

RL78/G1G

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

16 ビット・シングルチップ・マイクロコントローラ RL78 ファミリ

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して、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。

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っていると、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RL78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。

RSKRL78G1G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	RSKRL78G1G ユーザーズマニュアル	R20UT3022JG
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グ方法の説明	RSKRL78G1G チュートリアルマニュアル	R20UT3019JG
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	RSKRL78G1G クイックスタートガイド	R20UT3020JG
コード生成支援ツール 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	RSKRL78G1G コード生成支援ツール チュートリアルマニュアル	R20UT3021JG (本マニュアル)
回路図	CPUボードの回路図	RSKRL78G1G CPUボード回路図	R20UT3017EG
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	RL78/G1G ユーザーズマニュアル ハードウェア編	R01UH0499JJ

2. 略語および略称の説明

略語／略称	英語名	備考
ADC	Analog-to-Digital Converter	A/D コンバータ
API	Application Programming Interface	アプリケーションプログラムインタフェース
Bps	Bits per second	転送速度を表す単位、ビット/秒
CMT	Compare Match Timer	コンペアマッチタイマ
COM	COMmunications port referring to PC serial port	シリアル通信方式のインタフェース
CPU	Central Processing Unit	中央処理装置
DVD	Digital Versatile Disc	デ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
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	ルネサススタータキット
RTC	Realtime Clock	リアルタイムクロック
SAU	Serial Array Unit	シリアルアレイユニット
SCI	Serial Communications Interface	シリアルコミュニケーションインタフェース
SPI	Serial Peripheral Interface	シリアルペリフェラルインタフェース
TAU	Timer Array Unit	タイマアレイユニット
TFT	Thin Film Transistor	薄膜トランジスタ
TPU	Timer Pulse Unit	タイマパルスユニット
UART	Universal Asynchronous Receiver/Transmitter	調歩同期式シリアルインタフェース
USB	Universal Serial Bus	シリアルバス規格の一種
WDT	Watchdog timer	ウォッチドッグタイマ

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。

目次

1. 概要	7
1.1 目的	7
1.2 特徴	7
2. はじめに	8
3. プロジェクトの作成	9
3.1 はじめに	9
3.2 プロジェクトの作成	9
4. コード生成プラグインによるコード生成	10
4.1 はじめに	10
4.2 コード生成の有効化	11
4.3 コード生成ツアール	12
4.4 コード生成	13
4.4.1 共通/クロック発生回路	13
4.4.2 ポート機能	14
4.4.3 タイマ・アレイ・ユニット	16
4.4.4 ウォッチドッグ・タイマ	16
4.4.5 A/D コンバータ	17
4.4.6 シリアル・アレイ・ユニット	18
4.4.7 生成コード	20
5. プロジェクトの設定	21
5.1 プロジェクトへのカテゴリフォルダ追加	22
6. ユーザコードの統合	23
6.1 RSK チュートリアル用ファイルのコピー	23
6.2 CS+プロジェクトへファイルを追加	23
6.3 コード生成ファイルへのコード追加	24
6.3.1 r_cg_main.c コード追加	24
6.3.2 r_cg_adc_user.c コード追加	29
6.3.3 r_cg_sau.h コード追加	29
6.3.4 r_cg_sau.c コード追加	30
6.3.5 r_cg_sau_user.c コード追加	31
6.3.6 r_cg_userdefine.h コード追加	32
6.3.7 r_cg_tau_user.c コード追加	33
7. プロジェクトのデバッグ設定	36
7.1 チュートリアルコードの実行	37
8. 追加情報	38

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 RL78 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 RL78 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せん。

- CS+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 RL78/G1G 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。

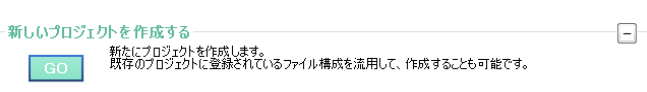
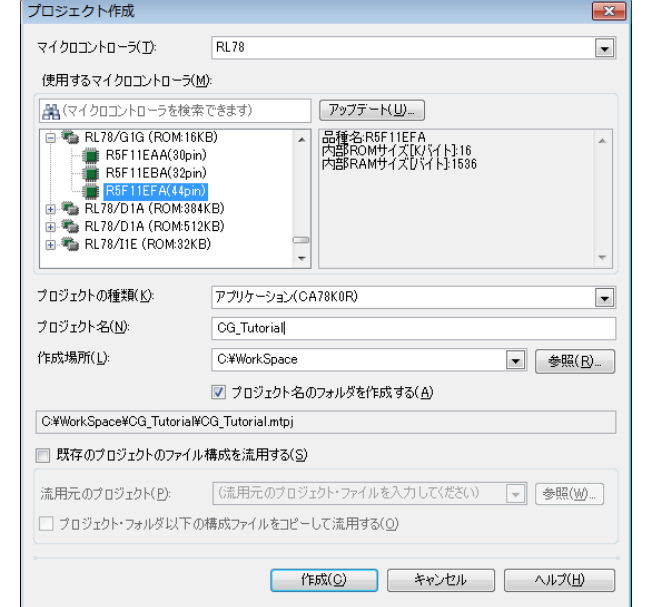
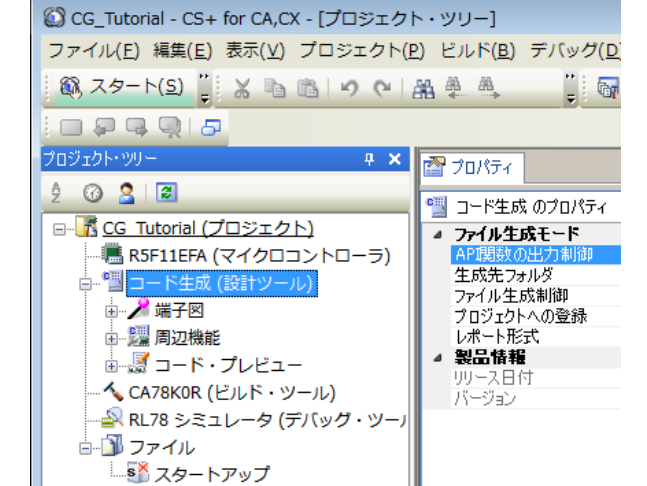
このプロジェクト作成の手順はマイクロコントローラ特有のプロジェクトを作成し、ソースをデバッグするのに必要です。

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 CA,CX (78K,RL78,V850)

Windows™ 8:  をクリックして [アプリ] ビューを表示 > “CS+ for CA,CX (78K,RL78,V850)” アイコン

<ul style="list-style-type: none"> 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。 	
<ul style="list-style-type: none"> 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'RL78'を選択してください。 マイクロコントローラ一覧で下にスクロールします。'RL78/G1G'の '+' を展開し R5F11EFA (44pin) を選択してください。 'プロジェクトの種類(K):'プルダウンから'アプリケーション(CA78K0R)'を選択してください。 'プロジェクト名(N):'と'作成場所(L)'を指定し、<作成>をクリックしてください。 注：右のスクリーンショットのプロジェクト名および作成場所は、本チュートリアル用のプロジェクト設定例です。 'フォルダが存在しません。作成しますか?'のダイアログが表示された場合、'はい(Y)'をクリックしてください。 	
<ul style="list-style-type: none"> 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・アレイ・ユニット(SAU)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Pmod LCDディスプレイに表示します。また、LED0-3でA/D変換回数をバイナリ点灯表示します。

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。

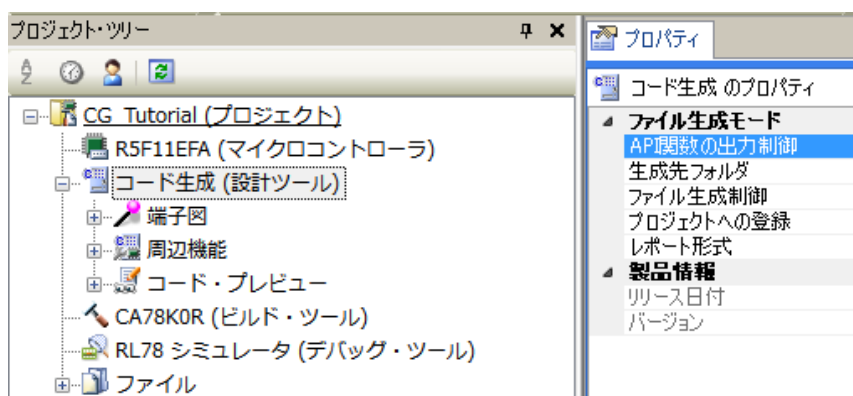
4.2 コード生成の有効化

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、「追加機能」タブの「コード生成プラグイン」のチェックボックスをチェックしてください。



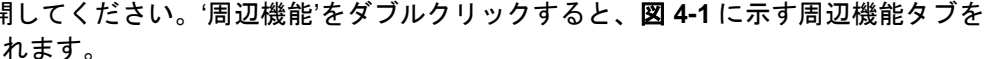
基本機能		追加機能
モジュール名	説明	
<input checked="" type="checkbox"/> IronPythonコンソール・プラグイン	IronPythonのコマンドとCS+拡張機能が使用できるコンソールです。	
<input checked="" type="checkbox"/> アップデート・マネージャ・プラグイン	CS+ アップデート・マネージャと連携するプラグインです。	
<input checked="" type="checkbox"/> エディタ・パネル	エディタ・パネルのプラグインです。	
<input type="checkbox"/> コード生成プラグイン	デバイスドライバを自動生成するプラグインです。(V850, 78K0, 78K0R, RL78/G12, G13, G14, G1A, I1A, L12, F12, F13, F14用)	
<input checked="" type="checkbox"/> コード生成/端子図プラグイン	デバイスドライバを自動生成および端子配置を表示するプラグインです。(RX, コード生成プラグインに記載のないRL78用)	

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。各操作の詳細につきましては、Application Leading Tool 共通操作編ユーザーズマニュアル(R20UT2663)を参照ください。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、コード生成プラグインのマニュアルとして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。‘周辺機能’をダブルクリックすると、 に示す周辺機能タブを含むメイン画面が表示されます。

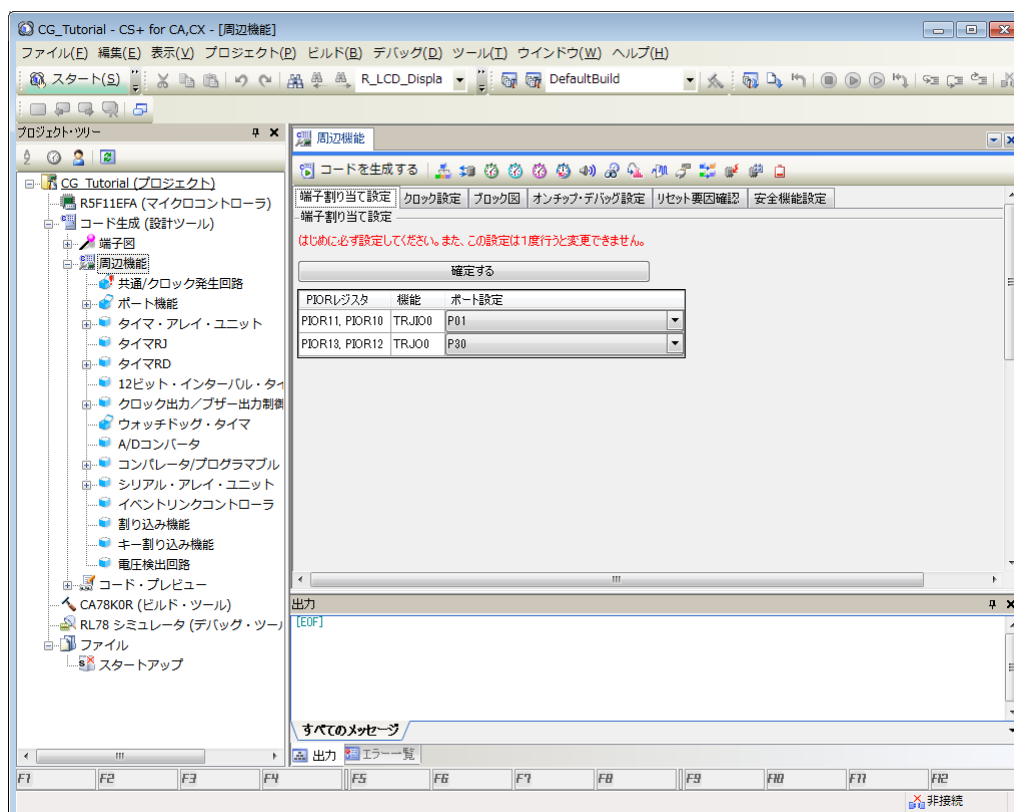


図 4-1: 初期画面

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、外部トリガによる A/D 変換モジュール、シリアル・アレイ・ユニット(SAU)、タイマ・アレイ・ユニット(TAU)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 共通/クロック発生回路

新規プロジェクトを作成した場合、“端子割り当て設定”タブが現れます(図 4-1 参照)。この設定は、RL78/G1G の周辺 I/O リダイレクト機能に該当し、兼用機能を割り当てるポートを切り替える機能です。他の周辺機能を設定する前に確定してください。ただし、一度確定すると“端子割り当て設定”は変更することができなくなりますのでご注意ください。変更したい場合は、再度新規プロジェクトを作成する必要があります。ここでは“端子割り当て設定”を変更せずに、“確定する”ボタンを押して次に進んでください。“確定する”ボタンをクリックすると、“確定する”ボタンはグレースアウトします。

“共通/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高速システム・クロック(fMX)に 20MHz 水晶発振子を使用します。“ブロック図”タブは、クロック発生回路図を示します。

プロジェクト・ツリーの“コード生成”->“周辺機能”->“共通/クロック発生回路”をダブルクリックし、“クロック設定”タブを選択してください。

図 4-2 を参照して設定してください。

端子割り当て設定	クロック設定	ブロック図	オンチップ・デバッグ設定	リセット要因確認	安全機能設定
動作モード設定					
<input type="radio"/> 高速メイン・モード 4.0(V) ≤ VDD ≤ 5.5(V) <input type="radio"/> 高速メイン・モード 3.6(V) ≤ VDD ≤ 5.5(V) <input checked="" type="radio"/> 高速メイン・モード 2.7(V) ≤ VDD ≤ 5.5(V) <input type="radio"/> 低速メイン・モード 2.7 (V) ≤ VDD ≤ 5.5 (V)					
メイン・システム・クロック(fMAIN)設定					
<input type="radio"/> 高速オンチップ・オシレータクロック(fIH) <input checked="" type="radio"/> 高速システム・クロック(fMX)					
高速オンチップ・オシレータクロック設定					
<input type="checkbox"/> 動作					
周波数 48 (fHOCO=48, fIH=24) (MHz)					
高速システム・クロック設定					
<input checked="" type="checkbox"/> 動作					
<input checked="" type="radio"/> X1発振(fX)					
<input type="radio"/> 外部クロック入力(fEX)					
周波数 20 (MHz)					
発振安定時間 2 ⁻¹⁸ /fX 13107.2 (μs)					
低速内蔵発振クロック(fIL)設定					
周波数 15 (kHz)					
インターバル・タイマ動作クロック/タイマRJカウンタ・ソース動作クロック設定					
インターバル・タイマ動作クロック/タイマRJカウンタ・ソース fIL 15 (kHz)					
CPUと周辺クロック設定					
CPUと周辺クロック(fCLK) fMX 20000 (kHz)					

図 4-2: クロック設定

4.4.2 ポート機能

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。ポート設定の概要を表 4-1 に示します。

表 4-1: RSK 用ポート設定

RSK 機能	Port	設定
SW1	P7.0	入力
SW2	P12.4	入力
SW3	P12.3	入力
LED0	P4.1	出力
LED1	P6.3	出力
LED2	P7.2	出力
LED3	P7.3	出力
PMOD	P6.1	出力
PMOD	P6.2	出力
PMOD	P7.1	出力

※例：P7.0 は、Port7 のビット 0 を示します。

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。図 4-3 が開きます。

Port4、Port6、Port7 と Port12 を設定します。“Port4”タブ、“Port6”タブ、“Port7”と“Port12”タブを各々設定してください。各ポートの設定は、図 4-4～図 4-6 を参照して設定してください。それ以外のポートは、デフォルト設定の状態にしておいてください。

ユーザ LED は、初期設定で消灯状態にします。“1 を出力”チェックボックスをクリックしてチェックしてください。

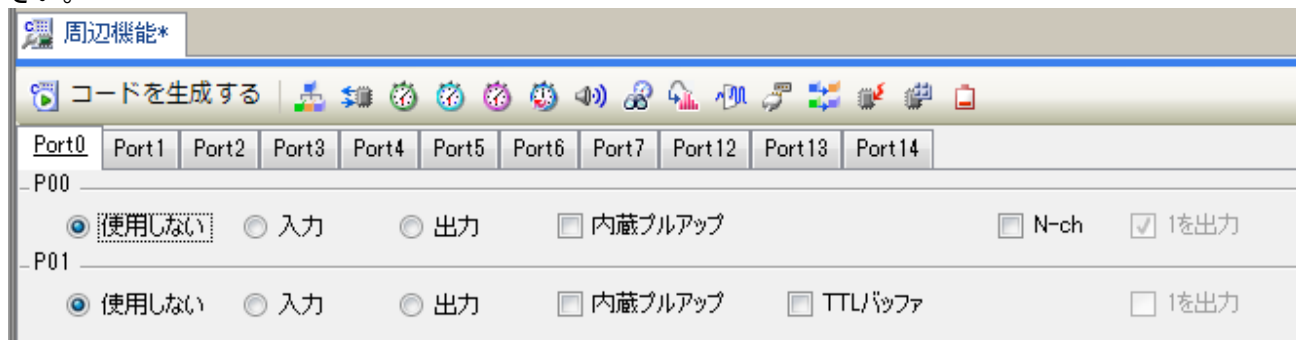


図 4-3: ポート設定初期画面

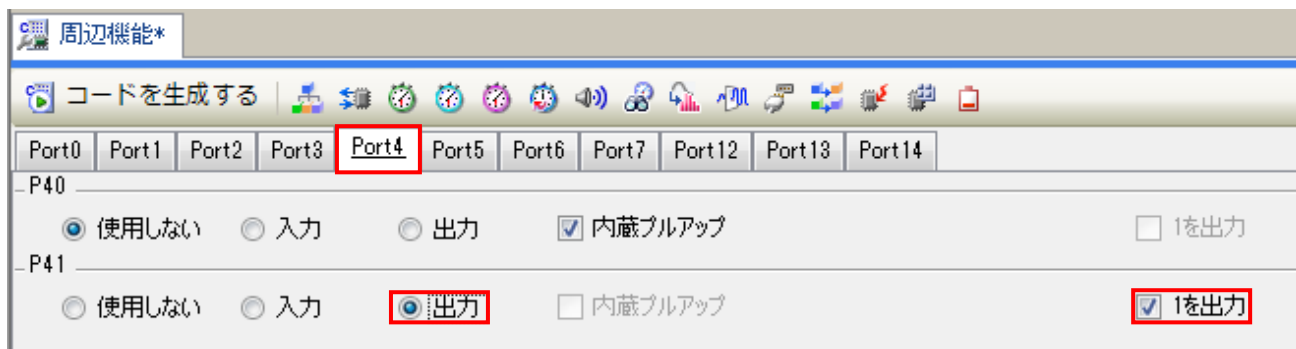


図 4-4: ポート 4 設定

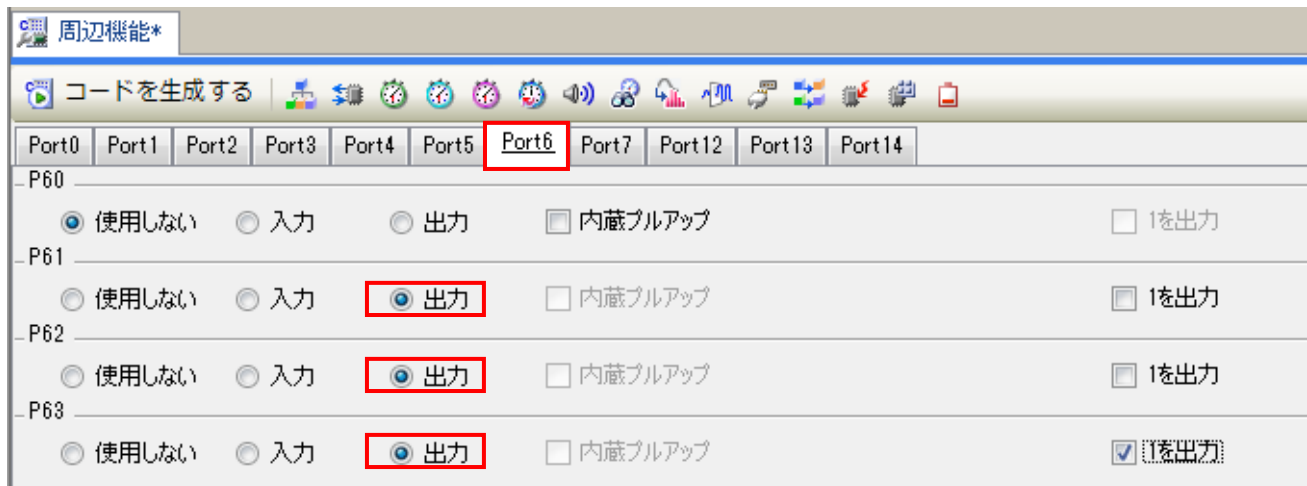


図 4-5: ポート 6 設定

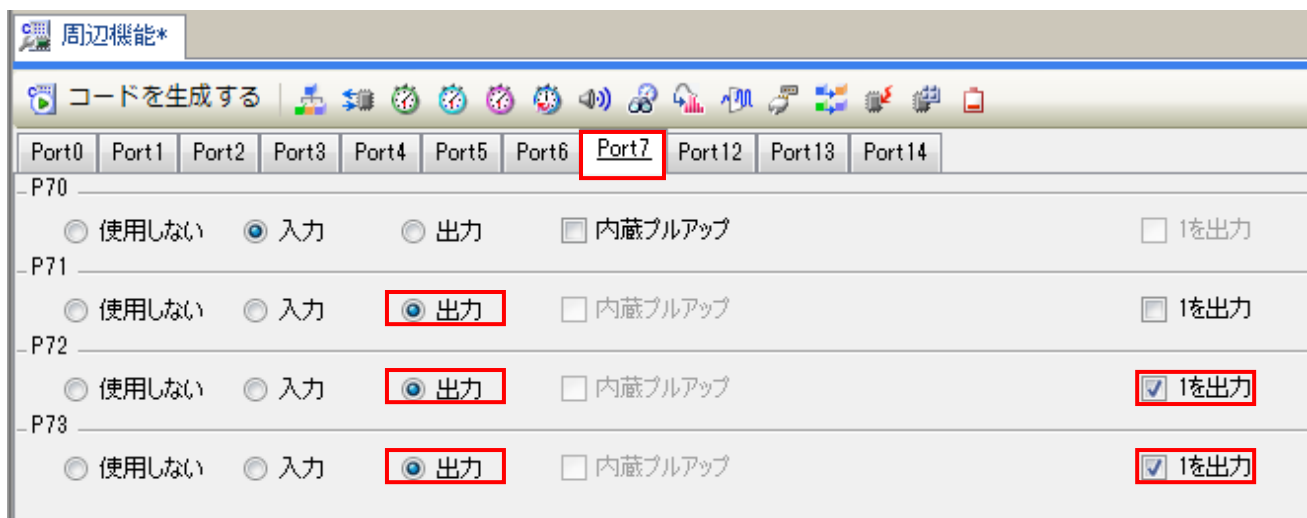


図 4-6: ポート 7 設定

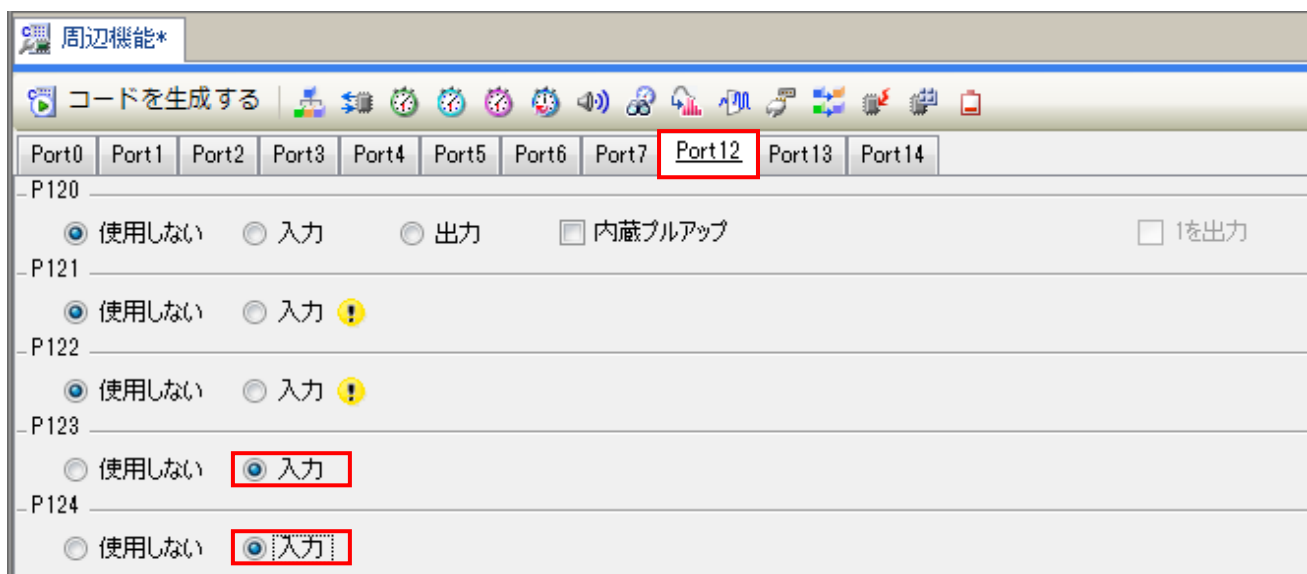


図 4-7: ポート 12 設定

注：⚠️アイコンは無視して問題ありません。

4.4.3 タイマ・アレイ・ユニット

TAUのチャンネル0とチャンネル2を1msインターバル・タイマ設定します。

プロジェクト・ツリーの“コード生成”->“周辺機能”->“タイマ・アレイ・ユニット”をダブルクリックしてください。図 4-8 を参照して設定してください。

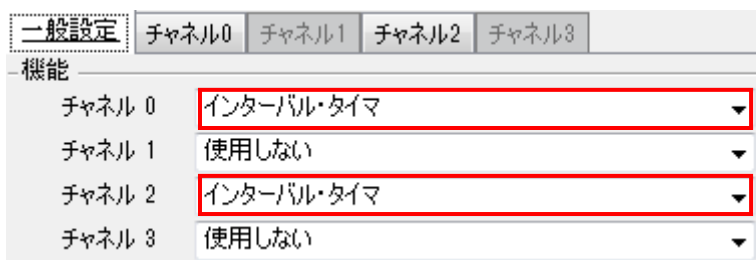


図 4-8: TAU チャンネル 0 とチャンネル 2 設定

“チャンネル0”タブをクリックしてください。図 4-9 を参照してチャンネル 0、2 の設定をしてください。

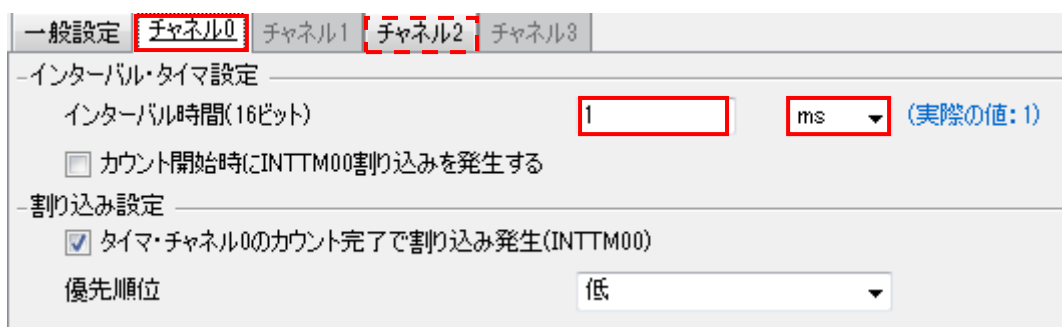


図 4-9: TAU チャンネル 0 設定

“チャンネル2”タブをクリックして、“チャンネル0”タブと同等の設定を行ってください。

4.4.4 ウォッチドッグ・タイマ

このプロジェクトでは使用しませんが、ウォッチドッグ・タイマはデフォルト設定の状態です。“動作する”が選択されています。

プロジェクト・ツリーの“コード生成”->“周辺機能”->“ウォッチドッグ・タイマ”をダブルクリックし、“ウォッチドッグ・タイマ動作設定”を“使用しない”に設定してください。

4.4.5 A/D コンバータ

RSKでは、ANI0端子とポテンショメータRV1が接続されています。ANI0端子に入力された電圧を10ビット分解能、ワンショット・セレクト・モードでA/D変換するように設定します。

プロジェクト・ツリーの“コード生成”->“周辺機能”->“A/Dコンバータ”をダブルクリックし、**図 4-10**を参照して設定してください。

図 4-10: A/D コンバータ設定

4.4.6 シリアル・アレイ・ユニット

SAU は Pmod LCD(CSI00)と PC(UART1)の通信に使用されます。

UART1 の TxD1 と RxD1 は事前に USB シリアル変換インタフェースに設定された RL78/G1C に接続されています。

プロジェクト・ツリーの“コード生成”->“周辺機能”->“シリアル・アレイ・ユニット”をダブルクリックし、SAU の各チャンネルを、**図 4-11** を参照して設定してください。

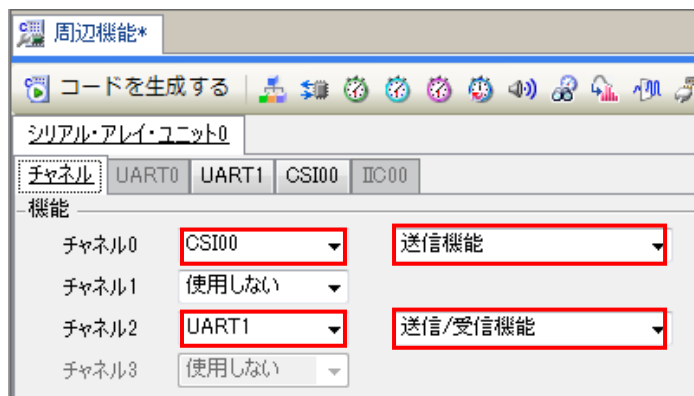


図 4-11: シリアル・アレイ・ユニットチャンネル設定

“CSI00”タブをクリックして、**図 4-12** の通りに設定してください。



図 4-12: CSI00 設定

“UART1”タブをクリックして、**図 4-13** と **図 4-14** を参照して設定してください。

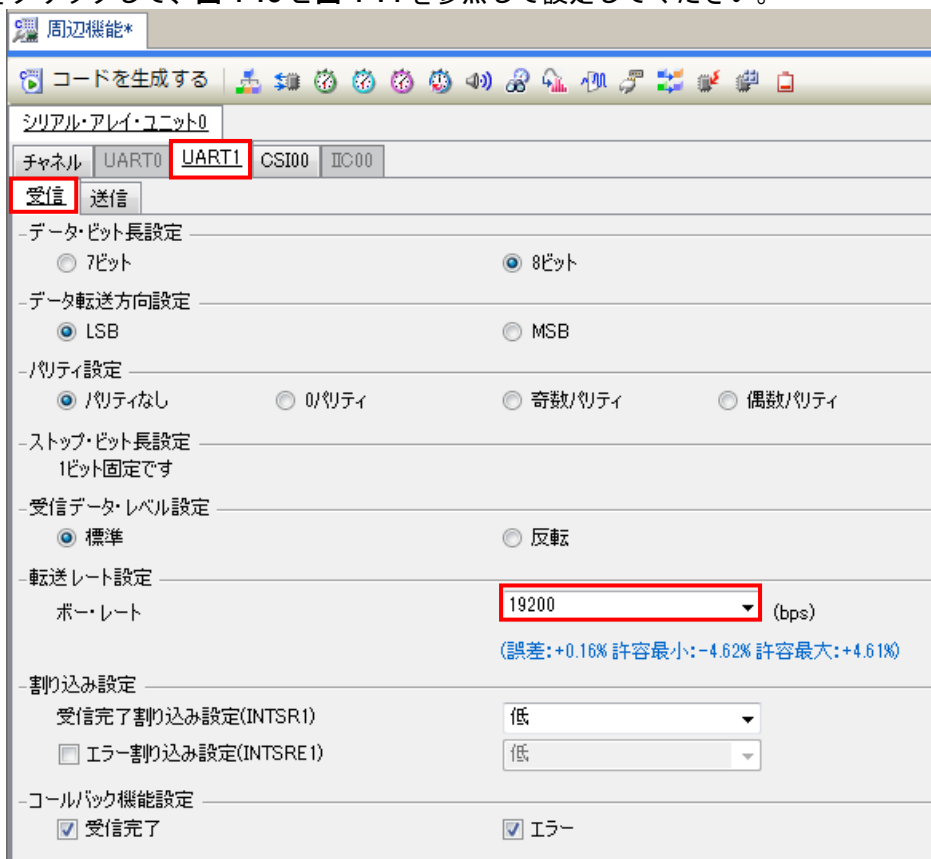


図 4-13: UART1 受信タブ設定

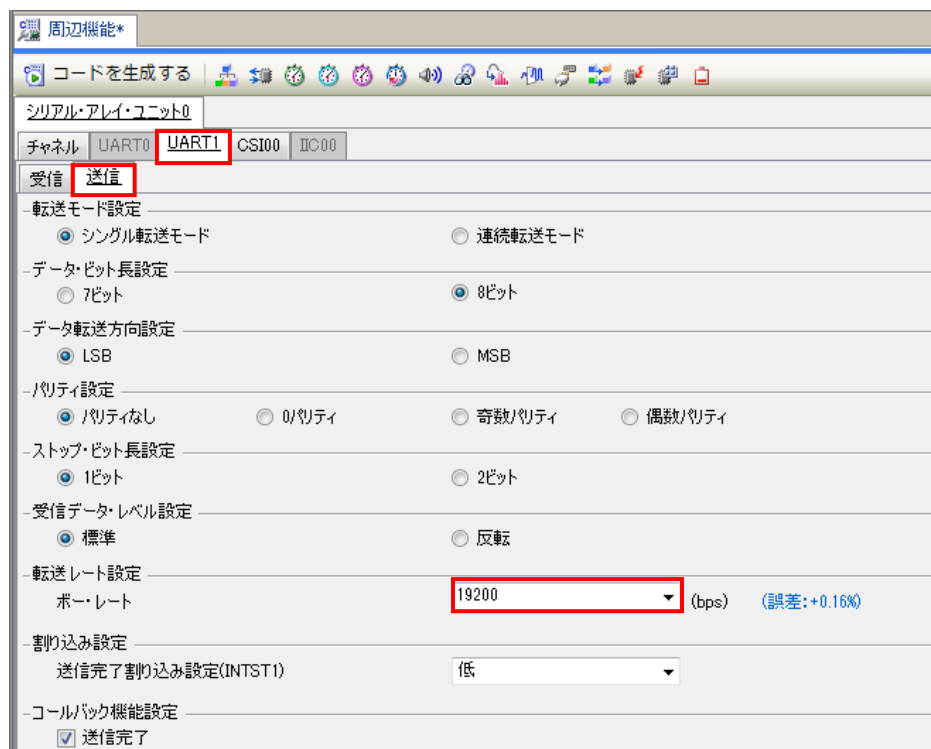


図 4-14: UART1 送信タブ設定

これでコード生成による周辺機能の設定は全て完了しました。メニューバーの“ファイル” -> “すべてを保存(L)”を選択し、プロジェクトを保存してください。次のセクションに進んでください。

4.4.7 生成コード

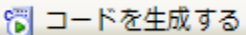
周辺機能の設定は全て完了しました。次に、<コードを生成する>  ボタンをクリックして、コードを生成してください。

図 4-15 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。

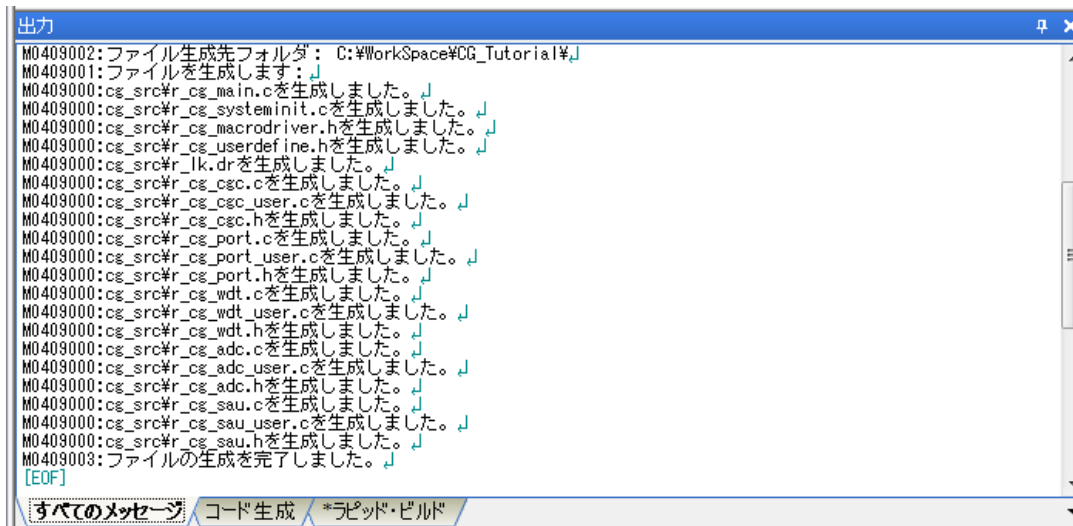


図 4-15: コード生成画面

図 4-16 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。

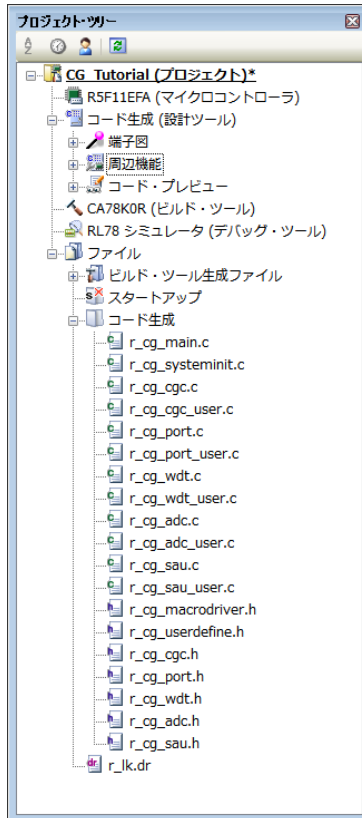


図 4-16: プロジェクト・ツリー中のコード生成ファイル

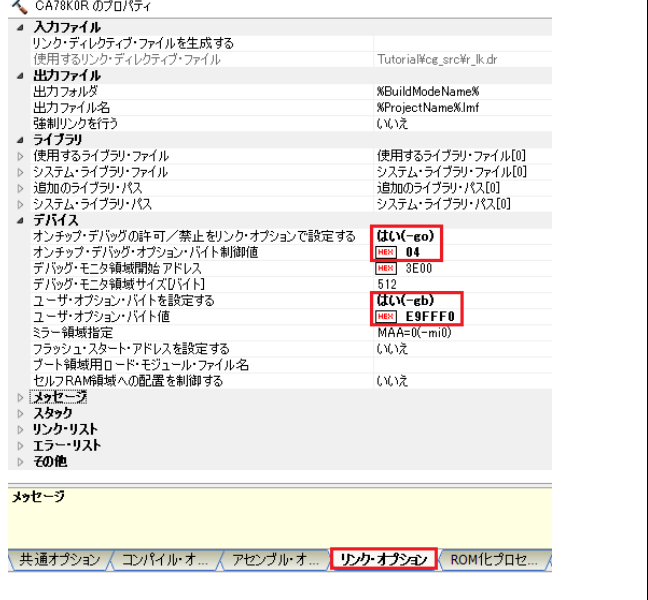
5. プロジェクトの設定

この章では、RSK を動作させるためのプロジェクト設定を行います。

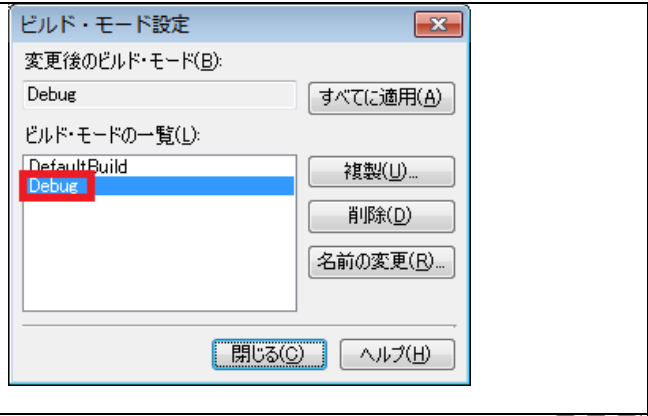
- プロジェクト・ツリーの"CA780KR(ビルド・ツール)"をダブルクリックして、プロパティを表示してください。
- 新規作成されたプロジェクトは、ビルド・モードに"DefaultBuild"が設定されています。また、標準で最適化(レベル 2)が行われます。



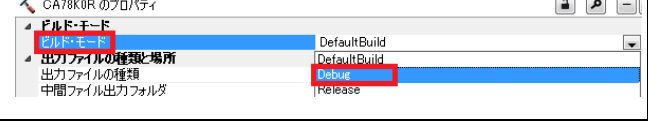
- 下のタブ"リンク・オプション"をクリックしてください。リンク・オプションタブ内の、"デバイス"項目を設定してください。
- オプション・デバッグの許可/禁止をリンク・オプションで設定する：はい(-go)
- オンチップ・デバッグ・オプション・バイト制御値：04
- ユーザ・オプション・バイトを設定する：はい(-gb)
- ユーザ・オプション・バイト値：E9FFF0

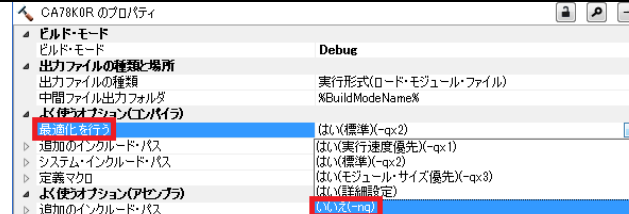


- ### 共通オプション-ビルド・モード(Debug)設定
- メニューバーから"ビルド(B)"を選択し、さらに"ビルド・モード設定(M)"を選択してください。
 - ビルド・モード設定画面で、"複製(U)"をクリックしてください。
 - "文字列(S)"に"Debug"と入力して"OK"をクリックしてください。
 - ビルド・モードの一覧(L)に"Debug"が追加されていることを確認してください。
 - 閉じる(C)をクリックしてください。


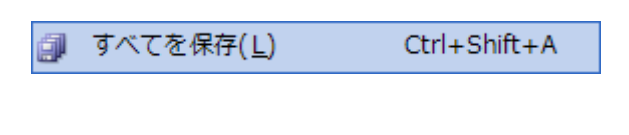


- CA780KR プロパティ画面の"共通オプション"タブのビルド・モードから"Debug"を選択してください。

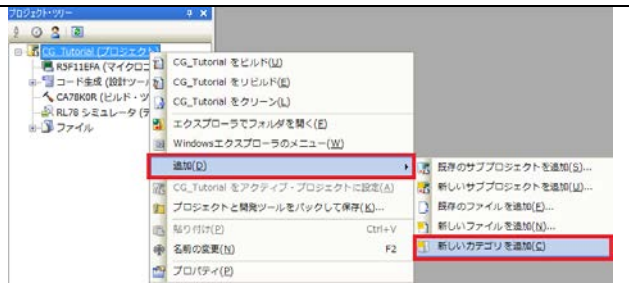
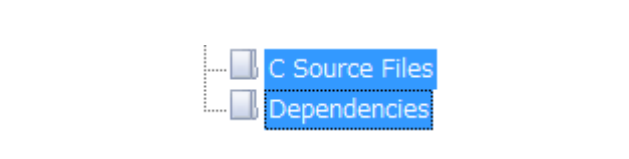


<ul style="list-style-type: none"> 続けて、“最適化を行う”のプルダウンメニューから“いいえ(-nq)”を選択してください。これにより、ビルド・モード“Debug”では、コードの最適化が行われなくなります。チュートリアルでは、このビルド・モードを使用します。 	
---	--

共通オプション-ビルド・モード(Release)設定

<ul style="list-style-type: none"> 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(Default Build、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“Release”を追加してください。 “Release”では、“最適化を行う”は初期設定の状態にしてください。 右図上段、画面下の“コンパイル・オプション”タブの“デバッグ情報を生成する”のプルダウンメニューから“いいえ(-ng)”を選択してください。 右図中段、画面下の“アセンブル・オプション”タブの“デバッグ情報を生成する”のプルダウンメニューから“いいえ(-ng, -nga)”を選択してください。 右図下段、画面下の“リンク・オプション”タブの“デバッグ情報を生成する”のプルダウンメニューから“いいえ(-ng)”を選択してください。 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。 	
<ul style="list-style-type: none"> 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 プロジェクトへのカテゴリフォルダ追加

<ul style="list-style-type: none"> (1)カテゴリを作成します。 プロジェクト・ツリーのCG_Tutorialプロジェクトを右クリックして、“追加”を選択、さらに“新しいカテゴリを追加”を選択してください。 	
<ul style="list-style-type: none"> (2)新しいカテゴリフォルダの名前を“C Source Files”に変更してください。 上記(1)、(2)と同様の方法で、“Dependencies”カテゴリフォルダも作成してください。 	

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 _xxxx_. 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。

```
r_ascii.c,
r_ascii.h,
r_lcd.c,
r_lcd.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 CS+プロジェクトへファイルを追加

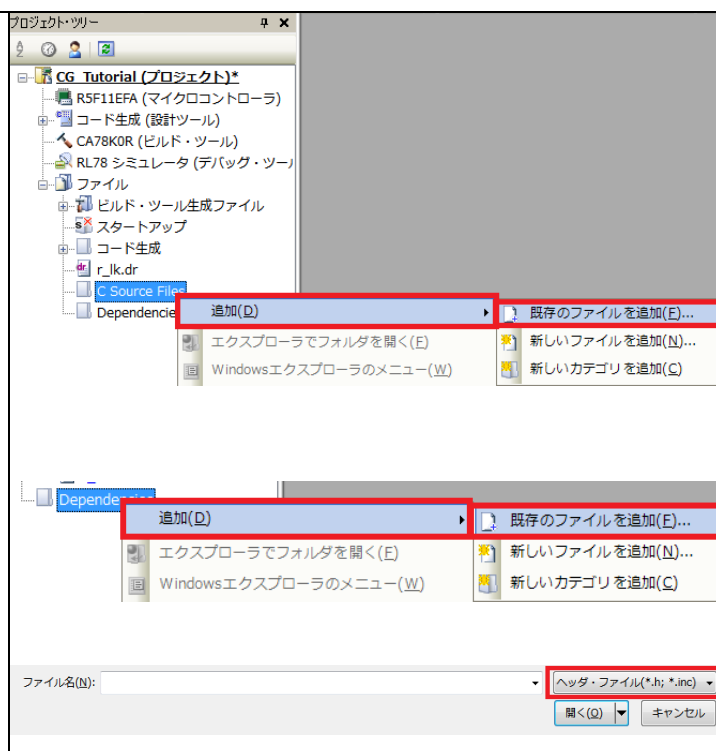
- 'C Source Files'カテゴリフォルダを右クリックして、'追加' -> '既存のファイルを追加'を選択してください。

- 以下のファイルを追加してください。

```
r_ascii.c
r_lcd.c
```

- 同様に、'Dependencies'カテゴリフォルダに以下のファイルを追加してください。エクスプローラで“ヘッダ・ファイル(*.h;*.inc)”をプルダウンメニューから選択すると表示されます。

```
r_ascii.h
r_lcd.h
```



6.3 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。

“Start ...”と“End ...”コメント文の間にカスタムコードを加える必要があるため、挿入位置にご注意ください。

6.3.1 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、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 <string.h>
#include "r_lcd.h"
/* End user code. Do not edit comment generated here */
```

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 */

/* Converts count to binary and displays on LEDs 0 to 3 */
static void led_display_count (const uint8_t count);

/* Read value from ADC. */
static uint16_t get_adc (void);

/* Read state of switches */
static void read_switch (volatile switch_t g_swn, uint8_t port_value);

/* Write to UART1 */
static void text_write (uint8_t * const msg_string);

/* Conversion to facilitate outputting to LCD module. */
static void uint16_to_string (uint8_t * const output_string, uint8_t pos, const uint16_t
input_number);

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

/* LCD module string buffer */
static uint8_t lcd_buf[10];

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

/* Character received from PC terminal */
extern volatile uint8_t g_rx_char;

/* Commands to clear terminal window and set cursor to start of window */
const uint8_t g_cmd_clr_scr[] =
{ 27, '[', '2', 'J', 0 };
const uint8_t g_cmd_cur_home[] =
{ 27, '[', 'H', 0 };

/* Variable to store the A/D conversion count for user display */
uint8_t adc_count = 0;
uint16_t adc_result;
uint8_t initial_adc_meas = TRUE;
```



```

/* Switch value (state of input port) */
uint8_t switch_value;

/* ADC rx complete interrupt flag */
extern volatile uint8_t g_adc_rx_int;

/* UART1 serial transmission in progress */
extern volatile uint8_t g_uart1_tx_busy;

/* Debounce state */
extern volatile uint8_t g_debounce_ongoing;

/* Switches */
extern volatile switch_t g_sw3;

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

```

main関数の以下に該当する箇所を書き換えてください。

```

/* Start user code. Do not edit comment generated here */
while (1U)
{
    ;
}
/* End user code. Do not edit comment generated here */

```

以下のコードに書き換えてください。

```

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

/* Initialise the LCD display */
init_lcd();

/* Display test information */
display_lcd(0, (uint8_t const *) "Renesas");
display_lcd(1, (uint8_t const *) "RL78/G1G");
display_lcd(3, (uint8_t const *) "Tutorial sample");
display_lcd(4, (uint8_t const *) "Connect USB to PC");
display_lcd(5, (uint8_t const *) "Serial configuration:");
display_lcd(6, (uint8_t const *) "Baud Rate 19200");
display_lcd(7, (uint8_t const *) "Data Bits 8");
display_lcd(8, (uint8_t const *) "Stop Bits 1");
display_lcd(9, (uint8_t const *) "Parity None");
display_lcd(10, (uint8_t const *) "Flow None");

/* Set up UART1 receive buffer and callback function */
R_UART1_Receive((uint8_t * const) &g_rx_char, 1);

/* Enable UART1 operations */
R_UART1_Start();

while (1U)
{
    /* Read SW3. */
    switch_value = SW3_VALUE;
    read_switch(g_sw3, switch_value);

    /* If a new press of SW3 then request a new A/D conversion. */
    if (TRUE == g_sw3.switch_new_press)
    {
        g_sw3.switch_new_press = FALSE;

        /* set the flag indicating a user requested A/D conversion is required */
        g_adc_trigger = TRUE;
    }

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

        /* Display the result on the LCD */
        uint16_to_string lcd_buf, (uint8_t) 0, adc_result);
        display_lcd(12, (uint8_t const *) lcd_buf);
    }
}

```

```

        /* Increment the adc_count and display using the LEDs if not the initial reading. */

        if (FALSE == initial_adc_meas)
        {
            if (16 == (++adc_count))
            {
                adc_count = 0;
            }
        }
        led_display_count(adc_count);

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

        /* Reset the flag */
        g_adc_trigger = FALSE;

        initial_adc_meas = FALSE;
    }
}
/* 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 : read_switch
* Description   : If the switch state has changed then trigger the debounce timer, which will set
the debounced switch
*               state to pressed or released as appropriate. The calling program must set the new
press or new
*               released state to false once processed.
* Argument      : none
* Return value  : none
*****
*****/
static void read_switch (volatile switch_t g_sw, uint8_t port_value)
{
    /* Start TAU channel 0 timer (debounce timer) if switch state change detected. */
    if (((SWITCH_PRESSED == port_value) && (SWITCH_RELEASED == g_sw.current_switch_state))
        || ((SWITCH_RELEASED == port_value) && (SWITCH_PRESSED == g_sw.current_switch_state)))
    {
        /* TAU channel 0 only needs to be started if it has already been stopped */
        if (FALSE == g_debounce_ongoing)
        {
            g_debounce_ongoing = TRUE;

            /* Start TAU channel 0. which is configured as a periodic timer to aid switch debouncing.
*/
            R_TAU0_Channel0_Start();
        }
    }
}
/*****
*****
* End of function read_switch
*****
*****/

/*****
*****
* Function Name : get_adc
* Description   : Reads the ADC result.
* Argument      : none
* Return value  : adc_result - Value of ADC conversion
*****
*****/
static uint16_t get_adc (void)
{
    uint16_t adc_result;

```

```

/* Enable comparator operation */
R_ADC_Set_OperationOn();

/* Start a conversion */
R_ADC_Start();

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

R_ADC_Get_Result(&adc_result);

/* stops comparator operation */
R_ADC_Set_OperationOff();

/* stops the AD converter */
R_ADC_Stop();

return adc_result;
}
/*****
*****
* End of function get_adc
*****
*****/

/*****
*****
* Function Name : uart_display_adc
* Description   : Converts adc result to a string and sends it to UART1.
* Argument      : adc_count - Number of ADC conversions (modulo 16)
*                adc result - Value of ADC conversion
* Return value  : none
*****
*****/
static void uart_display_adc (uint8_t adc_count, uint16_t adc_result)
{
    uint8_t str1[50];

    /* Clear terminal window and set cursor to start of window */
    text_write((uint8_t *) &g_cmd_clr_scr);
    text_write((uint8_t *) &g_cmd_cur_home);

    strcpy((char *) str1, "ADC value =      \r\n");
    uint16_to_string(str1, (uint8_t) 12, (uint16_t) adc_result);
    text_write(str1);

    strcpy((char *) str1, "Number of ADC conversions (modulo 16) =      \r\n");
    uint16_to_string(str1, (uint8_t) 40, (uint16_t) adc_count);
    text_write(str1);
}
/*****
*****
* End of function uart_display_adc
*****
*****/

/*****
*****
* Function name : text_write
* Description    : Transmits null-terminated string.
* Argument       : msg_string - null terminated string
* Argument       : None
*****
*****/
static void text_write (uint8_t * const msg_string)
{
    uint16_t i;

    for (i = 0; msg_string[i]; i++)

```

```

    {
        /* Send one byte and set UART transmit busy flag */

        R_UART1_Send(&msg_string[i], 1);
        g_uart1_tx_busy = TRUE;

        /* Wait until UART transfer is complete*/
        while (TRUE == g_uart1_tx_busy)
        {
            /* Wait */
        }
    }
}
/*****
*****
* End of Function text_write
*****
*****/

/*****
*****
* Function Name : led_display_count
* Description   : Converts count to binary and displays on LEDs 0 to 3
* Argument      : count - Number of ADC conversions (modulo 16)
* Return value  : none
*****/
static void led_display_count (const uint8_t count)
{
    /* Set LEDs according to lower nibble of count parameter */
    LED0 = (count & 0x01) ? LED_ON : LED_OFF;
    LED1 = (count & 0x02) ? LED_ON : LED_OFF;
    LED2 = (count & 0x04) ? LED_ON : LED_OFF;
    LED3 = (count & 0x08) ? LED_ON : LED_OFF;
}
/*****
*****
* End of function led_display_count
*****/

/*****
*****
* Function Name: uint16_to_string
* Description   : Function converts a 16 bit integer into a character string, inserts it into the
array via the pointer
*               passed at execution.
* Argument      : output_string - Pointer to char array that will hold character string.
*               pos - uint8_t number, element number to begin inserting the character
string from (offset).
* Return value  : none
* Note          : No input validation is used, so output data can overflow the array passed.
*****/
static void uint16_to_string (uint8_t * const output_string, uint8_t pos, const uint16_t
input_number)
{
    /* Declare 16bit mask variable */
    uint16_t mask = 0xF000;

    /* Declare temporary character storage variable, and bit_shift variable */
    uint8_t a = 0x00;
    uint8_t bit_shift = 12u;

    /* Loop through until each hex digit is converted to an ASCII character */
    while (bit_shift < 30u)
    {
        /* Mask and shift the hex digit, and store in temporary variable, a */
        a = (uint8_t) ((input_number & mask) >> bit_shift);

        /* Convert the hex digit into an ASCII character, and store in output
string */
    }
}

```


6.3.4 r_cg_sau.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_sau.c’をダブルクリックして開いてください。
global用ユーザコードエリアコメント文の間に以下のように追加してください。

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

```
volatile uint8_t g_csi0_tx_in_process = FALSE;
```

```
/* 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: send_csi0
* Description : This function sends CSI0 data to slave device. Adds flagging around R_CSI00_Send
* Arguments : tx_buf -
*             transfer buffer pointer (Not used when transmit data handled by DTC)
*             tx_num -
*             buffer size
* Return Value : status -
*               MD_OK or MD_ARGERROR
*****
*****/

```

```
void send_csi0 (uint8_t * const tx_buf, uint16_t const tx_num)
```

```
{
    g_csi0_tx_in_process = TRUE;
    R_CSI00_Send(tx_buf, tx_num);
}
```

```

*****
* End of function send_csi0
*****/

```

```

*****
* Function Name : csi0_tx_is_busy
* Description : reports if CSI00 is transmitting
* Argument : none
* Return value : None
*****/

```

```
uint8_t csi0_tx_is_busy (void)
{
    return (g_csi0_tx_in_process);
}
```

```

*****
* End of function csi0_tx_is_busy
*****/

```

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

6.3.5 r_cg_sau_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_sau_user.c'をダブルクリックして開いてください。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ように追加してください。

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

```
extern volatile uint8_t g_csi0_tx_in_process;
extern volatile uint8_t g_adc_trigger;
```

```
/* UART1 serial transmission in progress */
volatile uint8_t g_uart1_tx_busy = FALSE;
```

```
/* Character received from PC terminal */
volatile uint8_t g_rx_char;
```

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

r_uart1_callback_receiveend関数に以下のカスタムコードを追加してください。

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

```
/* Check the character received from the PC */
if (('c' == g_rx_char) || ('C' == g_rx_char))
{
    g_adc_trigger = TRUE;
}
```

```
/* Set up UART1 receive buffer and callback function again */
R_UART1_Receive((uint8_t * const) &g_rx_char, 1);
```

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

r_uart1_callback_sendend関数に以下のカスタムコードを追加してください。

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

```
/* UART1 serial transmission finished */
g_uart1_tx_busy = FALSE;
```

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

r_csi00_callback_sendend関数に以下のカスタムコードを追加してください。

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

```
g_csi0_tx_in_process = FALSE;
```

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

6.3.6 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。
function 用ユーザコードエリアコメント文の間に以下のように追加してください。

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

```
/* Switch port settings */  
#define SW1          (1)  
#define SW2          (2)  
#define SW3          (3)  
#define SW1_VALUE   (P7.0)  
#define SW2_VALUE   (P12.4)  
#define SW3_VALUE   (P12.3)  
#define SWITCH_PRESSED (0)  
#define SWITCH_RELEASED (1)
```

```
/* Switch debounce settings */  
#define PRESSED_DEBOUNCE_COUNT (10)  
#define RELEASED_DEBOUNCE_COUNT (20)
```

```
/* LED port settings */  
#define LED0         (P4.1)  
#define LED1         (P6.3)  
#define LED2         (P7.2)  
#define LED3         (P7.3)
```

```
/* LED lights. */  
#define LED_ON       (0)  
#define LED_OFF      (1)
```

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

```
/* Switches */  
typedef struct  
{  
    uint8_t current_switch_state;  
    uint8_t switch_new_press;  
    uint8_t switch_new_release;  
    uint8_t debounce_counter;  
} switch_t;
```

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


6.3.7 r_cg_tau_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_tau_user.c'をダブルクリックして開いてください。global用ユーザコードエリアコメント文の間、function用ユーザコードエリアコメント文の間に以下のように追加してください。

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

/* TAU0 channel2 interrupt count */
volatile uint16_t g_tau_ch2_cnt = 0;

/* Switches */
volatile switch_t g_sw1 =
{ SWITCH_RELEASED, FALSE, FALSE, 0 };
volatile switch_t g_sw2 =
{ SWITCH_RELEASED, FALSE, FALSE, 0 };
volatile switch_t g_sw3 =
{ SWITCH_RELEASED, FALSE, FALSE, 0 };

/* Debounce state */
volatile uint8_t g_debounce_ongoing = FALSE;

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

r_tau0_channel0_interrupt関数に以下のカスタムコードを追加してください。

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

/* This ISR will debounce switches SW1, SW2 and SW3. The debounce algorithm will check that the
switch state (either
* pressed or released is stable over a defined period, which can be modified at compile time.
The debounce time
* for pressing and releasing can be independently configured. Once a switch is pressed or
released then the state
* of the switch is sampled over the predefined time; a counter is incremented every time the
sampled signal is the
* same as the previous state. On reaching the end of the debounce period, if it has been stable
for the whole
* period then the change in switch state is deemed to be valid (debounced) and the new state is
updated. If the
* sampled switch state is not the same as the previous one then the counter is reset and
counting recommences.
* This timer will start when any of the switches have been pressed or released and will stop
only after all
* switches have been debounced. */

/* Check the last current stable state of SW1. */
if (SWITCH_RELEASED == g_sw1.current_switch_state)
{
    /* Switch is in the RELEASED state so it must have been pressed. Read switch input value,
clear debounce counter
* if switch has bounced back to the release position (open), else increment debounce counter
and confirm new
* state and new switch pressed once debounce count is reached. */
    if (SWITCH_RELEASED == SW1_VALUE)
    {
        g_sw1.debounce_counter = 0;
    }
    else
    {
        g_sw1.debounce_counter++;

        /* If at the end of the debounce period, then update the current state and indicate that
a new press has
* been detected. */
        if (PRESSED_DEBOUNCE_COUNT == g_sw1.debounce_counter)
        {
            g_sw1.current_switch_state = SWITCH_PRESSED;
            g_sw1.switch_new_press = TRUE;
        }
    }
}
else
{
```

```

        if (SWITCH_PRESSED == g_sw1.current_switch_state)
        {
            /* Switch is in the PRESSED state so it must have been released. Read switch input value,
            clear debounce
            * counter if switch has bounced back to the pressed position (closed), else increment
            debounce counter and
            * confirm new state and new switch released once debounce count is reached. */
            if (SWITCH_PRESSED == SW1_VALUE)
            {
                g_sw1.debounce_counter = 0;
            }
            else
            {
                g_sw1.debounce_counter++;

                /* If at the end of the debounce period, then update the current state and indicate
            that a new release
            * has been detected. */
                if (RELEASED_DEBOUNCE_COUNT == g_sw1.debounce_counter)
                {
                    g_sw1.current_switch_state = SWITCH_RELEASED;
                    g_sw1.switch_new_release = TRUE;
                }
            }
        }

        /* Check the last current stable state of SW2. */
        if (SWITCH_RELEASED == g_sw2.current_switch_state)
        {
            /* Switch is in the RELEASED state so it must have been pressed. Read switch input value,
            clear debounce counter
            * if switch has bounced back to the release position (open), else increment debounce counter
            and confirm new
            * state and new switch pressed once debounce count is reached. */
            if (SWITCH_RELEASED == SW2_VALUE)
            {
                g_sw2.debounce_counter = 0;
            }
            else
            {
                g_sw2.debounce_counter++;

                /* If at the end of the debounce period, then update the current state and indicate that
            a new press has
            * been detected. */
                if (PRESSED_DEBOUNCE_COUNT == g_sw2.debounce_counter)
                {
                    g_sw2.current_switch_state = SWITCH_PRESSED;
                    g_sw2.switch_new_press = TRUE;
                }
            }
        }
        else
        {
            if (SWITCH_PRESSED == g_sw2.current_switch_state)
            {
                /* Switch is in the PRESSED state so it must have been released. Read switch input value,
            clear debounce
            * counter if switch has bounced back to the pressed position (closed), else increment
            debounce counter and
            * confirm new state and new switch released once debounce count is reached. */
                if (SWITCH_PRESSED == SW2_VALUE)
                {
                    g_sw2.debounce_counter = 0;
                }
                else
                {
                    g_sw2.debounce_counter++;

                    /* If at the end of the debounce period, then update the current state and indicate
            that a new release
            * has been detected. */
                    if (RELEASED_DEBOUNCE_COUNT == g_sw2.debounce_counter)
                    {
                        g_sw2.current_switch_state = SWITCH_RELEASED;
                        g_sw2.switch_new_release = TRUE;
                    }
                }
            }
        }
    }
}

```

```

    }
}

/* Check the last current stable state of SW3. */
if (SWITCH_RELEASED == g_sw3.current_switch_state)
{
    /* Switch is in the RELEASED state so it must have been pressed. Read switch input value,
clear debounce counter
    * if switch has bounced back to the release position (open), else increment debounce counter
and confirm new
    * state and new switch pressed once debounce count is reached. */
    if (SWITCH_RELEASED == SW3_VALUE)
    {
        g_sw3.debounce_counter = 0;
    }
    else
    {
        g_sw3.debounce_counter++;

        /* If at the end of the debounce period, then update the current state and indicate that
a new press has
        * been detected. */
        if (PRESSED_DEBOUNCE_COUNT == g_sw3.debounce_counter)
        {
            g_sw3.current_switch_state = SWITCH_PRESSED;
            g_sw3.switch_new_press = TRUE;
        }
    }
}
else
{
    if (SWITCH_PRESSED == g_sw3.current_switch_state)
    {
        /* Switch is in the PRESSED state so it must have been released. Read switch input value,
clear debounce
        * counter if switch has bounced back to the pressed position (closed), else increment
debounce counter and
        * confirm new state and new switch released once debounce count is reached. */
        if (SWITCH_PRESSED == SW3_VALUE)
        {
            g_sw3.debounce_counter = 0;
        }
        else
        {
            g_sw3.debounce_counter++;

            /* If at the end of the debounce period, then update the current state and indicate
that a new release
            * has been detected. */
            if (RELEASED_DEBOUNCE_COUNT == g_sw3.debounce_counter)
            {
                g_sw3.current_switch_state = SWITCH_RELEASED;
                g_sw3.switch_new_release = TRUE;
            }
        }
    }
}

/* Stop TAU channel 0 timer if no switches are in the process of being debounced */
if (((0 == g_sw1.debounce_counter) && (0 == g_sw2.debounce_counter)) && (0 ==
g_sw3.debounce_counter))
{
    g_debounce_ongoing = FALSE;
    R_TAU0_Channel0_Stop();
}

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

```

r_tau0_channel2_interrupt関数に以下のカスタムコードを追加してください。

```

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

/* TAU0 channel2 interrupt count */
g_tau_ch2_cnt++;

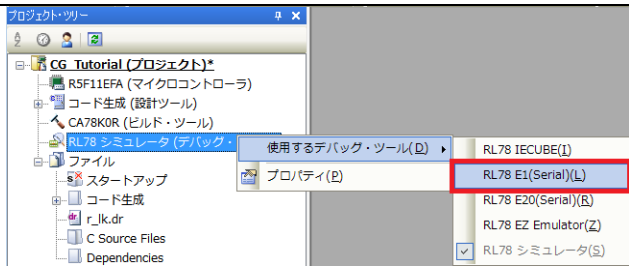
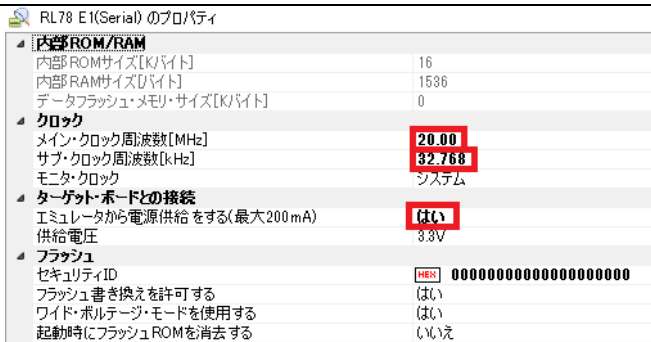
/* End user code. Do not edit comment generated here */

```

7. プロジェクトのビルドとデバッグ設定

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。

<ul style="list-style-type: none"> RL78 シミュレータ (デバッグ・ツール) を右クリックし、RL78 E1 (Serial) を選択してください。 	
<ul style="list-style-type: none"> RL78 E1 (Serial) を右クリックし、プロパティを選択してください。 接続用設定タブをクリックしてください。 以下に設定変更してください。 <p>[クロック] メイン・クロック周波数[MHz]:20MHz サブ・クロック周波数[kHz]:32.768kHz</p> <p>[ターゲット・ボードとの接続] エミュレータから電源供給をする(最大200mA):はい</p> <ul style="list-style-type: none"> その他はデフォルト設定のままにしてください。 	
<ul style="list-style-type: none"> E1 をコンピュータの USB ポートに接続してください。 PMOD1 に LCD モジュールが接続されていることを確認してください。 	

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。	

プログラムは、Pmod LCD に以下を表示します。

```

Renesas
RL78/G1G

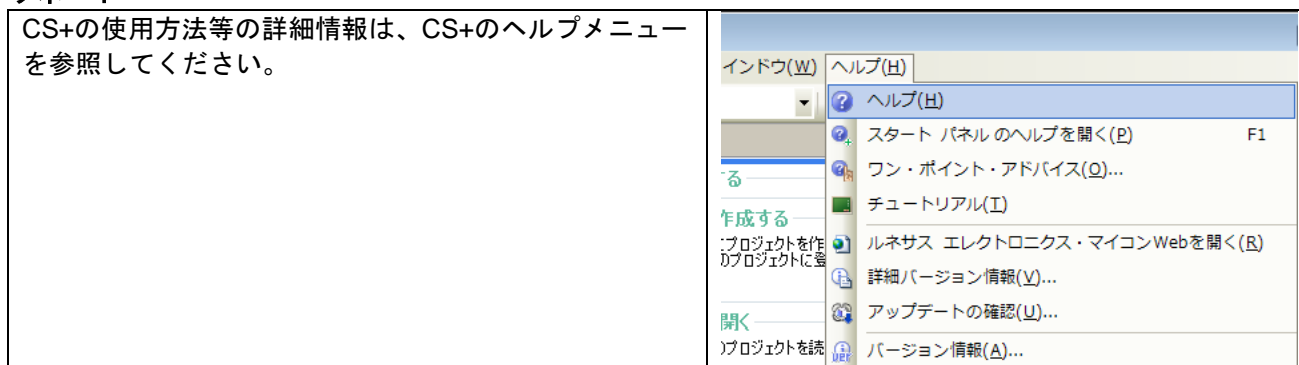
Tutorial sample
Connect USB to PC
Serial configuration:
Baud Rate 19200
Data Bits 8
Stop Bits 1
Parity None
Flow None

```

プログラムは、CPU ボード上の SW3 または、ターミナル画面でキーボードの“c”キーを押すことで A/D 変換を実行します。そして、ポテンショメータでコントロールされた電圧値の A/D 変換結果を LCD およびターミナル画面に表示します。さらに、CPU ボード上のユーザ LED0-3 で A/D 変換回数をバイナリ形式で点灯表示します。

8. 追加情報

サポート



RL78/G1G マイクロコントローラに関する詳細情報は、RL78/G1G 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、RL78 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/rskrl78g1g> (日本サイト)
<http://www.renesas.com/rskrl78g1g> (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

本書の内容の一部または全てを予告無しに変更することがあります。
 本書の著作権はルネサス エレクトロニクス株式会社にあります。ルネサス エレクトロニクス株式会社の書面での承諾無しに、本書の一部または全てを複製することを禁じます。

© 2015 Renesas Electronics Europe Limited. All rights reserved.
 © 2015 Renesas Electronics Corporation. All rights reserved.
 © 2015 Renesas System Design Co., Ltd. All rights reserved.

改訂記録	RSKRL78G1G コード生成支援ツールチュートリアルマニュアル
------	-----------------------------------

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2015.01.15	－	初版発行

RSKRL78G1G コード生成支援ツールチュートリアルマニュアル

発行年月日 2015年1月15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/>

RL78G1G