Smart Analog IC101

R21AN0012JJ0100
Rev.1.00
2014.09.01

サンプルコード導入手順書兼 API Builder SAIC101 仕様書 (RL78/L13 編)

要旨

本ドキュメントでは、Smart Analog IC101 (RAA730101)を制御するための API およびサンプルコードの導入手順について、導入を補助するコード開発支援ツール「API Builder SAIC101」の説明を兼ねて手順を示します。

注: SAIC101 は Smart Analog IC101 の略称です。

動作確認デバイス

Smart Analog IC 101（品名：RAA730101）、RL78/L13（品名：R5F10WMGAFB）

目次

1. 概要

本ドキュメントでは、Renesas Starter Kit for RL78/L13 と Smart Analog IC 搭載 RSK オプション評価ボード「TSA-OP-IC101」を組合わせた環境において、コード開発支援ツール「API Builder SAIC101」を使用して、Smart Analog IC101 の UART 通信接続を制御するための API およびサンプルコードの導入手順を実際の例に沿って説明します。

また、本ドキュメントは、ユーザー環境に応じた Smart Analog IC101 の API およびサンプルコードを簡単に編集、プロジェクトへの組み込み手順を簡素化できるコード開発支援ツール「API Builder SAIC101」の仕様についても説明します。

2. 動作確認条件

本ドキュメントは、下記の条件で動作を確認しています。

表 2-1 動作確認条件

項目	内容
評価ボード	・Renesas Starter Kit for RL78/L13 [R0K5010WMS900BE] - Renesas Starter Kit for RL78/L13 CPU ボード 略称：RSK CPU ボード - Renesas Starter Kit LCD Application Board V2 略称：LCD 拡張ボード ・Smart Analog IC 搭載 RSK オプション評価ボード [TSA-OP-IC101] 略称：SAIC101 評価ボード
使用マイコン	R5F10WMGAFB（RL78/L13）
API Builder SAIC101	Ver1.00
統合開発環境 (CubeSuite+)	V2.02.01　[20 Jun 2014]
C コンパイラ (CubeSuite+)	CA78K0R V4.02.00.03 [16 Jan 2014]
RL78/L13 コードライブラリ (CubeSuite+)	V1.02.01.02 [11 Jun 2014] 注 1
統合開発環境 (e2studio)	V3.0.0.22
C コンパイラ(e2studio)	GNURL78 v14.01
RL78/L13 コードライブラリ (e2studio)	V1.02.00.03 [11 Feb 2014] 注 2

注 1 : CubeSuite+用のコードライブラリはコード生成プラグインに内包されています。本ドキュメントでは「CubeSuite+ Code_Generator for RL78_78K V2.04.00」で動作確認を行っております。
注 2 : e2studio 用のコードライブラリは e2studio 本体に内包されています。

3.　コード開発支援ツール API Builder SAIC101 使用の流れ

API Builder SAIC101 を使用した SAIC101 API およびサンプルコード組み込み手順は以下の流れで行います。

① CubeSuite+または e2 studio でプロジェクトを作成。コード生成ツールを使用してシリアル・アレイ・ユニット等のマイクロコントローラ周辺機能設定を行い、コード生成実行。作業完了後、プロジェクトを閉じる。
② 手順①にて作成したプロジェクトを API Builder SAIC101 で読み込む。
③ API Builder SAIC101 上で SAIC と MCU のシリアル接続設定およびサンプルコードの出力設定を行う。
④ 手順③にて設定した情報に基づき API Builder SAIC101 で API およびサンプルコードをファイル出力、および手順①にて作成したプロジェクトへの自動組み込み。
⑤ 手順④にて API およびサンプルコードが組み込まれたプロジェクトを用いてビルド、動作確認。

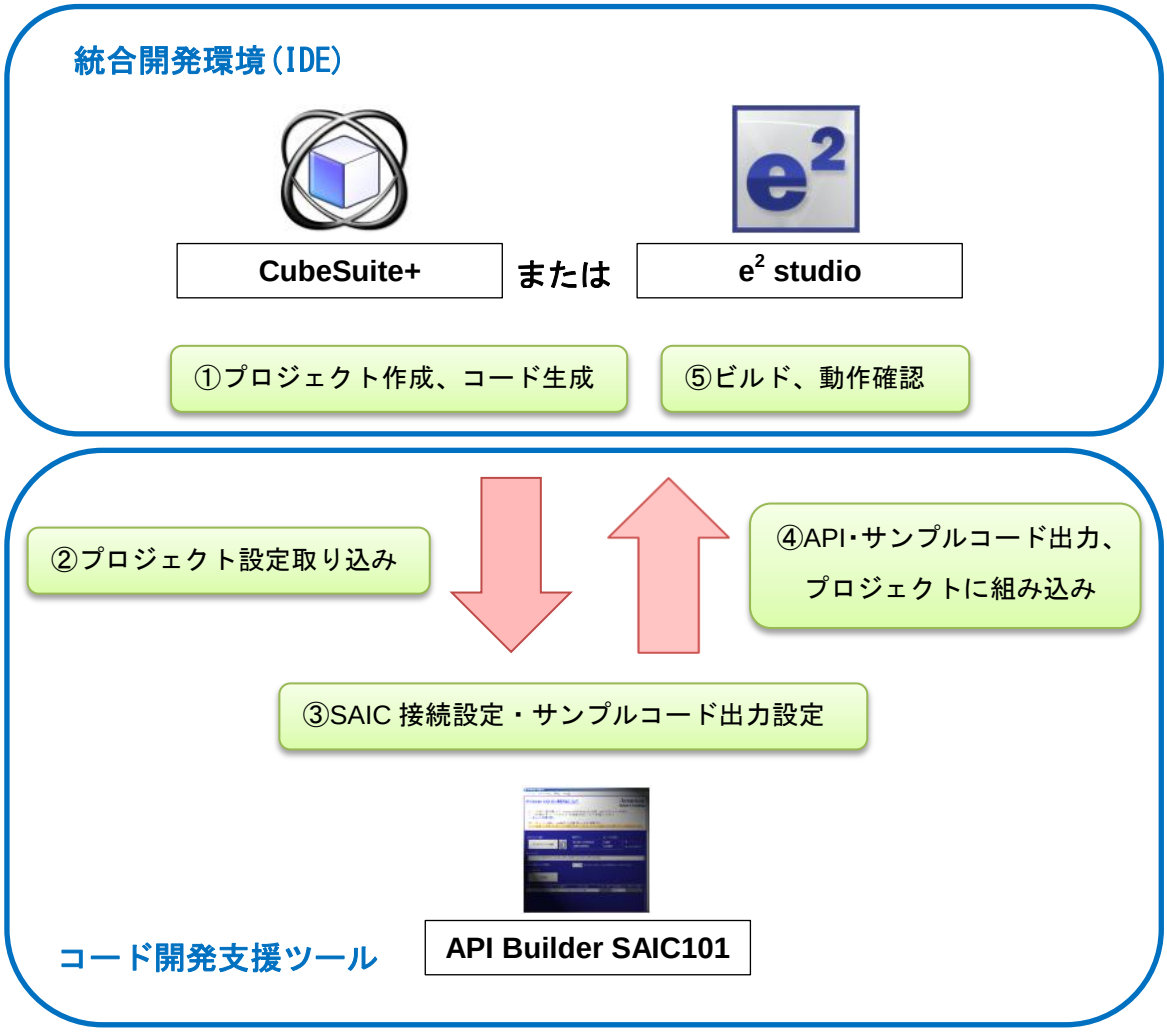

図 3-1　API Builder SAIC101 使用の流れ

4.　SAIC101 API およびサンプルコード組み込みチュートリアル

本章では、Smart Analog IC 搭載 RSK オプション評価ボード[TSA-OP-IC101] (以降、SAIC101 評価ボード)と Renesas Starter Kit for RL78/L13 CPU ボード(以降、RSK CPU ボード)、Renesas Starter Kit LCD Application Board V2 (以降、LCD 拡張ボード)を組み合わせた環境注で、SAIC101 評価ボードに搭載されているサーミスタを動作させるサンプルコードを CubeSuite+のコード生成機能と API Builder SAIC101 を用いて作成し、実行する手順について説明します。本章のチュートリアルでは SAIC101 と MCU 間に UART 接続を使用した例をあげて説明します。

注 : ボードの接続設定につきましては「Smart Analog Easy Starter 101 セットアップ手順書 (RL78/L13 編)」(R21AN0011J)をご参照ください。

4.1　CubeSuite+を用いる際の注意事項

CubeSuite+のプロジェクト・ツリー上に「コード生成 (設計ツール)」が表示されていない場合は、CubeSuite+のメニューバーより「ツール(T)」->「プラグインの管理(P)…」を選び、「プラグインの管理」ウィンドウの「追加機能」タブ内の「コード生成プラグイン」および「コード生成プラグイン 2」にチェックを入れて CubeSuite+を再起動してください。

動作確認を行う際は、デバッグ・ツールの設定に注意してください。CubeSuite+のプロジェクト・ツリー上に表示されているデバッグ・ツールの設定は初期状態で「RL78 シミュレータ(デバッグ・ツール)」となっておりますので、その文字の上で右クリック->「使用するデバッグ・ツール(D)」から使用するエミュレータを選択してください。例えば E1 を選択する際は「RL78 E1(Serial)(L)」を選択します。

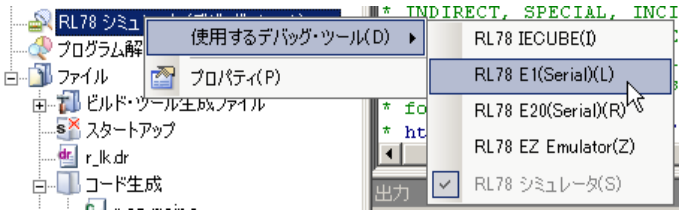

図 4-1　使用するデバッグ・ツールの設定

4.2　CubeSuite+を用いたプロジェクトの作成とコード生成の設定

最初に CubeSuite+のコード生成機能を用いて、ひな形となるプロジェクトの作成およびマイクロコントローラの周辺機能の設定を行います。4.2.1 以降の手順に従って実施してください。

4.2.1　プロジェクトの作成

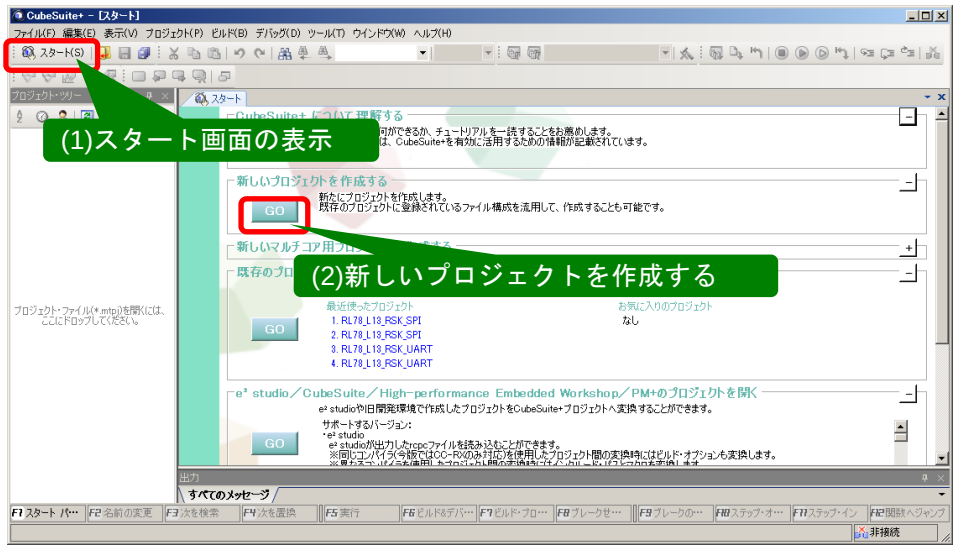

図 4-2　プロジェクトの作成

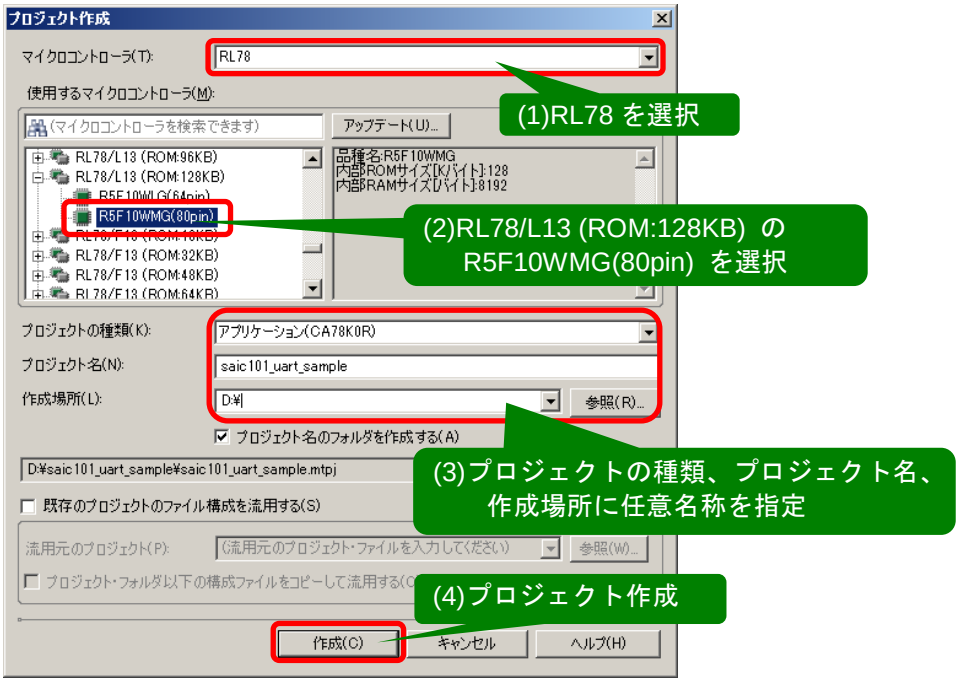

図 4-3　CubeSuite+プロジェクト作成画面

4.2.2　コード生成処理(設計ツール)の設定と実行

「コード生成 (設計ツール)」->「周辺機能」以下について、それぞれ設定を行ってください。

① 周辺 I/O リダイレクション・レジスタ(PIOR)の設定

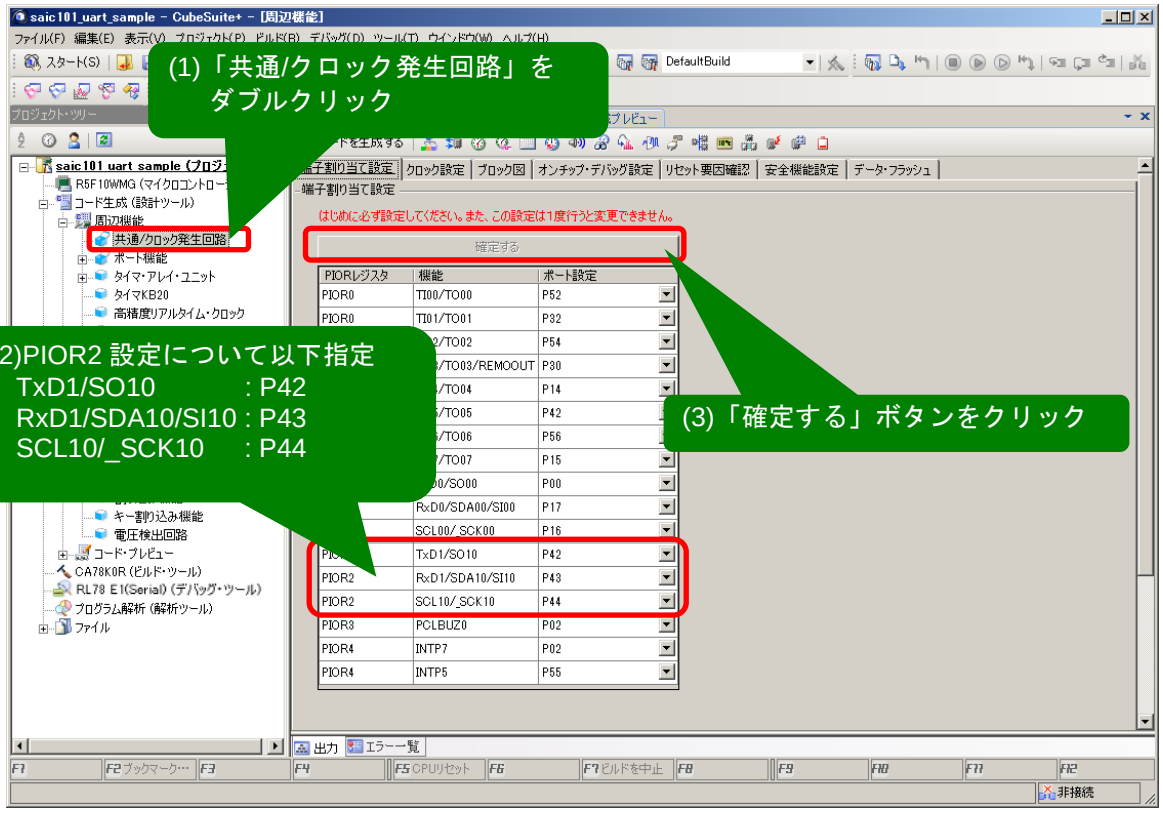

図 4-4　周辺機能 – 端子割り当て設定

② SAIC101 との通信に使用するシリアル・アレイ・ユニットの設定

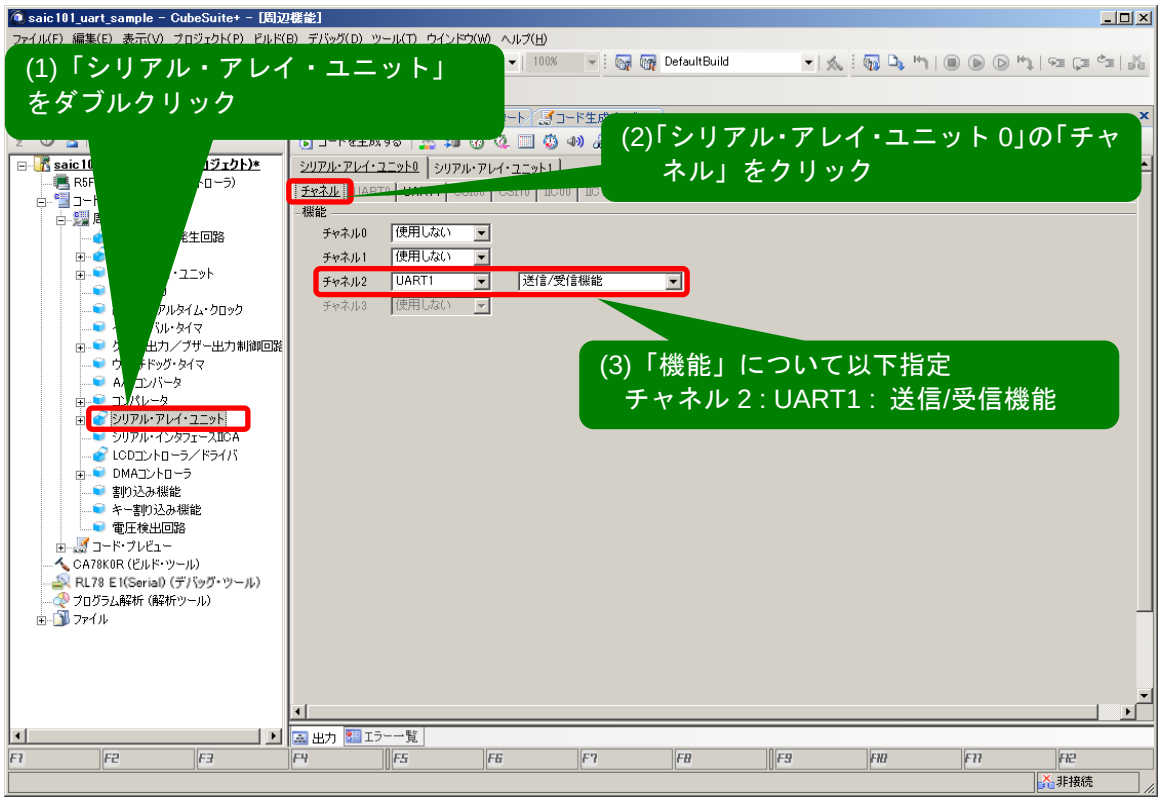

図 4-5 シリアル・アレイ・ユニット 0 – チャネル設定

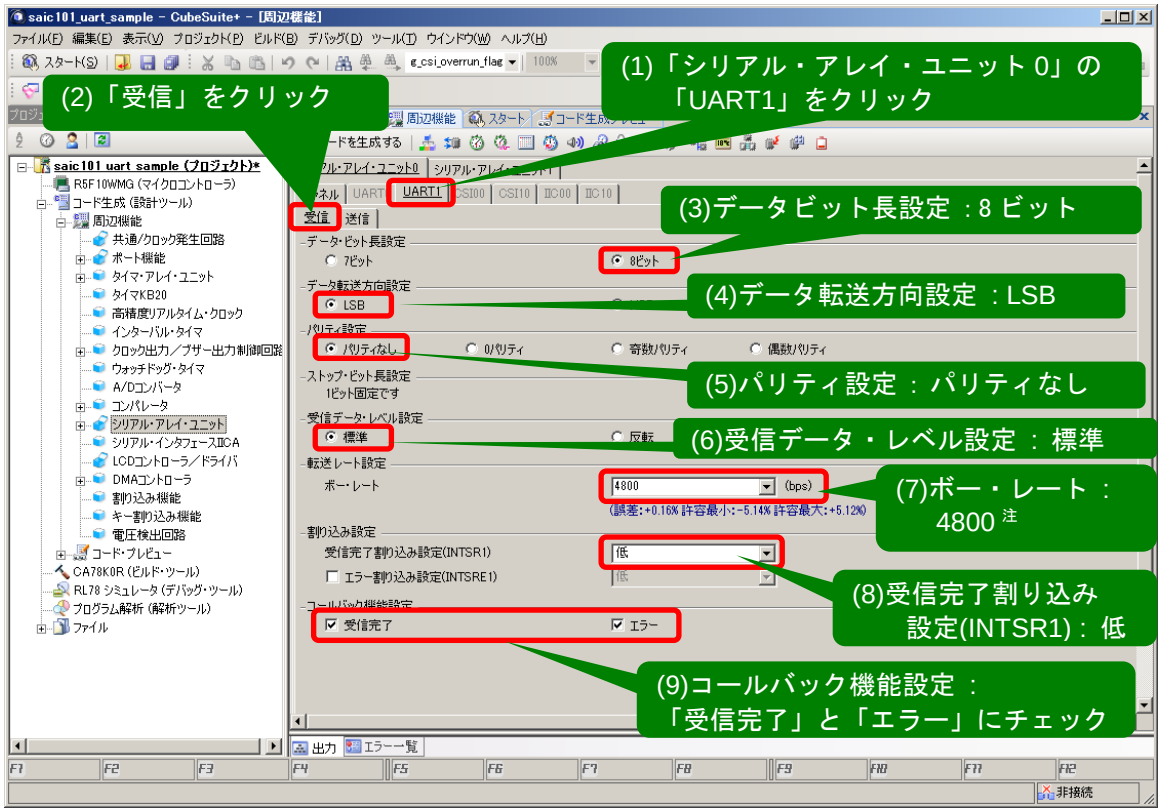

図 4-6 UART1 設定 - 受信設定

注: ボー・レートの設定はプルダウンから選択せず、直接数字を入力して値を指定してください。

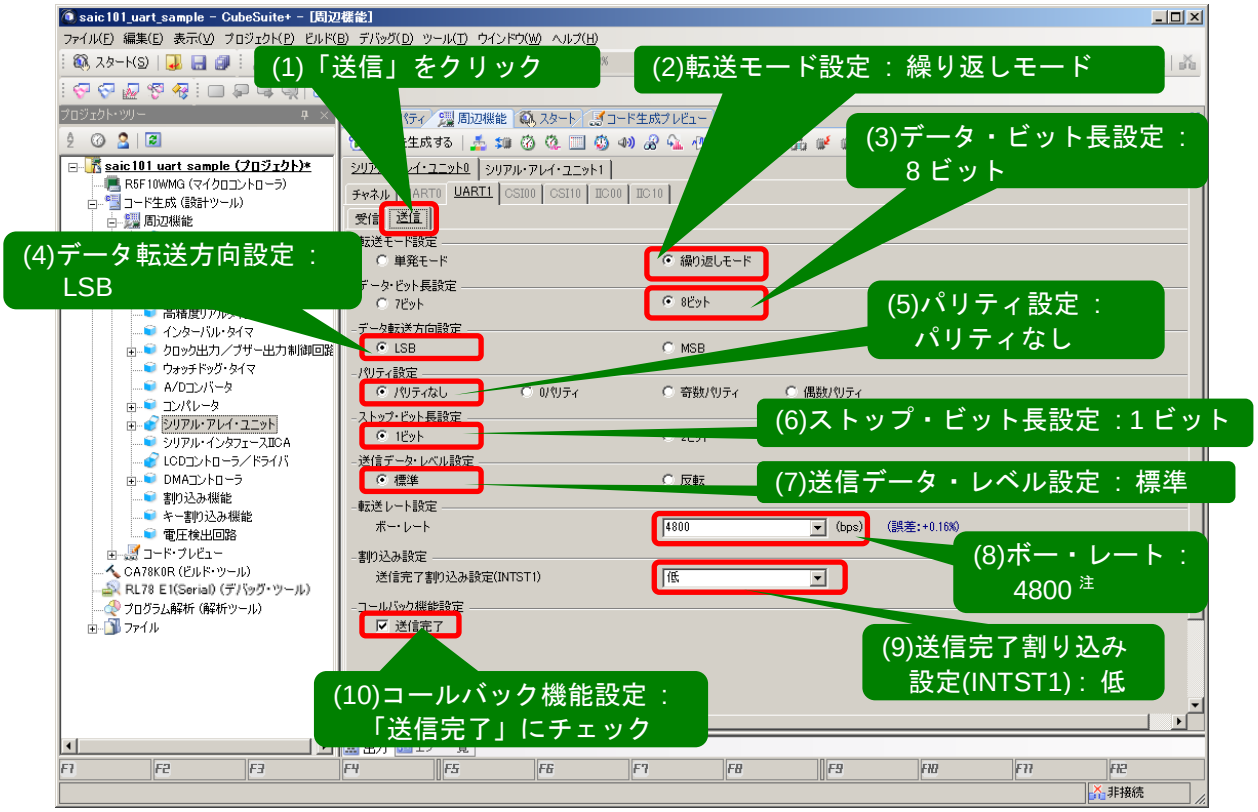

図 4-7　UART1 設定 - 送信設定

注: ボー・レートの設定はプルダウンから選択せず、直接数字を入力して値を指定してください。

③ ウォッチドッグ・タイマの設定

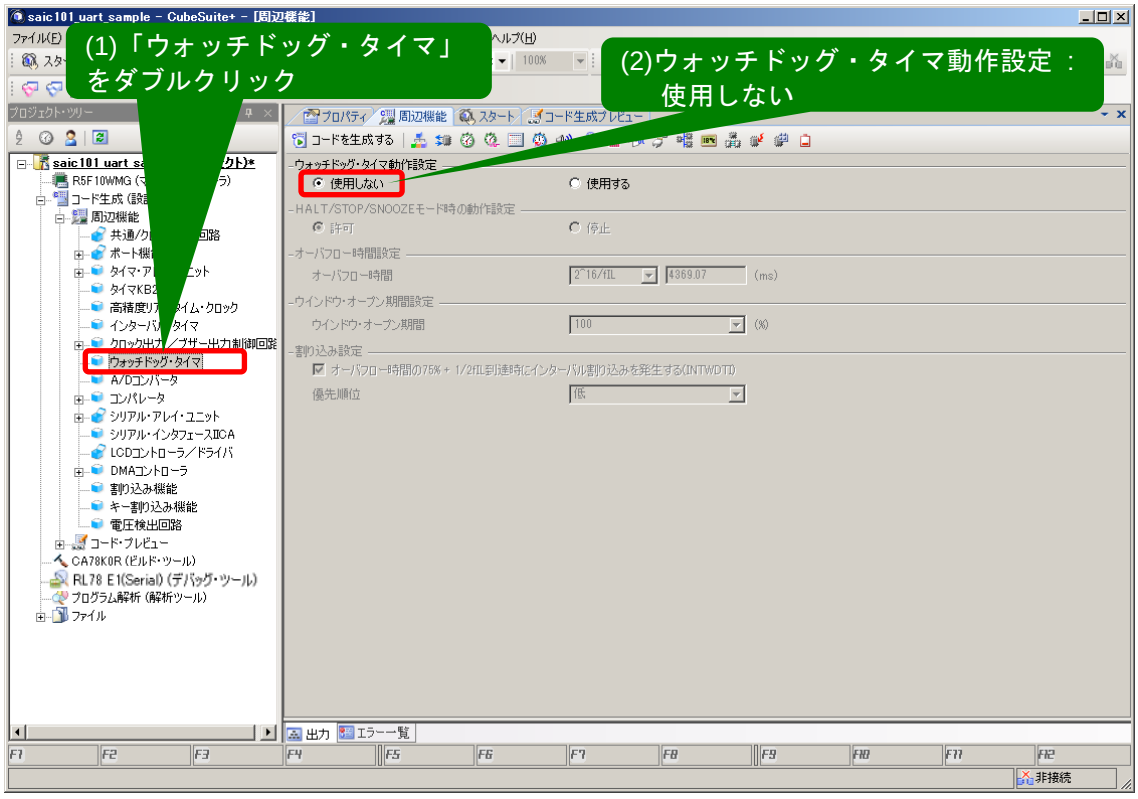

図 4-8　ウォッチドッグ・タイマ設定

④ LCD コントローラ/ドライバ設定

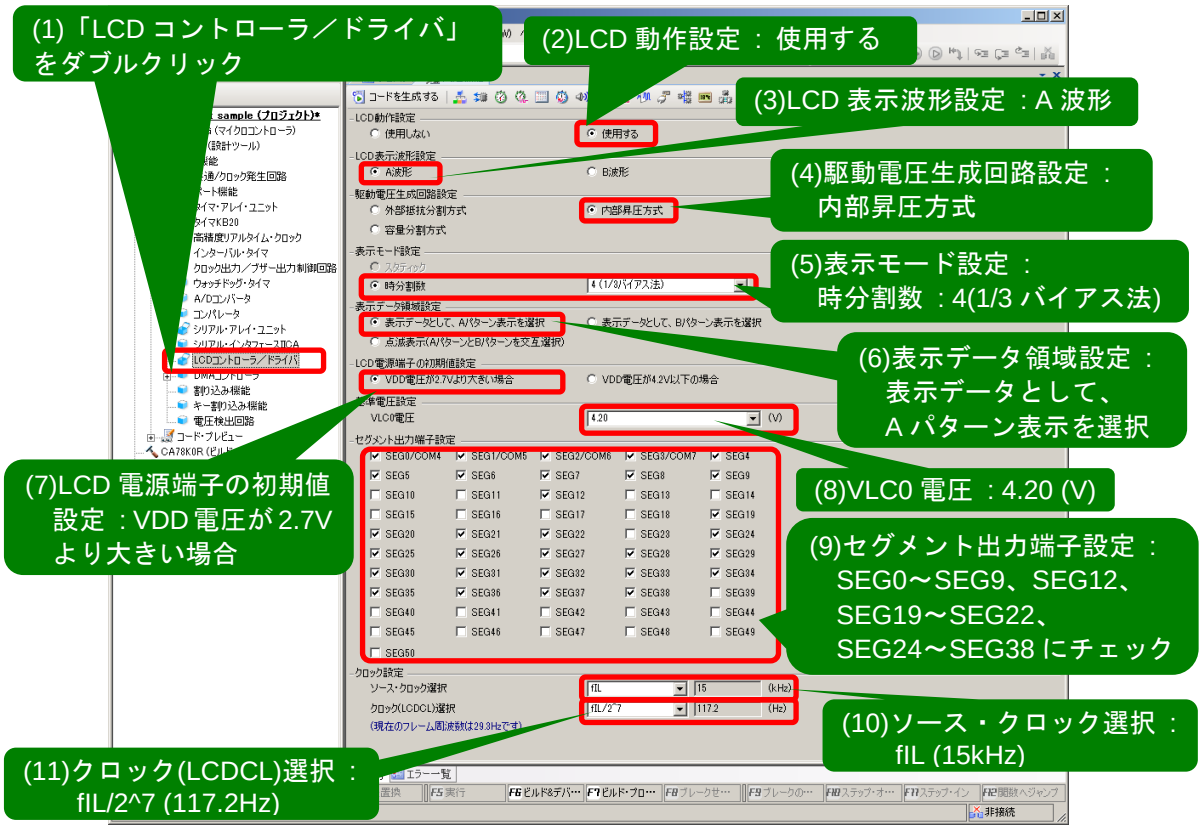

図 4-9　LCD コントローラ/ドライバ設定

⑤ コード生成の実行

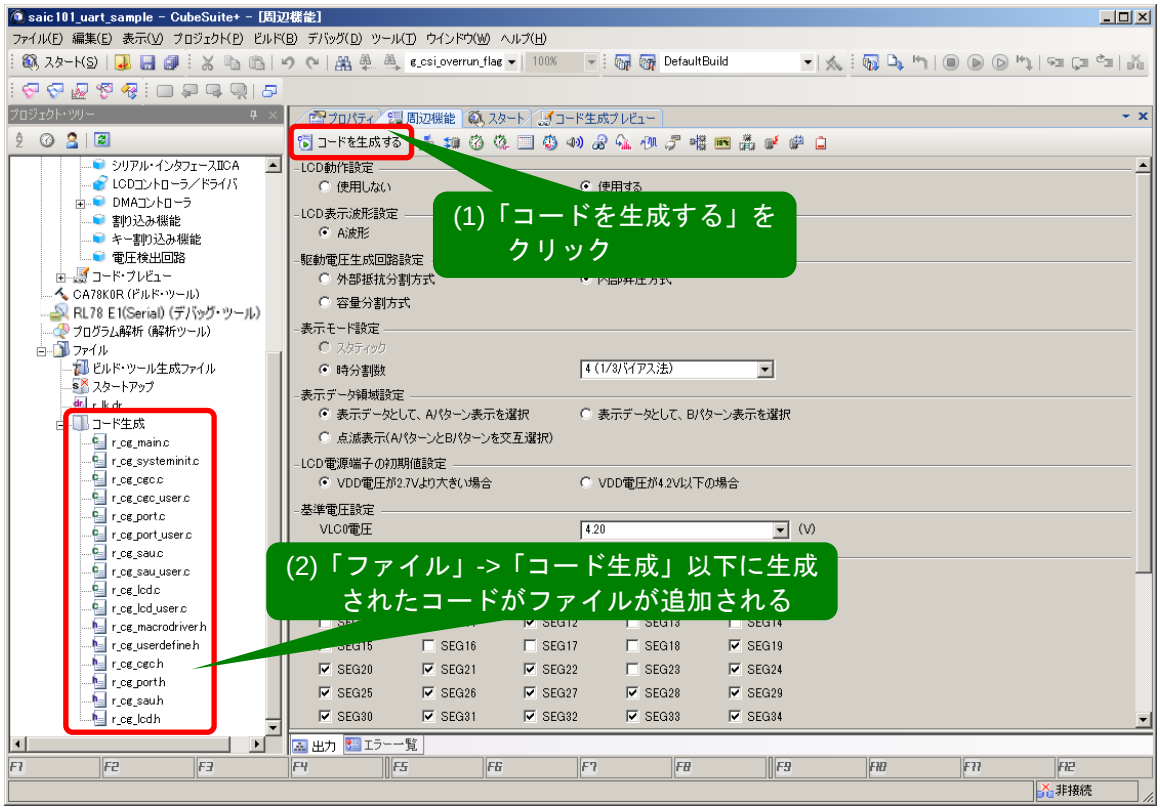

図 4-10　コード生成実行画面

⑥ CubeSuite+での変更内容を全て保存し、終了

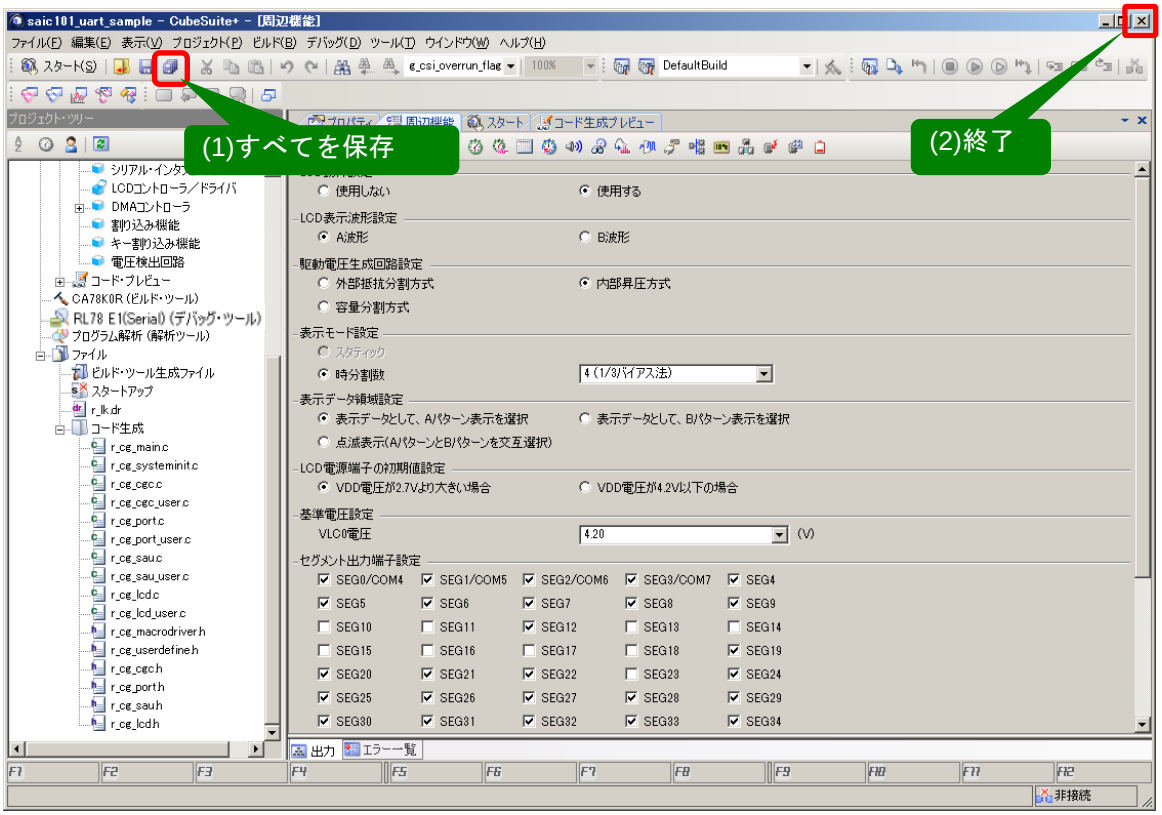

図 4-11　保存と終了

4.3　API Builder SAIC101 の適用

4.2.2 で作成したプロジェクトおよびソースコードに対して、API Builder SAIC101 にて API ファイルの追加とソースコードの変更を行います。ファイルを登録する初回設定時は必ず CubeSuite+上でプロジェクトファイルを閉じた状態で本手順を実施して下さい。

① API Builder SAIC101 の起動

図 4-12　アイコン画面

② API の組み込み対象プロジェクトの読み込みと情報表示

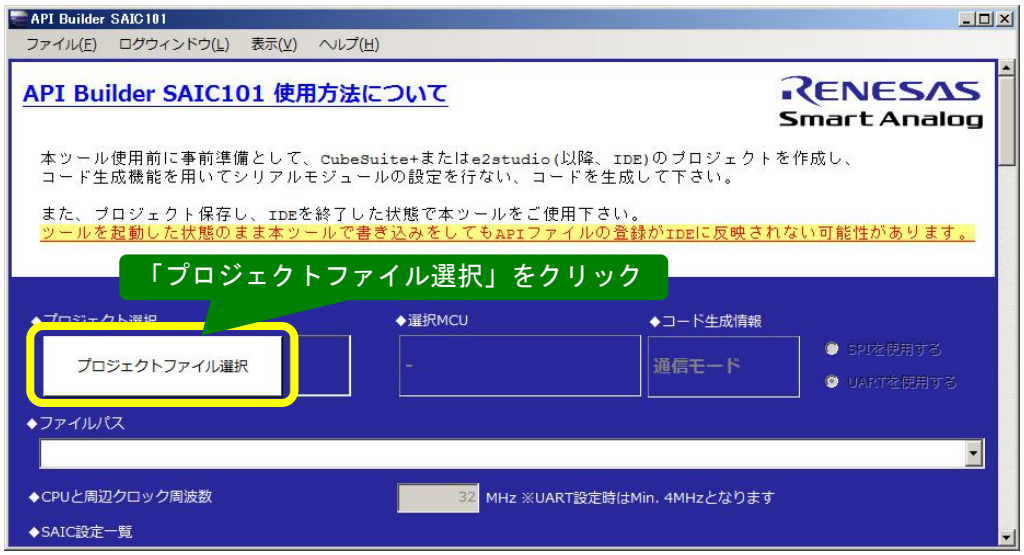

図 4-13　プロジェクトファイルの読み込み

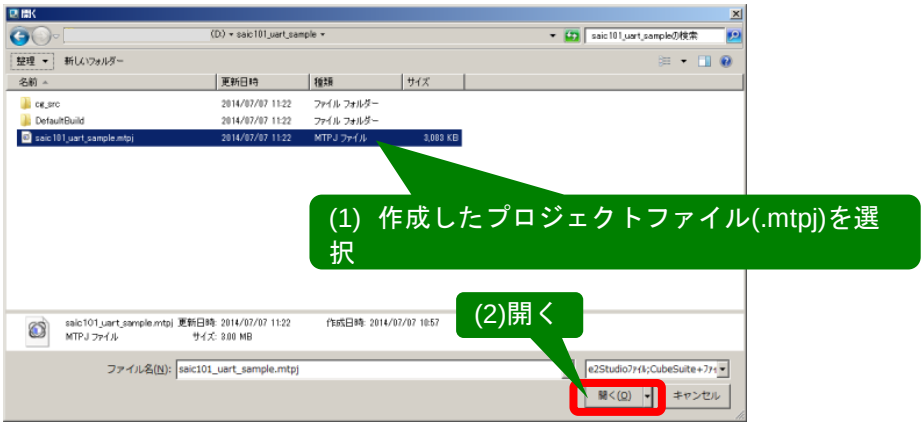

図 4-14　ファイルオープンダイアログ

図 4-15　読み込みプロジェクト情報確認

③ SAIC101 との接続設定、サンプルコード出力設定、ファイル出力実行

図 4-16　ツール設定画面

④ CubeSuite+起動し API ファイルが組み込まれていることを確認

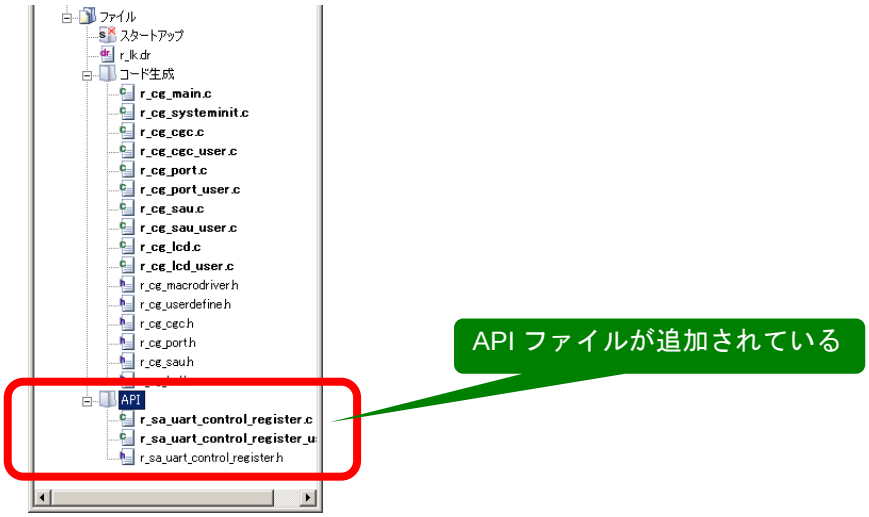

図 4-17　API ファイルの組み込み確認

⑤ 「r_cg_main.c」の「main」関数内にサーミスタ制御サンプルが追加されていることを確認

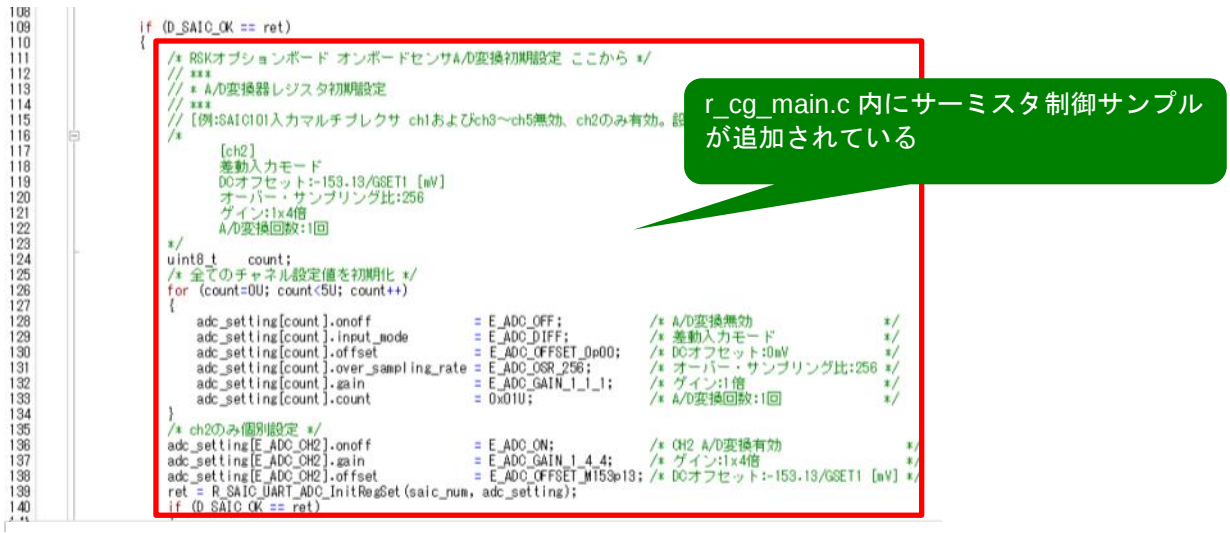

図 4-18　サーミスタ制御サンプルコード出力の確認(r_cg_main.c)

⑥ ソースコードがビルドできることを確認

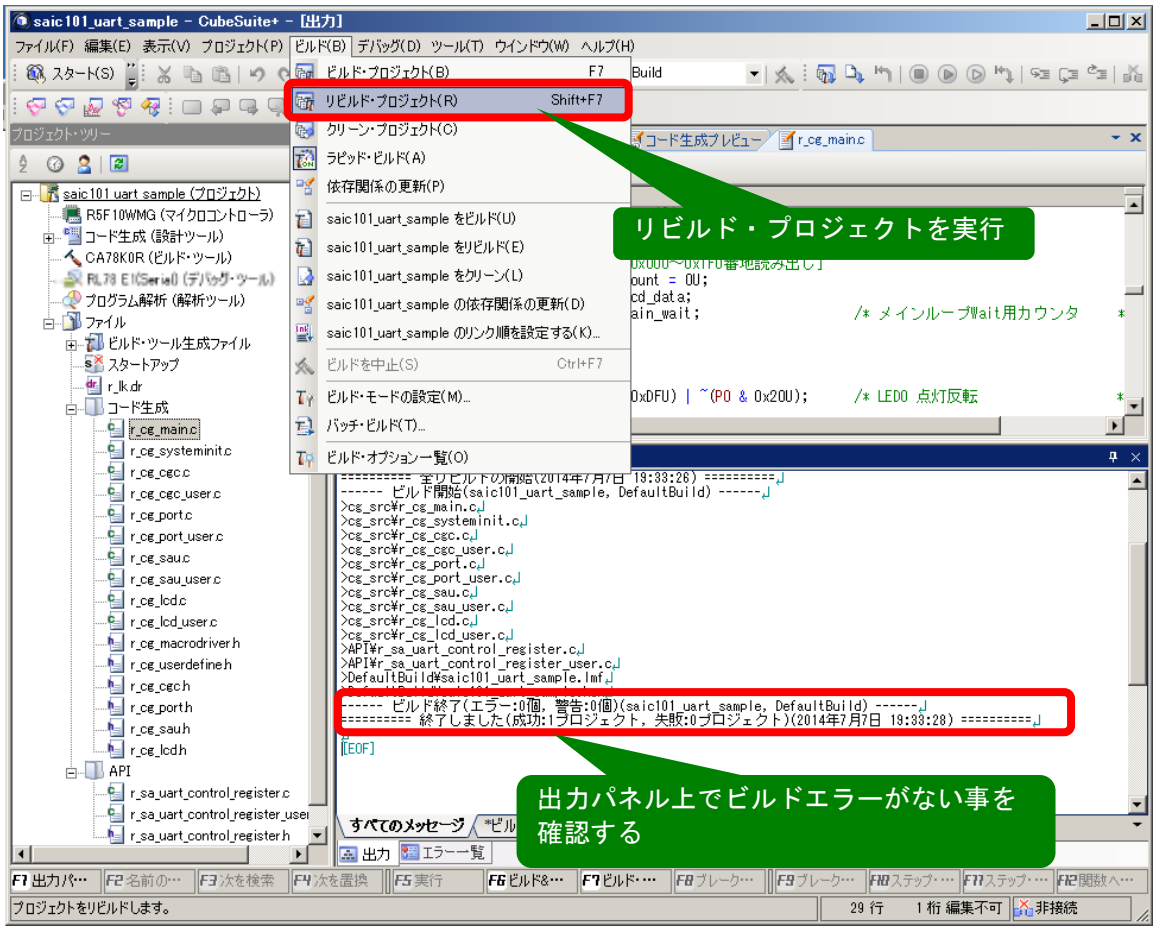

図 4-19　CubeSuite+ リビルド・プロジェクト結果画面

⑦ ターゲットボード接続、プログラムのダウンロード後、サンプルコードを実行

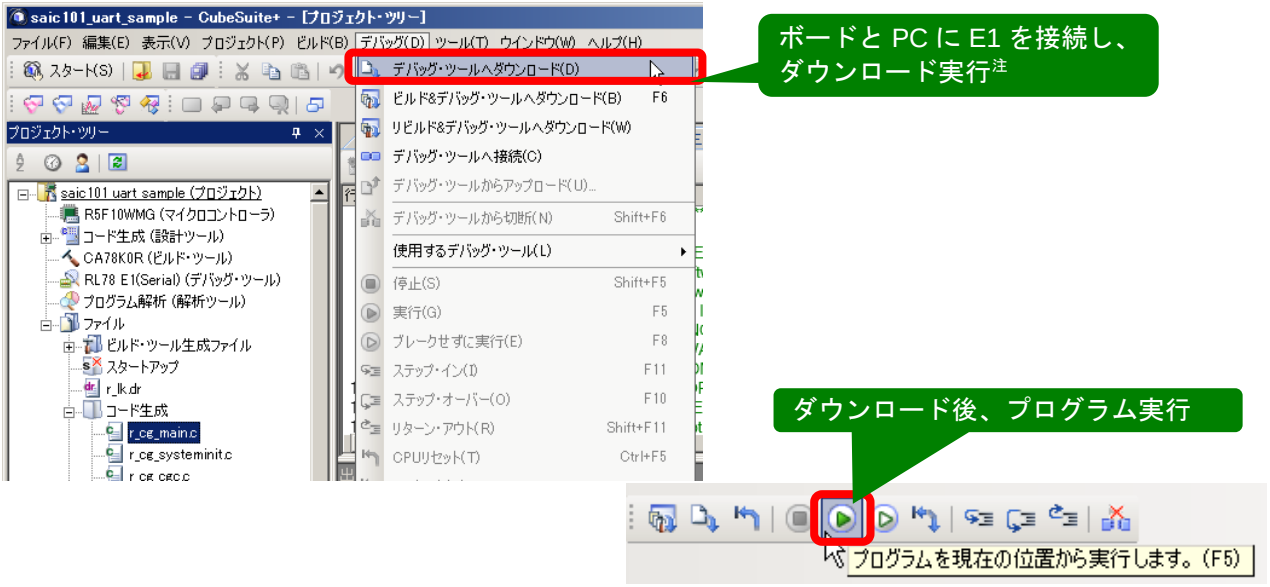

図 4-20　CubeSuite+ ダウンロード・実行画面

注：エミュレータからの電源供給が必要な場合は、本手順実施前に「RL78 E1(Serial)(デバッグ・ツール)」の「プロパティ」の「ターゲット・ボードとの接続」から「エミュレータから電源供給をする(最大 200mA):はい」および「供給電圧:5.0V」を設定してください。

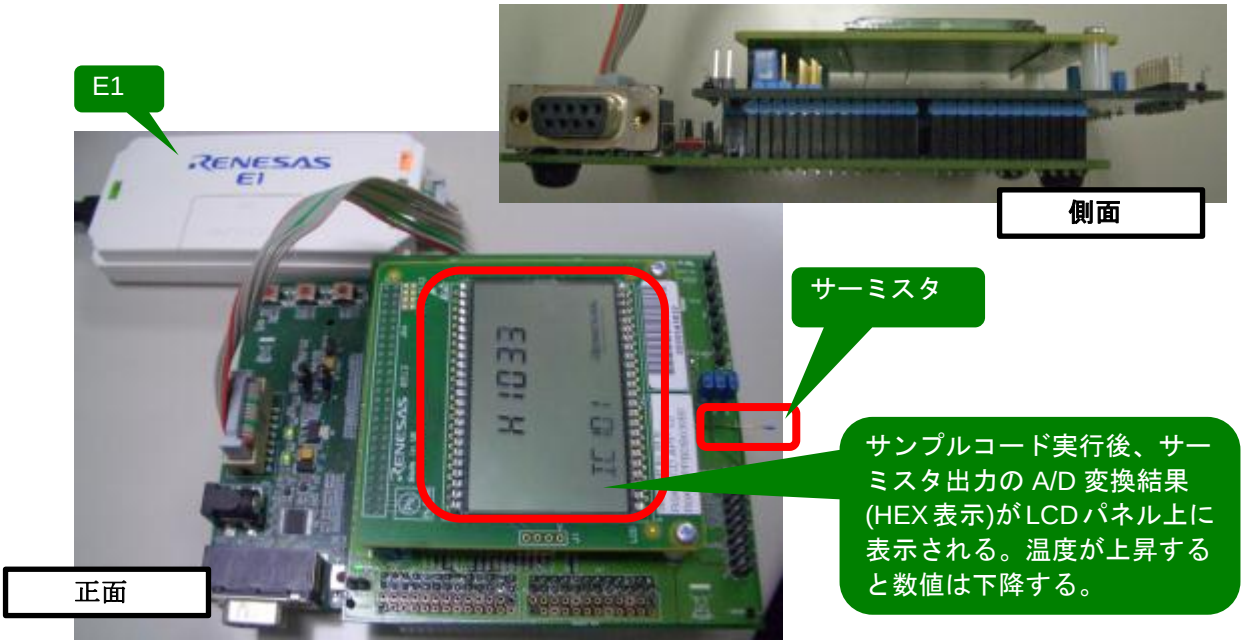
E1
側面
サーミスタ
サンプルコード実行後、サーミスタ出力の A/D 変換結果(HEX 表示)が LCD パネル上に表示される。温度が上昇すると数値は下降する。
正面

図 4-21　動作確認

5.　API Builder SAIC101

5.1　概要

API Builder SAIC101 は、Smart Analog IC101 を制御するための API を編集したのち API を使用した各種サンプルコードを生成し、プロジェクトに API を組み込む作業の補助をするためのコード開発支援ツールです。ユーザーの環境に応じた API の編集およびサンプルコードの選択、出力等の操作がすべて GUI による簡単操作で行えます。

5.2　システム構成

図 5-1　システム構成図

注 1: 現在対応している統合開発環境は、ルネサスエレクトロニクス社製 CubeSuite+及び e2 studio となります。

5.3　主な機能

- CubeSuite+や e2 studio で作成したプロジェクトファイルへ SAIC101 サンプルコードの自動組み込み。
- ユーザーの環境に応じたシリアル・インターフェース設定を GUI 操作で簡単に編集、API へ反映。
- 編集後の API を使用した各種サンプルコードから組み込み対象を選択し、RSK ボード向けのサンプル C ソースコードを出力。[注]

注 ： 現行サポートは RL78/L13 のみ

5.4　サポート環境

表 5-1　サポート環境

サポート OS	• Windows® 8 (32 ビット版、64 ビット版) • Windows® 7 (32 ビット版、64 ビット版)
Windows OS 以外に必要なソフトウェア環境	.Net Framework 4 以上

5.5　対応 MCU

- RL78/L13

  R5F10WLA
  R5F10WLC
  R5F10WLD
  R5F10WLE
  R5F10WLF
  R5F10WLG
  R5F10WMA
  R5F10WMC
  R5F10WMD
  R5F10WME
  R5F10WMF
  R5F10WMG

5.6　画面構成の説明

API Builder SAIC101 の画面構成を以下に説明します。

① プロジェクトの選択と情報表示

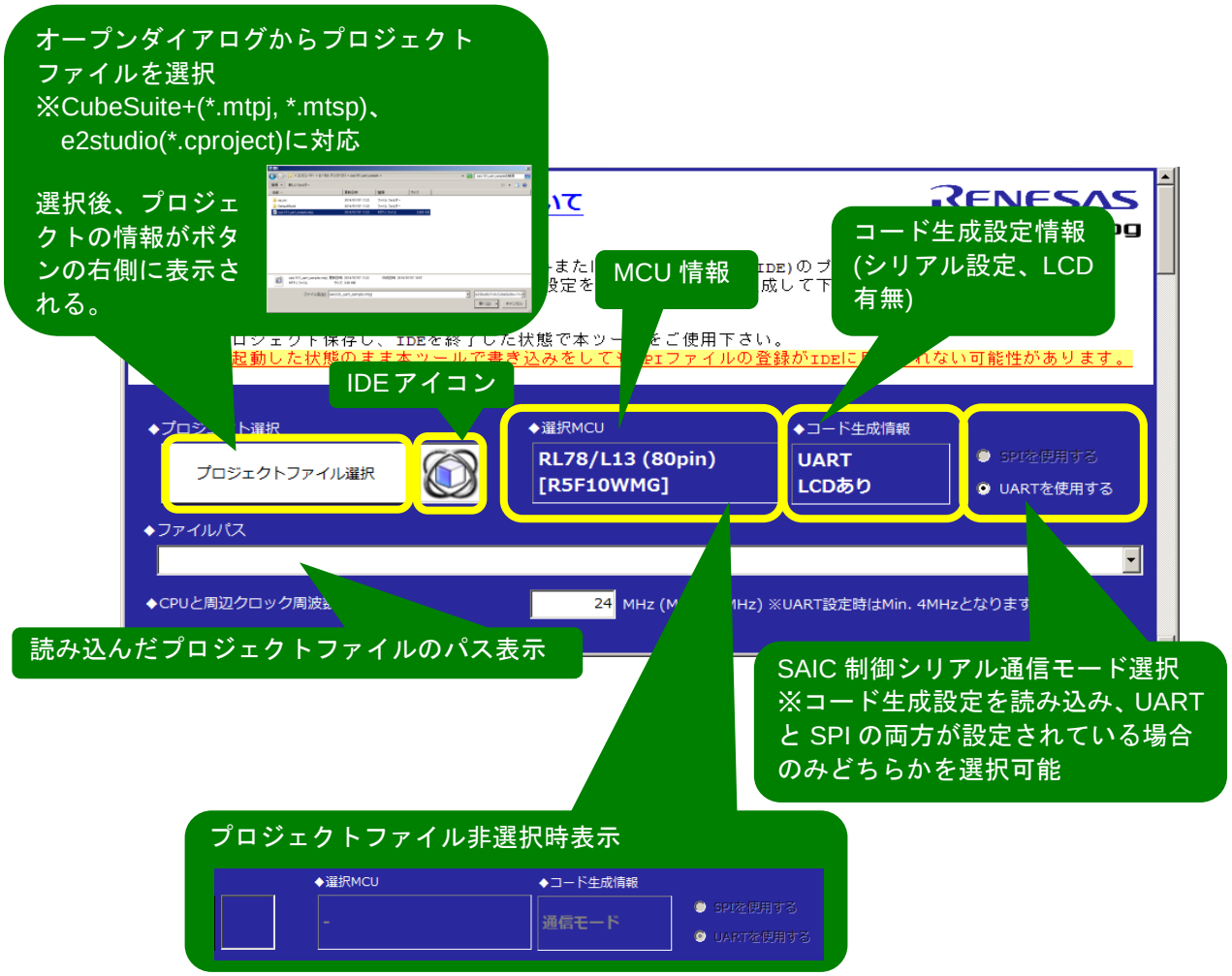

図 5-2　ファイル読み込み・表示更新確認

② ハードウェア依存部分の設定と、出力するサンプルコードの選択

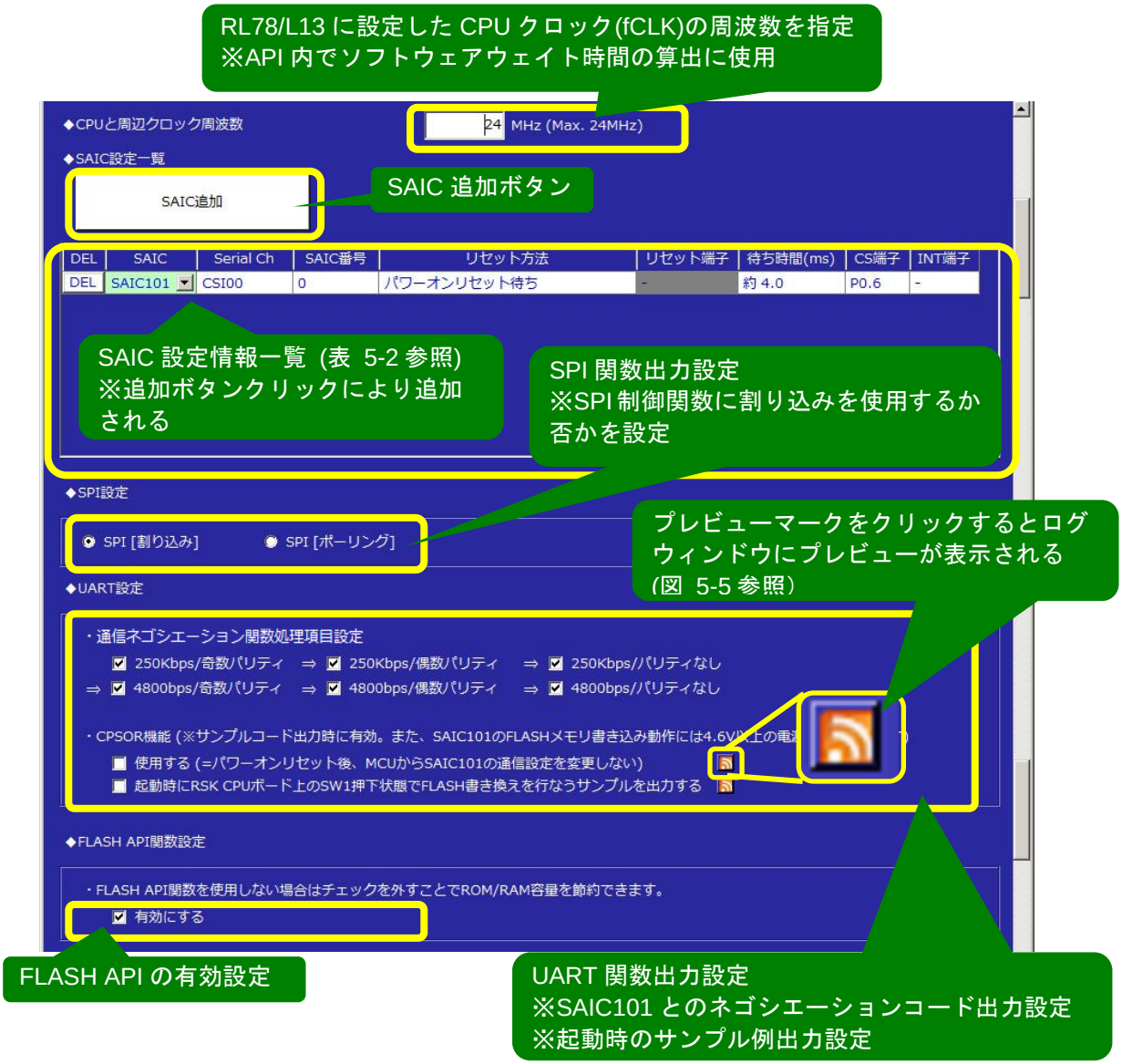

図 5-3　ハードウェア依存部分の設定と、出力するサンプルコードの選択(1/2)

図 5-4　ハードウェア依存部分の設定と、出力するサンプルコードの選択(2/2)

表 5-2　SAIC 設定情報一覧

項目	説明	備考
DEL	SAIC 設定情報一覧から該当行を 1 行削除する	―
SAIC	接続対象 SAIC の型名を指定する	SPI 時はプルダウンメニューから SAIC101、SAIC300、SAIC301、SAIC500、SAIC501、SAIC502 が選択可能。UART 時は SAIC101 のみ選択可能。
Serial Ch	シリアル ch を指定する	プルダウンメニューからコード生成にて設定済みのシリアル ch が選択可能。
SAIC 番号	API 関数を呼び出す際に、引数として指定する番号	unsigned char 型の範囲(0～255)で指定可能。注 1
リセット方法	SAIC の起動時のリセット処理を指定する	プルダウンメニューから以下を選択可能。 • パワーオンリセット待ち注 2 「待ち時間(ms)」で指定した時間分ウエイト • 外部リセット(RESET ポート=L)注 3 「リセット端子」で指定したポートを、「待ち時間(ms)」で指定した時間分 L 出力 • 内部リセット(RESET レジスタ=1)注 3 SAIC のリセット制御レジスタ(RC)の RESET ビットに 1 をセット後クリア
リセット端子注 3	リセット端子に接続しているポートを指定する	「リセット方法」に外部リセット選択時、指定可能。外部リセット選択時にリセット端子を指定しないとファイル出力時にエラーが発生するため、必ず設定してください。
待ち時間(ms)	「リセット方法」にパワーオンリセット待ち選択時、ウエイト時間を指定する。外部リセット選択時、リセット端子を L にする時間を指定する。	float 型の範囲で指定可能。
CS 端子	CS 端子接続しているポートを指定する	SPI 時のみ設定可能。SPI 時に CS 端子を指定しないとファイル出力時にエラーが発生するため、必ず設定してください。
INT 端子	INT 端子に接続しているポートを指定する	SPI 時、かつ SAIC101 のみ設定可能

注 1 : サンプルコードは API 関数呼び出し時に SAIC 番号を 0 固定としているため、サンプルコード使用時は 0 を設定してください。
注 2 : パワーオン・リセット機能は SAIC101 と SAIC502 のみ対応。
注 3 : SAIC 欄に SAIC101 を指定時は設定不可。

5.7　ログウィンドウ

ログウィンドウについて以下に説明します。

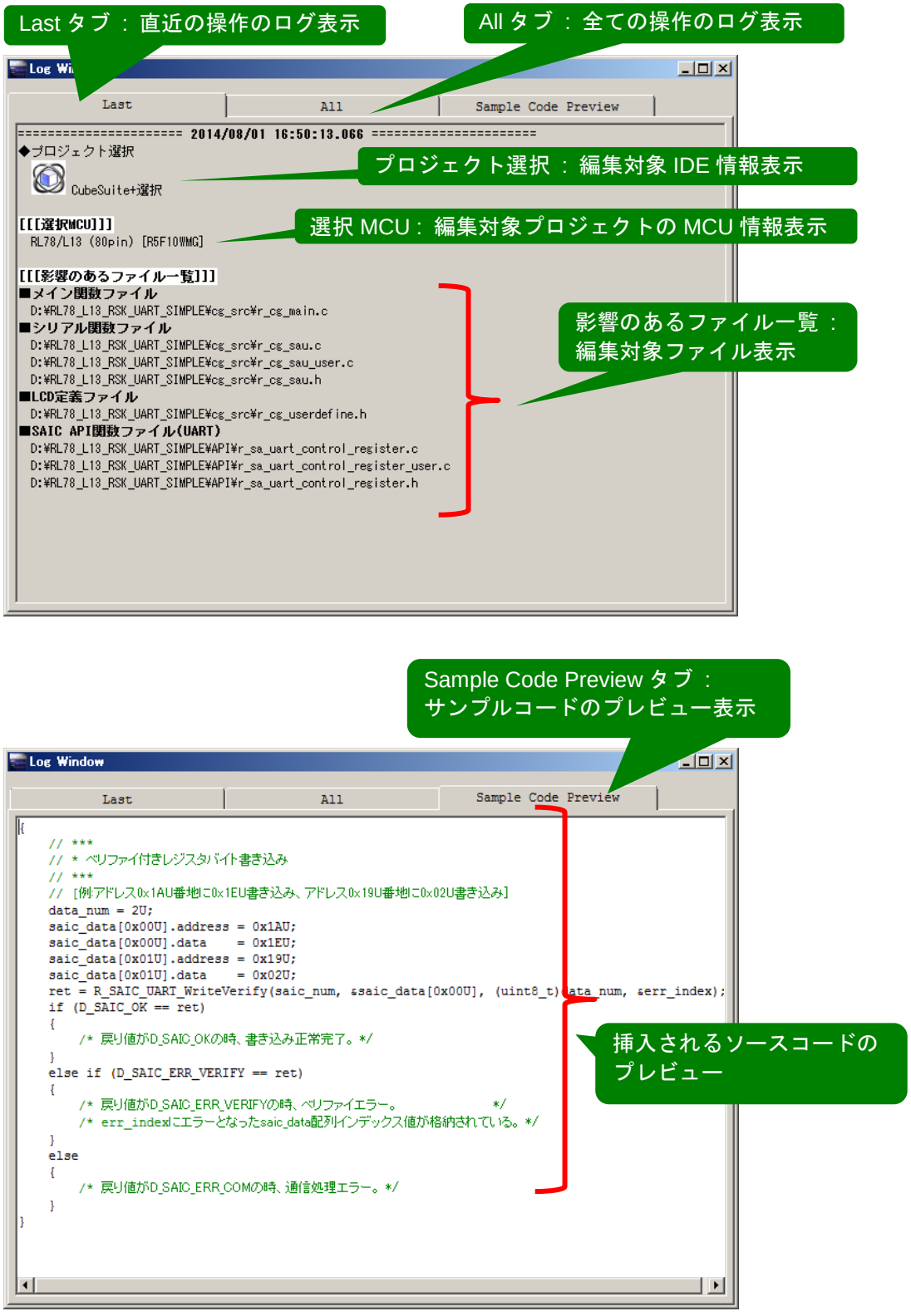

図 5-5　Log Window 画面

5.8　エラーメッセージ

エラーメッセージと、その説明を表 5-3 に示します。致命的エラーが表示された場合は構成ファイルが壊れている可能性がありますので、弊社から配布している API Builder SAIC101 の ZIP ファイルから再インストールをしてください。

表 5-3 エラー一覧

No	エラー画面	備考
1	[致命的エラー] CHIP 情報の CSV ファイル形式が正しくありません。構成ファイルが壊れている可能性があります。	本ツールでは CSV ファイルにマイコン固有情報を格納しています。エラーの場合は読み込んだマイコン用の CSV ファイル形式が正しくありません。
2	[エラー] シリアル設定が自動生成されていません。	本 API ではコード生成ツールで出力される関数を使用しています。そのためツール実行前にコード生成でシリアルインタフェース(UART/SPI)の設定を生成してください。
3	[エラー] 対応した MCU の情報ファイルがありません。	本ツール対応の MCU 情報ファイルがありません。Chips フォルダに対象プロジェクトで選択した MCU 名の情報ファイルがあるか確認してください。ない場合は非対応の MCU となります。
4	[エラー] SAIC の設定が 1 つもありません。	ファイル出力時に SAIC が 1 つも設定されていない場合に表示されます。最低 1 つ以上をご指定ください。
5	[エラー] 書き込みエラーが発生しました。 [該当書き込みファイル名]	ファイル書き込み時にエラーが発生した場合に表示されます。エラーが発生した該当のファイル名が表示されます。
6	[エラー] SPI の SAIC 番号が重複しています。 UART の SAIC 番号が重複しています。	「SAIC 番号」が重複していた場合に表示されます。必ずユニークな番号を設定してください。
7	[エラー] ポート:PX.X が重複しています。	「CS 端子」及び「リセット端子」が重複していた場合に表示されます。対象ハードウェアの回路設定をよく確認し、正しいポート番号をご指定ください。
8	[エラー] CS 端子が設定されていません	「CS 端子」が設定されていない場合に表示されます。SPI 通信時は必須設定となりますので、必ず CS 端子を設定してください。
9	[エラー] リセット時間の値が不正です。	「待ち時間(ms)」に数値以外の値が入力された場合に表示されます。数値を入力してください。
10	[エラー] リセット端子の設定値が不正です。	「リセット端子」の設定値が正しくない場合に表示されます。正しい端子名を設定してください。
11	[エラー] 1 つの UART に対して複数の SAIC が設定されています。	UART のチャネル 1 つに対して SAIC は 1 つのみ設定できます。重複は出来ませんので、それぞれ別々のチャネルを設定してください。
12	[入力エラー] 0 ～ 255 の間を設定して下さい。	SAIC 番号の有効範囲は 0～255 です。有効範囲内で設定してください。

6. API Builder SAIC101 を使わず API 関数を組み込む場合

RSK CPU ボード＋SAIC101 評価ボード環境にて API を組み込むために API Builder SAIC101 が設定するファイル一覧を表 6-1 に示します。API Builder SAIC101 を使用しない場合には、本章で示すソースコードを手動で変更していただく必要があります。

本章では UART 向け API 組み込みを例に説明いたします。SPI 向けについては、uart/UART の文字を spi/SPI に読みかえてご参照ください。本章では、UART 向けは UART1、SPI 向けは CSI10 を使用した場合を例に説明します。

表 6-1　API を組み込む際に設定が必要なファイル

ファイル名	備考
r_cg_main.c	コード生成ファイル
r_cg_sau.c	コード生成ファイル
r_cg_sau.h	コード生成ファイル
r_cg_sau_user.c	コード生成ファイル
r_sa_uart_control_register.c	API ファイル注
r_sa_uart_control_register.h	API ファイル注
r_sa_uart_control_register_user.c	API ファイル注

注：API ファイルはサンプルコードからコピーしてご使用ください。UART 用は UART¥source フォルダ以下に、SPI 用は SPI¥source フォルダ以下に格納されています。

各ファイルの設定内容を下記に示します。

― r_cg_main.c 修正内容一覧

- [UART/SPI 共通] API 関数を使用するためのインクルードの追加

```
#include <stddef.h>
#include "r_sa_uart_control_register.h"
```

- [UART/SPI 共通] R_MAIN_UserInit 関数内に API 初期化関数の追加

```
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    EI();

    // ***
    // * SAIC 初期化(UART)
    // ***
    /* 必ずメインループの前にコールして下さい。 */
    R_SAIC_UART_Init();    /* SmartAnalogIC Initialize. */

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

― r_cg_sau_user.c 修正内容一覧

- [UART/SPI 共通] API 関数を使用するためのインクルードファイルの追加

```
#include "r_sa_uart_control_register.h"
```

- [UART/SPI 共通]ビット制御用ファイル内グローバル変数の追加

```
static const uint8_t  gs_bit_tbl[] =
{
    0x01U, 0x02U, 0x04U, 0x08U, 0x10U, 0x20U, 0x40U, 0x80U,
};
```

- [UART のみ] API 関数で使用するフラグ用ファイル間グローバル変数追加

```
uint8_t g_uart_tx_end_flag = 0U;
uint8_t g_uart_rx_end_flag = 0U;
```

- [SPI のみ] API 関数で使用するフラグ用ファイル間グローバル変数追加

```
uint8_t g_csi_overrun_flag = 0U;
```

- [UART のみ] r _uart1_callback_receiveend 関数内にフラグ更新処理を追加

```
static void r_uart1_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    g_uart_rx_end_flag |= gs_bit_tbl[E_UART1];

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

- [SPI のみ(通信モジュール割り込み使用時)] r_csi10_callback_receiveend 関数内に CS=H 処理および通信終了処理を追加

```
static void r_csi10_callback_receiveend(void)
{
    /* Start user code. Do not edit comment generated here */
    R_SAIC_SPI_CSDisable(E_CSI10);
    R_CSI10_Stop();
    /* End user code. Do not edit comment generated here */
}
```

- [SPI のみ(通信モジュール割り込み使用時)] r_csi10_callback_error 関数内にフラグ更新処理、CS=H 処理および通信終了処理を追加

```
static void r_csi10_callback_error(uint8_t err_type)
{
    /* Start user code. Do not edit comment generated here */
    g_csi_overrun_flag |= gs_bit_tbl[E_CSI10];
    R_SAIC_SPI_CSDisable( E_CSI10 );
    R_CSI10_Stop();
    /* End user code. Do not edit comment generated here */
}
```

- [UART のみ] r_uart1_callback_sendend 関数内にフラグ更新処理を追加

```
static void r_uart1_callback_sendend(void)
{
    /* Start user code. Do not edit comment generated here */
    g_uart_tx_end_flag |= gs_bit_tbl[E_UART1];

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

- [UART のみ] API から呼び出される R_UART1_SettingChange 関数の追加

```
void R_UART1_SettingChange(uint8_t setting)
{
    e_uart_setting_t uart_setting = (e_uart_setting_t)setting;
    switch (uart_setting)
    {
        /*   4800bps, Parity=None */
        case E_UART_4800bps_None:
            SPS0 = 0x0055U;             /* シリアル・クロック選択レジスタ */
            SDR02 = 0x9A00U;            /* ボーレート設定                 */
            SDR03 = 0x9A00U;            /* ボーレート設定                 */
            SCR02 &= ~0x0300U;          /* パリティ設定                   */
            SCR03 &= ~0x0300U;          /* パリティ設定                   */
            break;
        /*   4800bps, Parity=Odd  */
        case E_UART_4800bps_Odd:
            SPS0 = 0x0055U;             /* シリアル・クロック選択レジスタ */
            SDR02 = 0x9A00U;            /* ボーレート設定                 */
            SDR03 = 0x9A00U;            /* ボーレート設定                 */
            SCR02 |= 0x0300U;           /* パリティ設定                   */
            SCR03 |= 0x0300U;           /* パリティ設定                   */
            break;
        /*   4800bps, Parity=Even  */
        case E_UART_4800bps_Even:
            SPS0 = 0x0055U;             /* シリアル・クロック選択レジスタ */
            SDR02 = 0x9A00U;            /* ボーレート設定                 */
            SDR03 = 0x9A00U;            /* ボーレート設定                 */
            SCR02 &= ~0x0300U;          /* パリティ設定                   */
            SCR03 &= ~0x0300U;          /* パリティ設定                   */
            SCR02 |= 0x0200U;           /* パリティ設定                   */
            SCR03 |= 0x0200U;           /* パリティ設定                   */
            break;
        /* 250000bps, Parity=None */
        case E_UART_250Kbps_None:
            SPS0 = 0x0000U;             /* シリアル・クロック選択レジスタ */
            SDR02 = 0x5E00U;            /* ボーレート設定                 */
            SDR03 = 0x5E00U;            /* ボーレート設定                 */
            SCR02 &= ~0x0300U;          /* パリティ設定                   */
            SCR03 &= ~0x0300U;          /* パリティ設定                   */
            break;
```

```
        /* 250000bps, Parity=Odd  */
        case E_UART_250Kbps_Odd:
            SPS0 = 0x0000U;             /* シリアル・クロック選択レジスタ */
            SDR02 = 0x5E00U;            /* ボーレート設定                 */
            SDR03 = 0x5E00U;            /* ボーレート設定                 */
            SCR02 |= 0x0300U;           /* パリティ設定                   */
            SCR03 |= 0x0300U;           /* パリティ設定                   */
            break;
        /* 250000bps, Parity=Even  */
        case E_UART_250Kbps_Even:
            SPS0 = 0x0000U;             /* シリアル・クロック選択レジスタ */
            SDR02 = 0x5E00U;            /* ボーレート設定                 */
            SDR03 = 0x5E00U;            /* ボーレート設定                 */
            SCR02 &= ~0x0300U;          /* パリティ設定                   */
            SCR03 &= ~0x0300U;          /* パリティ設定                   */
            SCR02 |= 0x0200U;           /* パリティ設定                   */
            SCR03 |= 0x0200U;           /* パリティ設定                   */

            break;
    }
}
```

- [UART のみ] API から呼び出される R_UART1_GetHeader 関数の追加

```
/***********************************************************************
* Function Name: R_UART1_GetHeader
* Description  : This function returns the process header data received
by the UART1.
* Arguments    : uint8_t *packet_data -
*                    Header data
*              : uint8_t rx_buffer[] -
*                    Receive buffer
*              : uint16_t read_pos -
*                    Buffer read position
* Global Value : g_uart1_rx_count
*                    Number of received data in the UART1
* SFR          : None
* Return Value : uint8_t -
*                    0=Invalid, 1=Valid
***********************************************************************/
uint8_t R_UART1_GetHeader( uint8_t *packet_data, uint8_t rx_buffer[],
uint16_t read_pos )
{
    uint8_t ret = 0U;

    if (read_pos < g_uart1_rx_count)
    {
        *packet_data = rx_buffer[read_pos];
        ret = 1;
    }

    return (ret);
}
```

- [UART のみ] API から呼び出される R_UART1_Getdata 関数の追加

```
/***********************************************************************
* Function Name: R_UART1_Getdata
* Description  : This function check of bytes of data that is not less
than the number specified has been received.
* Arguments    : uint16_t rx_cnt -
*                 number of bytes specified data
* Global Value : g_uart1_rx_count
*                 Number of received data in the UART1
* SFR          : None
* Return Value : uint8_t -
*                 0=Invalid, 1=Valid
***********************************************************************/
uint8_t R_UART1_Getdata(uint16_t rx_cnt)
{
    uint8_t ret = 0U;

    if (rx_cnt <= g_uart1_rx_count)
    {
        ret = 1U;
    }

    return (ret);
}
```

- [SPI のみ(ポーリング使用時)] API から呼び出される R_CSI10_MaskStart 関数の追加

```
/***********************************************************************
* Function Name: R_CSI10_MaskStart
* Description  : This function starts the CSI10 module operation.
* Arguments    : None
* Return Value : None
***********************************************************************/
void R_CSI10_MaskStart(void)
{
    SO0 |= _0400_SAU_CH2_CLOCK_OUTPUT_1;/* CSI10 clock initial level */
    SO0 &= ~_0004_SAU_CH2_DATA_OUTPUT_1;    /* CSI10 SO initial level */
    SOE0 |= _0004_SAU_CH2_OUTPUT_ENABLE;    /* enable CSI10 output */
    SS0 |= _0004_SAU_CH2_START_TRG_ON;     /* enable CSI10 */
    CSIIF10 = 0U;    /* clear INTCSI10 interrupt flag */
    CSIMK10 = 1U;    /* disable INTCSI10 interrupt */
}
```

― r_ cg_sau.h 修正内容一覧

- [UART のみ] r_cg_sau.c 内に追加した関数を API から参照できるように extern 宣言を追加

```
extern void    R_UART1_SettingChange(uint8_t setting);
extern uint8_t R_UART1_GetHeader(uint8_t *packet_data, uint8_t
rx_buffer[], uint16_t read_pos);
extern uint8_t R_UART1_Getdata(uint16_t rx_cnt);
```

- [SPI のみ] r_cg_sau.c 内に追加した変数を API から参照できるように extern 宣言を追加

```
extern uint8_t g_csi_overrun_flag;
```

- [SPI のみ(ポーリング使用時)] r_cg_sau.c 内に追加した関数を API から参照できるように extern 宣言を追加

```
extern void R_CSI10_MaskStart(void);
```

― r_sa_uart_control_register.h 修正内容一覧

- [SPI のみ] 通信モジュール割り込み使用/ポーリング使用設定。どちらか片方をコメントアウトして使用すること。(UART はポーリング設定不可のため割り込み使用固定とする。)

```
#define D_SPI_OPERATION D_SPI_USE_INTERRUPT        /* 通信モジュール 割り込み使用定義    */
//#define D_SPI_OPERATION D_SPI_REGISTER_POLLING     /* 通信モジュール 割り込み不使用定義 */
```

- [UART のみ] UART ネゴシエーション関数処理項目設定。必要なもの以外コメントアウトすること。

```
#define D_UART_NEGOTIATION_250KBPS_PARITY_ODD      /* UART baudrate=250000bps, Parity=Odd  */
#define D_UART_NEGOTIATION_250KBPS_PARITY_EVEN     /* UART baudrate=250000bps, Parity=Even */
#define D_UART_NEGOTIATION_250KBPS_PARITY_NONE     /* UART baudrate=250000bps, Parity=None */
#define D_UART_NEGOTIATION_4800BPS_PARITY_ODD      /* UART baudrate=4800bps, Parity=Odd    */
#define D_UART_NEGOTIATION_4800BPS_PARITY_EVEN     /* UART baudrate=4800bps, Parity=Even   */
#define D_UART_NEGOTIATION_4800BPS_PARITY_NONE     /* UART baudrate=4800bps, Parity=None   */
```

- [UART/SPI 共通] FLASH 関連 API の有効/無効設定。無効設定時、コメントアウトすること。

```
#define D_SAIC_FLASH_API_VALID
```

- [UART/SPI 共通] SAIC101 との通信 Wait 時にデッドロック判断するループ回数の設定。

```
/* deadlock count to prevent deadlock */
#define D_DEADLOCK_CNT      (11000000L)
```

- [UART のみ] UART ch の定義設定。MCU の UART モジュールチャネル数に合わせて修正。

```
typedef enum
{
    E_UART0 = 0x00U,                  /* UART0 */
    E_UART1,                         /* UART1 */
    E_UART2,                         /* UART2 */
    E_UART3,                         /* UART3 */
    E_UART_MAX

} e_uart_ch_t;
```

- [SPI のみ] CSI ch の定義設定。MCU の SPI モジュールチャネル数に合わせて修正。

```
typedef enum
{
    E_CSI00 = 0x00U,                /* CSI00 */
    E_CSI01,                        /* CSI01 */
    E_CSI10,                        /* CSI10 */
    E_CSI11,                        /* CSI11 */
    E_CSI20,                        /* CSI20 */
    E_CSI21,                        /* CSI21 */
    E_CSI30,                        /* CSI30 */
    E_CSI31,                        /* CSI31 */
    E_CSI_MAX

} e_csi_ch_t;
```

— r_sa_uart_control_register_user.c 修正内容一覧

- [UART/SPI 共通] コード生成シリアルファイルのインクルード設定。コード生成ツール出力ファイル名が異なる場合、修正。

```
#include "r_cg_sau.h"
```

- [UART/SPI 共通] CPU CLK の設定(MHz)。MCU の設定に応じて修正。

```
#define D_CPU_CLK_MHZ                ( 24.0F)     /* 動作クロック(MHz) */
```

- [UART/SPI 共通] パワーオンリセット時間の設定(ms)。必要に応じて時間修正。

```
#define D_WAIT_PON_RST_TIME_MS       ( 4.00F)    /* WAIT 時間(ms)      */
```

- [UATR/SPI 共通] パワーオンリセット時間の算出設定

```
#define D_PON_RST_NOP_CNT
((uint32_t)((D_WAIT_PON_RST_TIME_MS/(1.0F/D_CPU_CLK_MHZ))*1000.0F/7.0F))
```

- [UART のみ] SAIC 情報格納グローバル変数指定。変数の配列番号は各 API で使用する SAIC 番号に対応。接続チャネルと接続 SAIC を r_sa_uart_control_register.h で設定した ENUM 値で記載すること。

```
const uart_saic_t g_uart_saic_data_tbl[] =
{
//  { UART_ch,   sa_type,       }, /* format              */
    { E_UART1,   E_SAIC101,     }, /* SAIC 番号=0 の SAIC 情報 */

};/* SAIC 情報格納グローバル変数 */
```

- [SPI のみ] SAIC 情報格納グローバル変数指定。変数の配列番号は各 API で使用する SAIC 番号に対応。接続チャネル、接続 SAIC は r_sa_spi_control_register.h で設定した ENUM 値で記載し、CS 端子のアドレスとビット、INT 端子のアドレスとビットは以下を参考に記載すること。INT 端子不使用時は INT 端子のアドレスに NULL を記載すること。

```
const spi_saic_t g_spi_saic_data_tbl[] =
{
//  { csi_ch,  sa_type,  p_cs_addr, cs_bit_num, p_int_addr, int_bit_num, }, /* format         */
    { E_CSI10, E_SAIC101, &P0,       6U,         &P0,       7U,       }, /* SAIC 番号=0 の SAIC 情報 */

}; /* SAIC 情報格納グローバル変数 */
```

- [UART のみ] シリアルモジュール情報格納グローバル変数指定。変数の配列番号は SAIC 接続チャネルに対応。該当する関数名を記載する。

```
const uart_serial_t g_uart_serial_data_tbl[] =
{
#if (D_UART_OPERATION==D_UART_USE_INTERRUPT)
//  { R_UARTx_Start, R_UARTx_Stop, R_UARTx_Receive, R_UARTx_Send, R_UARTx_GetHeader, R_UARTx_Getdata, R_UARTx_SettingChange, },
    { NULL,         NULL,        NULL,           NULL,        NULL,            NULL,          NULL,               },
    { R_UART1_Start, R_UART1_Stop, R_UART1_Receive, R_UART1_Send, R_UART1_GetHeader, R_UART1_Getdata, R_UART1_SettingChange, },
    { NULL,         NULL,        NULL,           NULL,        NULL,            NULL,          NULL,               },
    { NULL,         NULL,        NULL,           NULL,        NULL,            NULL,          NULL,               },

#elif D_UART_OPERATION==D_UART_REGISTER_POLLING
    /* 未対応 */

#endif

}; /*シリアルモジュール情報格納グローバル変数 */
```

- [SPI のみ] シリアルモジュール情報格納グローバル変数指定。変数の配列番号は SAIC 接続チャネルに対応。通信モジュール割り込み使用には該当する関数名を記載すること。ポーリング使用時には以下を参考に通信用レジスタの各アドレスとビット、該当する関数名を記載すること。

```
const spi_serial_t g_spi_serial_data_tbl[] =
{
#if D_SPI_OPERATION==D_SPI_USE_INTERRUPT
//  { CSI_Start,    CSI_Stop,    CSI_Send_Receive,    }, /* format  */
    { NULL,         NULL,        NULL,              }, /* CSI00   */
    { NULL,         NULL,        NULL,              }, /* CSI01   */
    { R_CSI10_Start, R_CSI10_Stop, R_CSI10_Send_Receive, }, /* CSI10   */
    { NULL,         NULL,        NULL,              }, /* CSI11   */
    { NULL,         NULL,        NULL,              }, /* CSI20   */
    { NULL,         NULL,        NULL,              }, /* CSI21   */
    { NULL,         NULL,        NULL,              }, /* CSI30   */
    { NULL,         NULL,        NULL,              }, /* CSI31   */

#elif D_SPI_OPERATION==D_SPI_REGISTER_POLLING
    { NULL,             NULL,   NULL,  0U, NULL,             NULL,             NULL,            NULL,        }, /* CSI00    */
    { NULL,             NULL,   NULL,  0U, NULL,             NULL,             NULL,            NULL,        }, /* CSI01    */
    { (uint16_t *)&SMR02,&SIO10,&IF1L,1U,(uint16_t*)&SSR02,(uint16_t*)&SIR02,R_CSI10_MaskStart,R_CSI10_Stop,}, /* CSI10    */
    { NULL,             NULL,   NULL,  0U, NULL,             NULL,             NULL,            NULL,        }, /* CSI11    */
    { NULL,             NULL,   NULL,  0U, NULL,             NULL,             NULL,            NULL,        }, /* CSI20    */
    { NULL,             NULL,   NULL,  0U, NULL,             NULL,             NULL,            NULL,        }, /* CSI21    */
    { NULL,             NULL,   NULL,  0U, NULL,             NULL,             NULL,            NULL,        }, /* CSI30    */
    { NULL,             NULL,   NULL,  0U, NULL,             NULL,             NULL,            NULL,        }, /* CSI31    */

#endif
}; /*シリアルモジュール情報格納グローバル変数 */
```

- [UART/SPI 共通] RESET 情報格納グローバル変数指定。リセット方法を指定。SAIC101 の場合、パワーオンリセットのみ設定可能。SAIC 番号には SAIC 情報格納グローバル変数の配列の要素番号を記載すること。

```
const uart_reset_t g_uart_reset_data_tbl[] =
{
    //process, Port address, Bit num, nop_cnt, uart_saic_t 番号,},
    { E_SAIC_POWERON_RESET, NULL,  0U, D_PON_RST_NOP_CNT, 0U, },

};/* RESET 情報格納グローバル変数 */
```

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
Rev.1.00	2014.09.01	---	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 未使用端子の処理
   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。
   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。
2. 電源投入時の処置
   【注意】電源投入時は，製品の状態は不定です。
   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。
   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。
   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. リザーブアドレス（予約領域）のアクセス禁止
   【注意】リザーブアドレス（予約領域）のアクセスを禁止します。
   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
4. クロックについて
   【注意】リセット時は、クロックが安定した後、リセットを解除してください。
   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。
   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
5. 製品間の相違について
   【注意】型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。
   同じグループのマイコンでも型名が違うと、内部ROM、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ輻射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

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。

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/

Colophon 3.0