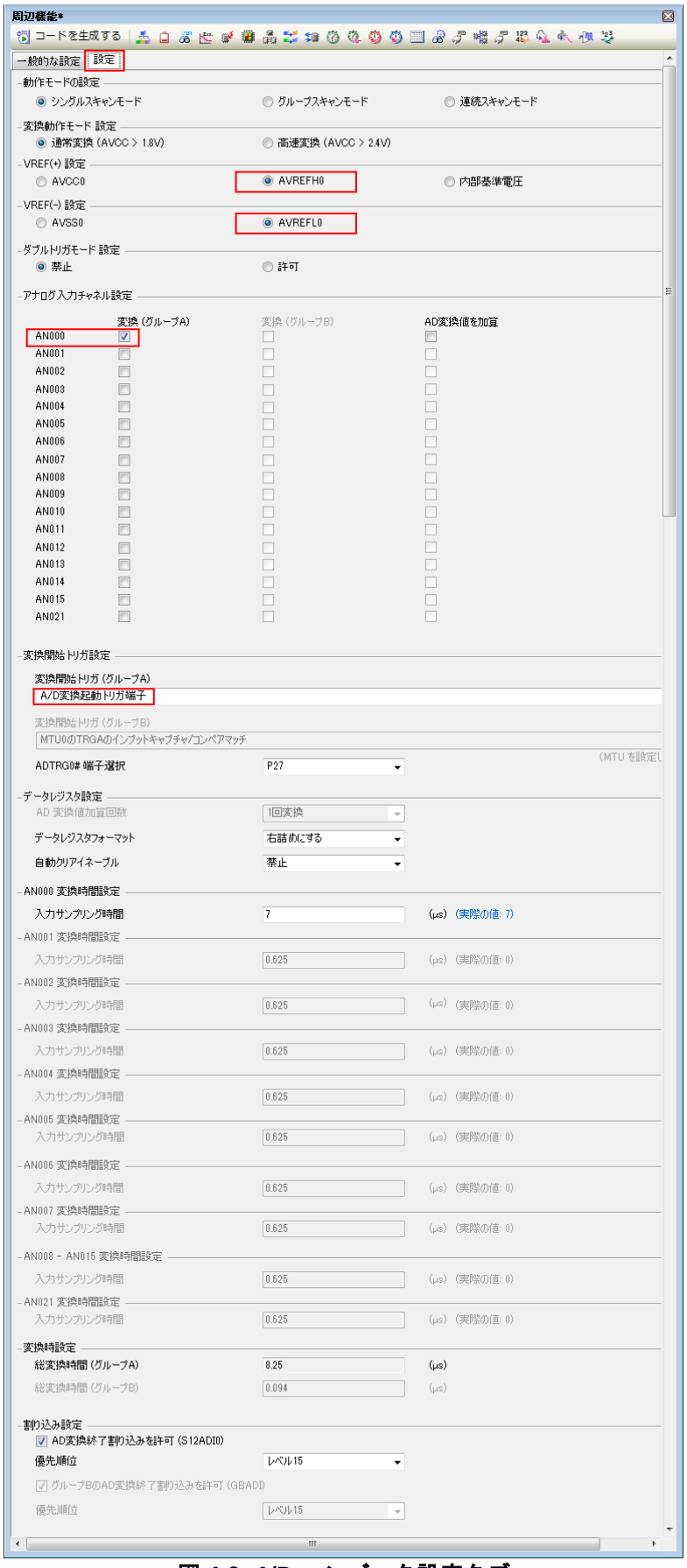



**図 4-9: A/D コンバータ設定タブ**

### 4.4.5 生成コード

周辺機能の設定は全て完了しました。次に、<コードを生成する>ボタンをクリックして、コードを生成してください。
**図 4-10** の示すようにコード生成画面に'ファイルの生成を完了しました。'が表示されます。

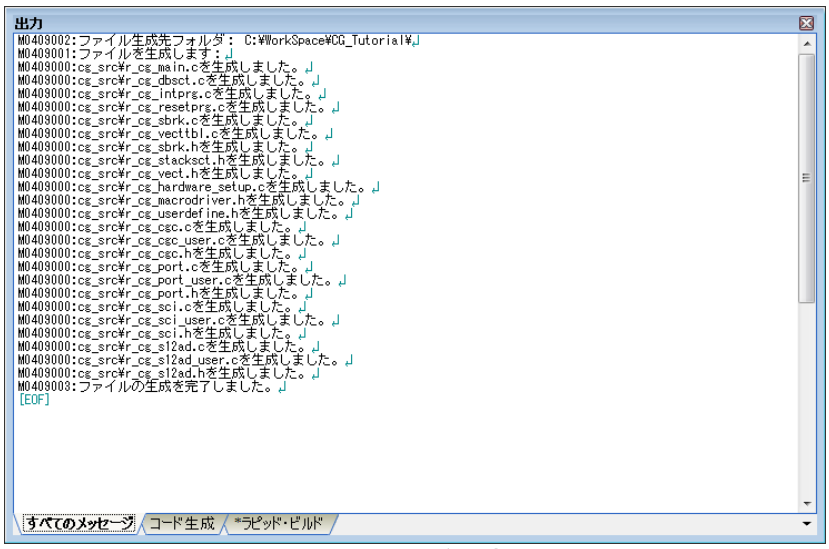



**図 4-10: コード生成画面**

**図 4-11** はプロジェクト・ツリー中のコード生成ファイルを示します。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 CG_Tutorial が完成します。

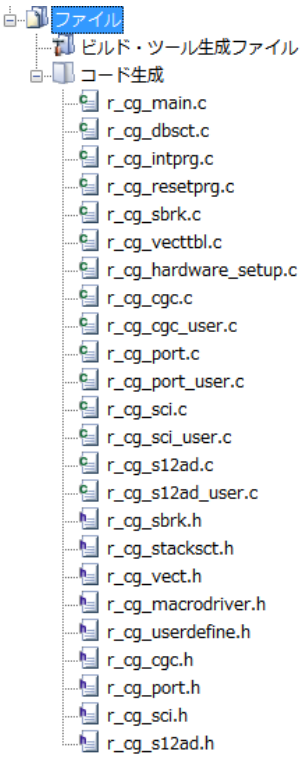
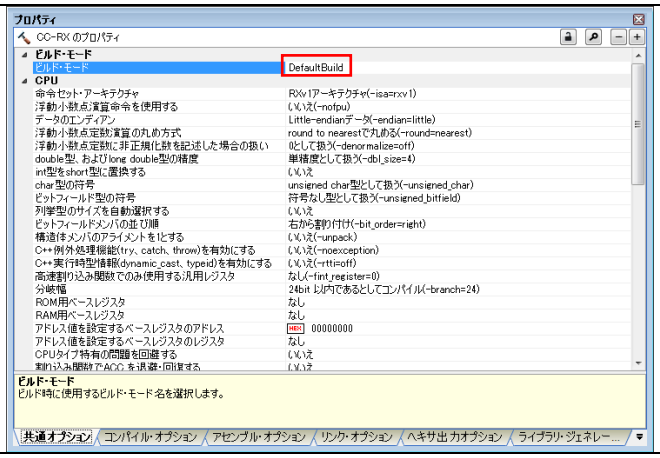



**図 4-11: プロジェクト・ツリー中のコード生成ファイル**

# 5. Tutorial プロジェクトの設定

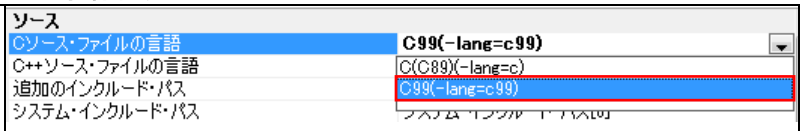
このセクションでは、RSK を実行するために空プロジェクトの設定変更を行います。

- 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。




**コンパイル・オプション-C ソース・ファイルの言語設定**

- 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。



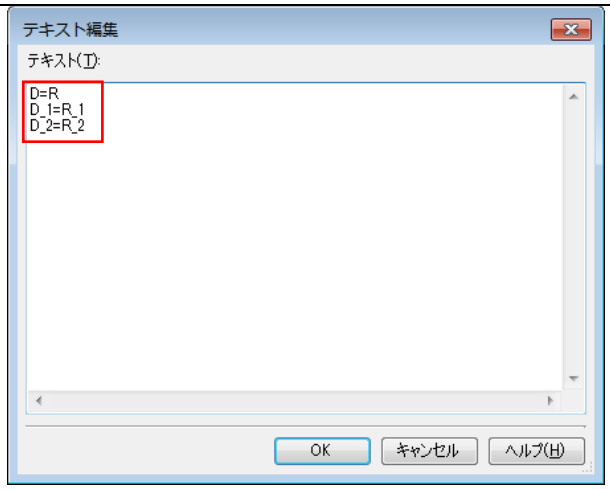

**リンク・オプション-ROM から RAM へマップするセクション設定**

- 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。

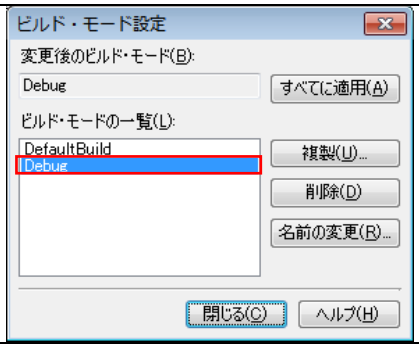
  D=R
  D_1=R_1
  D_2=R_2

  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。

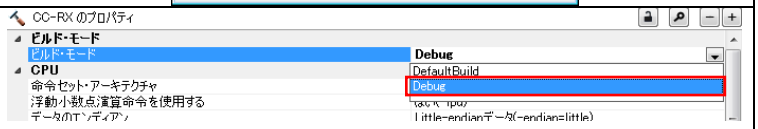


**共通オプション-ビルド・モード(Debug)設定**

- ビルドメニューから'ビルド・モードの設定'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。



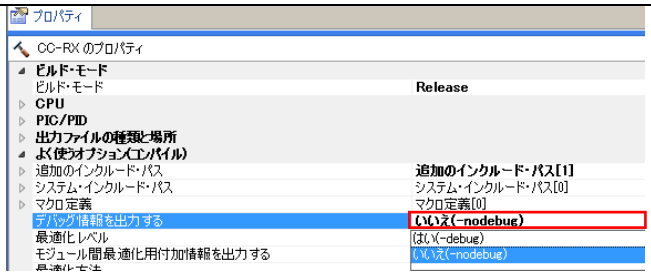

- ビルドプロパティ画面に戻り画面下の'共通オプション'タブをクリックしてください。'ビルド・モード'のプルダウンから先ほど追加した'Debug'を選択してください。




- 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ないようになります。
  チュートリアルでは、Debug ビルド・モードを使用します。

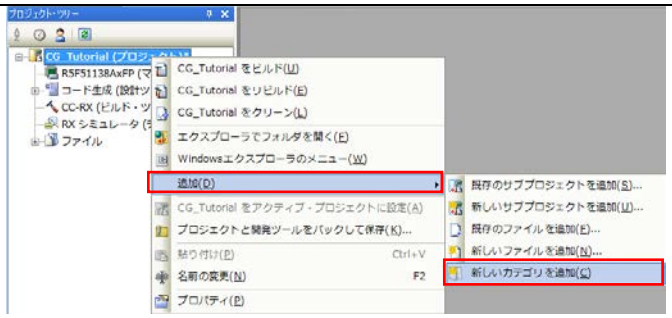
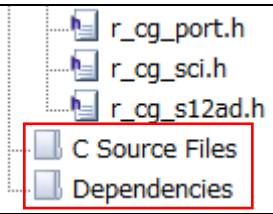

**共通オプション-ビルド・モード(Release)設定**

- 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、'デバッグ情報を生成する'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。




- チュートリアルでは、Debug ビルド・モードを使用するため、Release ビルド・モードから Debug ビルド・モードに戻してください。
- 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。

## 5.1 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。<br>• プロジェクト・ツリー内の CG_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。<br>• 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。 |  |

# 6. ユーザコードの統合

通常、開発プロジェクトはアプリケーションの要求に応じてコードを作成します。本チュートリアルでは、デモンストレーションとしてクイックスタートガイドの手順に従って作成した‘Tutorial’プロジェクトのコードとファイルを利用して、プロジェクトの完成を目指します。

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 _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することができます。

## 6.1 RSK チュートリアル用ファイルのコピー

RSK チュートリアル用に以下のファイルを提供します。

iodefine.h,
r_rsk_utility.c,
r_rsk_utility.h,
rskrx113def.h.

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から‘CG_Tutorial’フォルダに上記のファイルをコピーしてください。‘CG_Tutorial’フォルダの格納場所は、セクション 3.2 で指定した場所です。

## 6.2 LCD ファイルのコピー

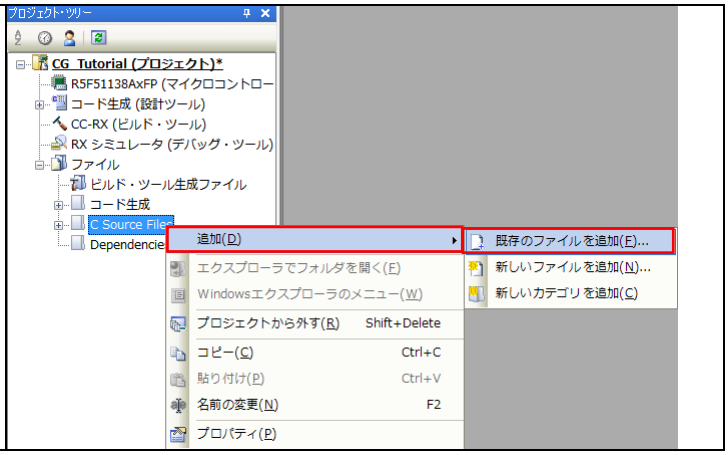
LCD パネルのための API 機能は、以下のファイルに含まれています。

r_lcd_appv2.c
r_lcd_appv2.h

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から‘CG_Tutorial’フォルダに上記のファイルをコピーしてください。‘CG_Tutorial’フォルダの格納場所は、セクション 3.2 章で指定した場所です。

## 6.3 CS+プロジェクトへファイルを追加

- ‘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。
- 以下のファイルを追加してください。
  r_rsk_utility.c
  r_lcd_appv2.c
- 同様に、‘Dependencies’フォルダに以下のファイルを追加してください。
  iodefine.h
  r_rsk_utility.h
  rskrx113def.h
  r_lcd_appv2.h



## 6.4 コード生成ファイルへのコード追加

このセクションでは、新しくコード生成されたファイルにコードを追加します。

コード生成ファイルは、CS+プロジェクトのプロジェクト・ツリーの‘ファイル’ -> ‘コード生成’をダブルクリックして開いてください。

以降のセクションに従って指定のファイルを展開します。次に、このドキュメントに記載されている赤枠内の内容をコピーし、展開したファイルの指定箇所にペーストしてください。

### 6.4.1 r_cg_userdefine.h コード追加

CS+プロジェクトのプロジェクト・ツリーの‘r_cg_userdefine.h’をダブルクリックして開いてください。

次に、ファイルの最後尾にある function 用ユーザコードコメント文の間に以下のカスタムコードを追加してください。

```
/* 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 */
```

### 6.4.2 r_cg_s12ad.c コード追加

CS+プロジェクトのプロジェクト・ツリーの‘r_cg_s12ad.c’をダブルクリックして開いてください。

次に、ファイルの最後尾にあるユーザコードエリアコメント文の間に以下のカスタムコードを追加してください。

```
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************************
 * Function Name: R_S12AD_SWTriggerStart
 * Description  : This function starts the AD0 converter.
 * Arguments    : None
 * Return Value : None
 ***********************************************************************************/
void R_S12AD_SWTriggerStart (void)
{
    S12AD.ADCSR.BIT.ADST = 1U;
}

/***********************************************************************************
 End of function R_S12AD_SWTriggerStart
 ***********************************************************************************/

/***********************************************************************************
 * Function Name: R_S12AD_SWTriggerStop
 * Description  : This function stops the AD0 converter.
 * Arguments    : None
 * Return Value : None
 ***********************************************************************************/
void R_S12AD_SWTriggerStop (void)
{
    S12AD.ADCSR.BIT.ADST = 0U;
}

/***********************************************************************************
 End of function R_S12AD_SWTriggerStop
 ***********************************************************************************/

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

### 6.4.3 r_cg_s12ad.h コード追加

CS+プロジェクトのプロジェクト・ツリーの‘r_cg_s12ad.h’をダブルクリックして開いてください。

次に、ファイルの最後尾にある function 用ユーザコードエリアコメント文の間に以下のカスタムコードを追加してください。

```
/* 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_S12AD_SWTriggerStart (void);
void R_S12AD_SWTriggerStop (void);
/* End user code. Do not edit comment generated here */
```

### 6.4.4 r_cg_s12ad_user.c コード追加

CS+プロジェクトのプロジェクト・ツリーの‘r_cg_s12ad_user.c’をダブルクリックして開いてください。

次に、global 用ユーザコードエリアコメント文の間、ユーザコードエリアコメント文の間に以下のカスタムコードを追加してください。

```
/* 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 */
```

以下のカスタムコードを **static void r_s12ad_interrupt(void)**に追加してください。

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

    /* Flag that the ADC had completed a sample */
    g_adc_complete = 1;

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

### 6.4.5 r_cg_sci_user.c コード追加

CS+プロジェクトのプロジェクト・ツリーの‘r_cg_sci_user.c’をダブルクリックして開いてください。

次に、global 用ユーザコードエリアコメント文の間、ユーザコードエリアコメント文の間に以下のカスタムコードを追加してください。

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

/* Flag used locally to detect transmission complete */
static volatile uint8_t sci1_txdone;

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

以下のカスタムコードを **static void r_sci1_callback_transmitend(void)**に追加してください。

```
    /* Start user code. Do not edit comment generated here */
    sci1_txdone = TRUE;

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

以下のカスタムコードを **static void r_sci1_callback_receiveend(void)**に追加してください。

```
    /* Start user code. Do not edit comment generated here */
    /* Check the contents of g_rx_char */

        g_rx_char = g_rx_char & 0xDF;         /* Ensure ASCII char is in upper case */

        /* Check for the 'c' trigger command */
    if ('C' == g_rx_char)
    {
         g_adc_trigger = TRUE;
    }

    /* Set up SCI1 receive buffer and callback function again */
    R_SCI1_Serial_Receive((uint8_t *)&g_rx_char, 1);

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

以下のカスタムコードをファイル最後尾にあるユーザコードエリアコメント文の間に追加してください。

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

### 6.4.6 r_cg_sci.h コード追加

CS+プロジェクトのプロジェクト・ツリーの‘r_cg_sci.h’をダブルクリックして開いてください。

次に、function 用ユーザコードエリアコメント文の間に以下のカスタム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_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;

/* Flag used to control transmission to PC terminal */
extern volatile uint8_t g_tx_flag;
/* End user code. Do not edit comment generated here */
```

### 6.4.7 r_cg_main.c コード追加

CS+プロジェクトのプロジェクト・ツリーの‘r_cg_main.c’をダブルクリックして開いてください。

次に、include 用ユーザコードエリアコメント文の間に以下のカスタムコードを追加してください。

```
/* Start user code for include. Do not edit comment generated here */
#include "r_cg_s12ad.h"
#include "r_lcd_appv2.h"
#include "r_rsk_utility.h"
#include "rskrx113def.h"

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

global 用ユーザコードエリアコメント文の間に以下のカスタムコードを追加してください。

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

/* Welcome banner - displayed on serial port at startup*/
static uint8_t welcome_banner[] = "RSK RX113 - Tutorial - Press 'c' or SW3 for ADC Conversion\r\n\0";

/* 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 ADC conversion count for user display */
static uint8_t adc_count = 0;

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

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

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

以下のカスタムコードを **void main(void)**に追加してください。
注：コード生成された while (1U)ループごと上書きしてください。

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

    /* Display Project Title on LCD*/
    R_LCD_DisplayPanelString( PANEL_LCD_LINE1, (uint8_t*) "TUTOR");

    /* Set up SCI1 receive buffer and callback function */
    R_SCI1_Serial_Receive((uint8_t *) &g_rx_char, 1);

    /* Enable SCI1 operations */
    R_SCI1_Start();

    /* Display Welcome Banner on Serial Port */
    R_SCI1_AsyncTransmit(welcome_banner, sizeof(welcome_banner));
```

```
    while (1U)
    {
        uint16_t adc_result;

        /* If the user has requested ADC sample via the serial port */
        if (TRUE == g_adc_trigger)
        {
            /* Call the function to perform an ADC conversion */
            adc_result = get_adc();

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

            /* Display count on LEDs */
            led_display_count(adc_count);

            /* Send the result to SCI1 UART */
            uart_display_adc(adc_count, adc_result);

            /* Increment the adc_count and check roll over */
            if (16 == (++adc_count))
            {
                adc_count = 0;
            }

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

        /* SW3 is directly wired into the ADTRG0n pin so will
         cause the conversion and interrupt */
        else if (TRUE == g_adc_complete)
        {
            /* Get the result of the ADC conversion */
            R_S12AD_Get_ValueResult(ADCHANNEL0, &adc_result);

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

            /* Display count on LEDs */
            led_display_count(adc_count);

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

            /* Increment the adc_count and check roll over */
            if (16 == (++adc_count))
            {
               adc_count = 0;
            }

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

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

以下のカスタムコードを **void R_MAIN_UserInit(void)**に追加してください。

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

    /* Initialise the LCD for the RSK LCD APP V2 display board */
    R_LCD_Create();
    R_LCD_Start();

    /* Start the ADC */
    R_S12AD_Start();

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

ファイルの最後尾にあるユーザコードエリアコメント文の間に以下のカスタムコードを追加してください。

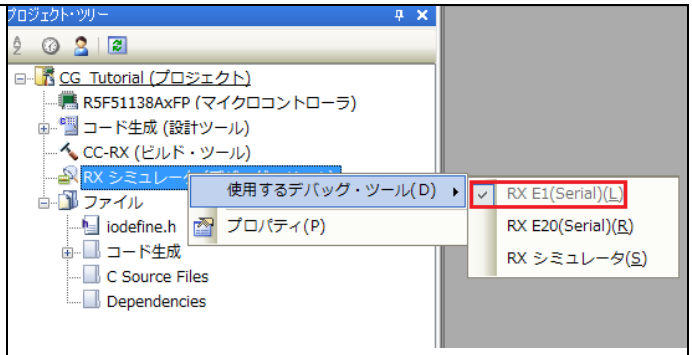
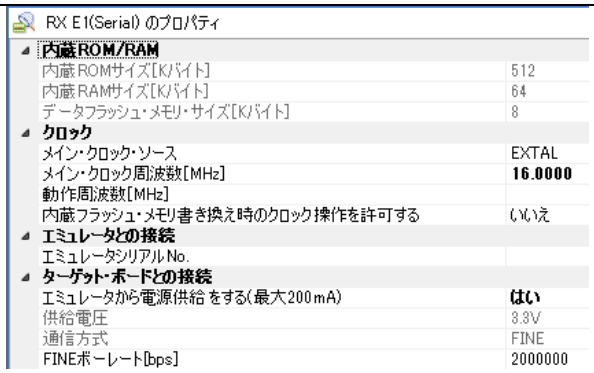
```
/* Start user code for adding. Do not edit comment generated here */
/***********************************************************************
 * Function Name : get_adc
 * Description   : Creates a ADC12 Software trigger and returns the ADC result,
 *                           once the ADC conversion is complete.
 * Argument      : none
 * Return value  : uint16_t ADC sample value
 ***********************************************************************/
static uint16_t get_adc (void)
{
    /* A variable to retrieve the ADC result */
    uint16_t adc_result;

    /* Start a conversion */
    R_S12AD_SWTriggerStart();

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

    /* Stop conversion */
    R_S12AD_SWTriggerStop();

    /* Clear ADC flag */
    g_adc_complete = FALSE;

    R_S12AD_Get_ValueResult(ADCHANNEL0, &adc_result);

    /* Set AD conversion start trigger source back to ADTRG0n pin */
    R_S12AD_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 temporary character string */
    char lcd_buf[4];

    /* Convert ADC result into a character string, and store in the
     local string lcd_buffer */
    uint16_to_string(lcd_buf, 0u, adc_result);

    /* Display the ADC value - Line 3 provides three
     * characters, so skip the unused leading zero
     */
    R_LCD_DisplayPanelString( PANEL_LCD_LINE3, (uint8_t *) lcd_buf + 1);
}

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

```
/***********************************************************************
 * 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 uint8_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. */
    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_SCI1_AsyncTransmit(uart_buffer, sizeof(uart_buffer));
}

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

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

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

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

# 7. プロジェクトのビルド、デバッグ設定

メニューバーの‘ビルド(B)’から‘ビルド・プロジェクト(B)’または‘F7’キーを選択してください。エラーが発生していないことを確認してください。

以下に、E1 デバッガとボード設定を示します。

| | |
|---|---|
| • RX シミュレータ（デバッグ・ツール）を右クリックし、RX E1（Serial）を選択してください。 |  |
| • RX E1（Serial）を右クリックし、プロパティを選択してください。<br>• 接続用設定タブをクリックしてしてください。<br>メイン・クロック周波数[MHz]：16.0000<br>エミュレータから電源供給をする：はい<br>• その他はデフォルト設定のままにしてください。 |  |
| • E1 をコンピュータの USB ポート、CPU ボードに接続してください。<br>• LCD Application Board V2 が CPU ボードの JA4 コネクタに正しく接続されているか確認してください。<br>• 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。 |  |

## 7.1 チュートリアルコードの実行

コードを実行する前に、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 ‘RL78G1C-USB’）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。

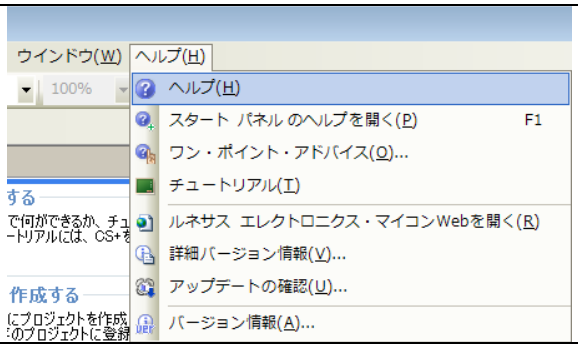
デバイスマネージャ上のポート(COM と LPT)に‘RSK USB Serial Port (COMx)’が現れますので、COM ポート番号を確認し、ターミナルソフトを起動して確認した COM ポート番号の設定を行ってください（ボーレート：19200、データ長：8、パリティ：なし、ストップビット：1）。

| | |
|---|---|
| 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。 |  |
| コードが 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、コードを実行することができます。現在のプログラムカウンタ位置からコードを始めるため‘実行’ボタンをクリックしてください。 |  |

コードは、ターミナル画面に‘RSK RX113 - Tutorial - Press 'c' or SW3 for ADC Conversion’、LCD パネルの左下位置に‘TUTOR’と表示します。また、CPU ボード上の SW3 を押すとポテンショメータによって調整された電圧値の A/D 変換が行われ、LCD およびターミナル画面に A/D 変換結果を表示します。

# 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。 |  |

RX113 マイクロコントローラに関する詳細情報は、RX113 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、情報等は以下のウェブサイトより入手可能です：
http://japan.renesas.com/rskrx113 （日本サイト）
http://www.renesas.com/rskrx113 （グローバルサイト）

**オンライン技術サポート**
技術関連の問合せは、以下を通じてお願いいたします。
日本： csc@renesas.com
グローバル： csc@renesas.com

ルネサスのマイクロコントローラに関する総合情報は、以下のウェブサイトより入手可能です：
http://japan.renesas.com/ （日本サイト）
http://www.renesas.com/ （グローバルサイト）

**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 | RSKRX113 コード生成支援ツール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 | 2014.12.03 | ー | 初版発行 |
| | | | |

RSKRX113 コード生成支援ツールチュートリアルマニュアル(CS+)

発行年月日　2014 年 12 月 3 日  Rev.1.00

発行　　ルネサス エレクトロニクス株式会社
　　〒211-8668　神奈川県川崎市中原区下沼部 1753

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 〒100-0004 千代田区大手町2-6-2（日本ビル）

■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：http://japan.renesas.com/contact/

# RX113 グループ

R20UT3254JG0100

ルネサス エレクトロニクス株式会社