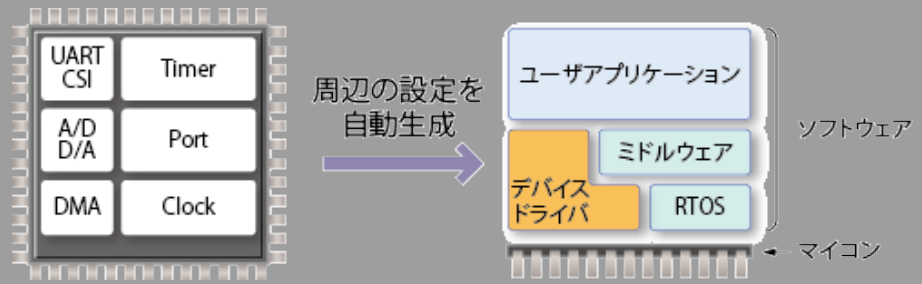
RL78/I1Eコード生成

RL78/I1E + 脈拍センサデモ
コンフィギュラブル・アンプ使用例

文書番号 R20UT3745JJ0110
ブロードベースソリューション事業部 ソフトウェア技術部
ルネサス エレクトロニクス株式会社

「コード生成」はクリック1 つで“かんたん”マイコン初期設定、開発工数を大幅削減する無償ツール。
CS+ , e² studio向けにプラグインを提供中。



2018.06.04

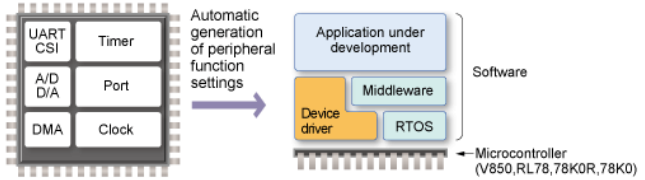
目次

コード生成概要 (5つの特長)

- わかりやすいGUIによる操作で、クロックを意識せず使用したい実際の値(タイマ周期、シリアルのボーレート、等)が入力可能です。

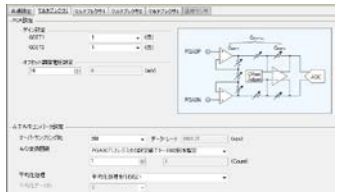
- 複数の周辺機能によるピン競合をチェックする機能、誤った設定値のチェック機能も装備しています。

- 周辺機能の制御プログラム(デバイスドライバ)だけでなく、メイン関数とAPI関数も生成します。(ルネサスRL78ファミリ用コンパイラCC-RLに対応済み)



- 設定した機能をファイル出力する充実したレポート機能

- マイコンに特化した周辺機能(LCD,アナログ系)のサポート



コード生成概要 (RLファミリ グループ別対応ツール一覧)

コード生成支援ツール	シリーズ	グループ
コード生成プラグイン (CS+, e² studio) Applilet3 for RL78※1 AP4 for RL78※1	RL78/F1x	RL78/F12, RL78/F13, RL78/F14, RL78/F15
	RL78/G1x	RL78/G10, RL78/G11, RL78/G12, RL78/G13, RL78/G14, RL78/G1A, RL78/G1C, RL78/G1D, RL78/G1E, RL78/G1F, RL78/G1G, RL78/G1H
	RL78/I1x	RL78/I1A, RL78/I1B, RL78/I1D, RL78/I1E
	RL78/L1x	RL78/L12, RL78/L13, RL78/L1C
Applilet3 for RL78※1	RL78/D1x	RL78/D1A

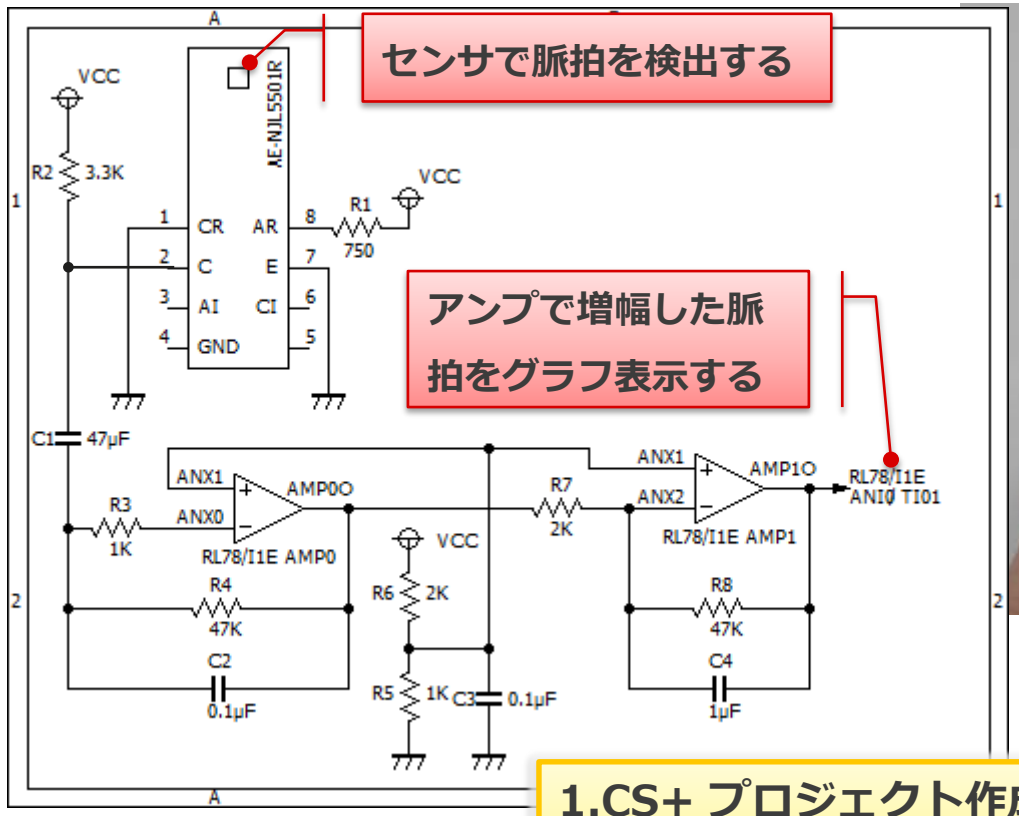
※1 AP4および Applilet は、スタンドアロンツールです。

製品情報の詳細は、以下のURLをご参照ください。
コード生成プラグイン　http://japan.renesas.com/cg_p
AP4, Applilet　http://japan.renesas.com/applilet

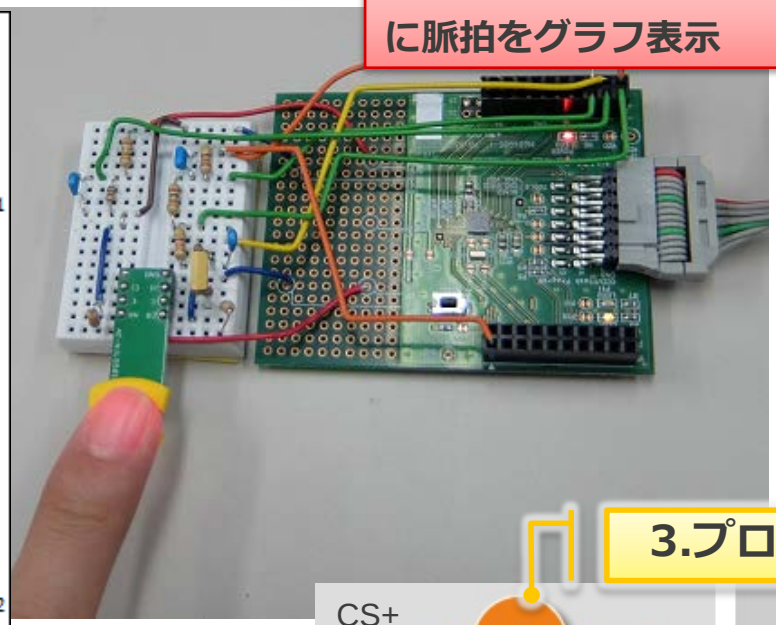
汎用アンプを使ったデモ概要

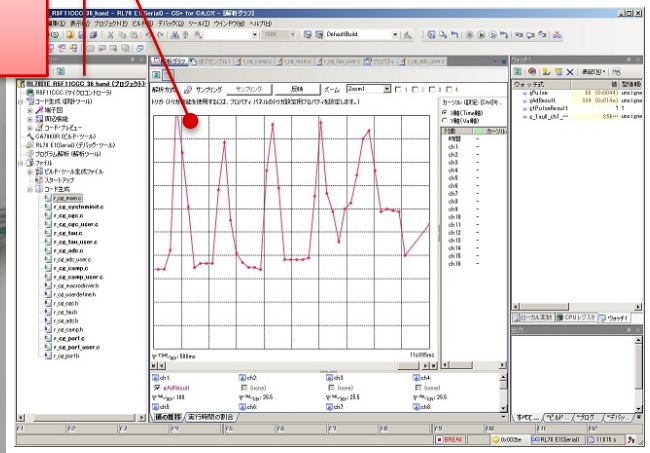
センサで脈拍を検出する

アンプで増幅した脈拍をグラフ表示する



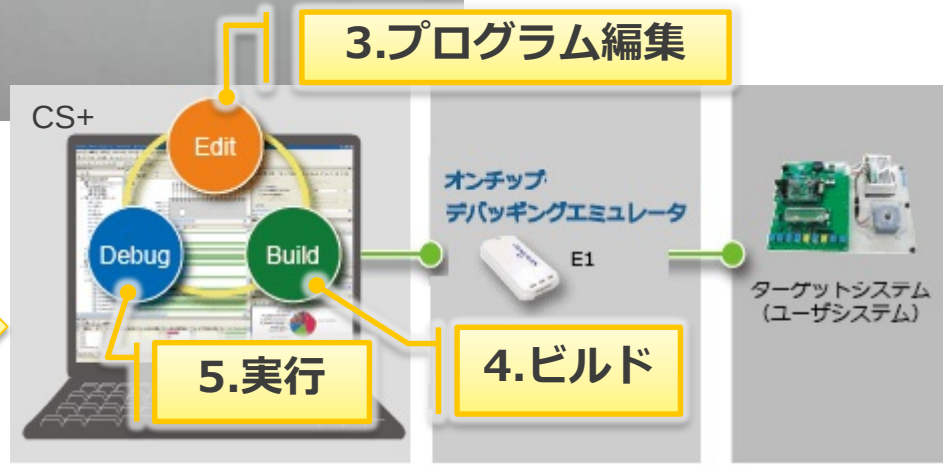
CS+上でリアルタイムに脈拍をグラフ表示





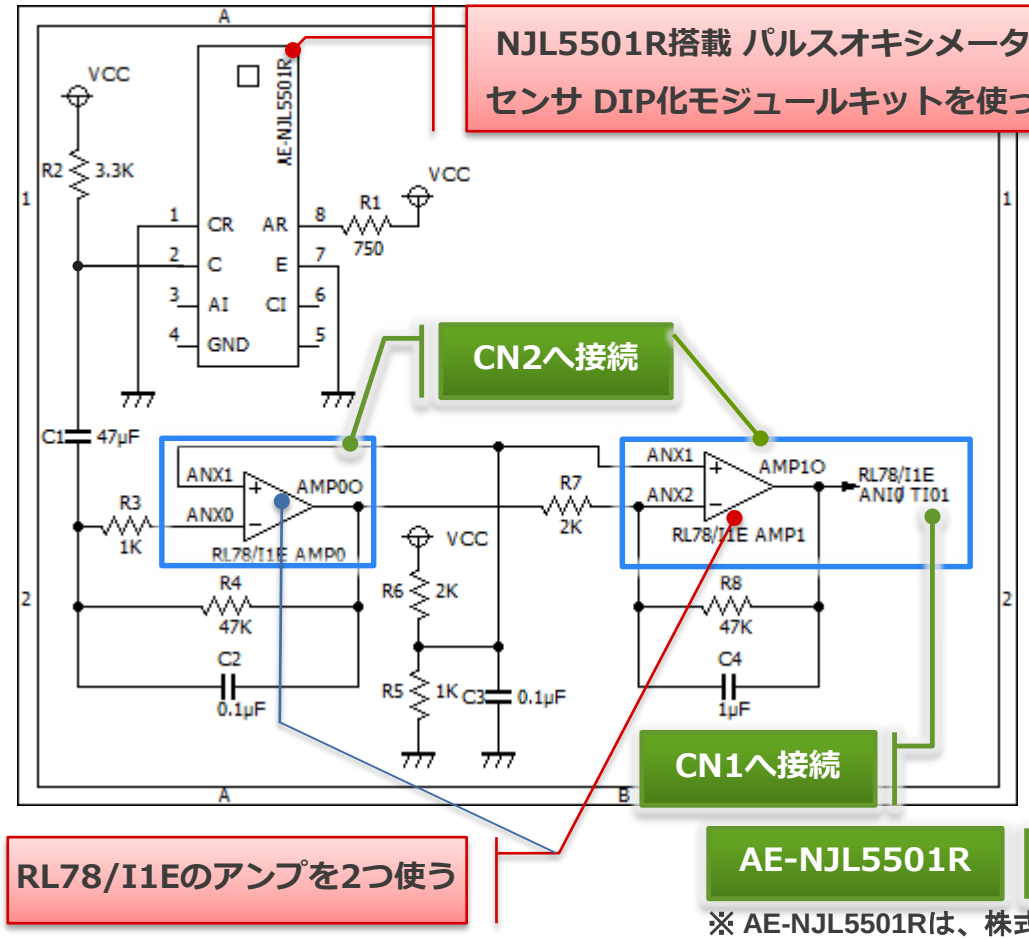
1.CS+ プロジェクト作成

2.コード生成で周辺機能設定



汎用アンプを使ったデモ概要 (デモ全景)

NJL5501R搭載 パルスオキシメータ用・反射型
センサ DIP化モジュールキットを使って脈拍計測

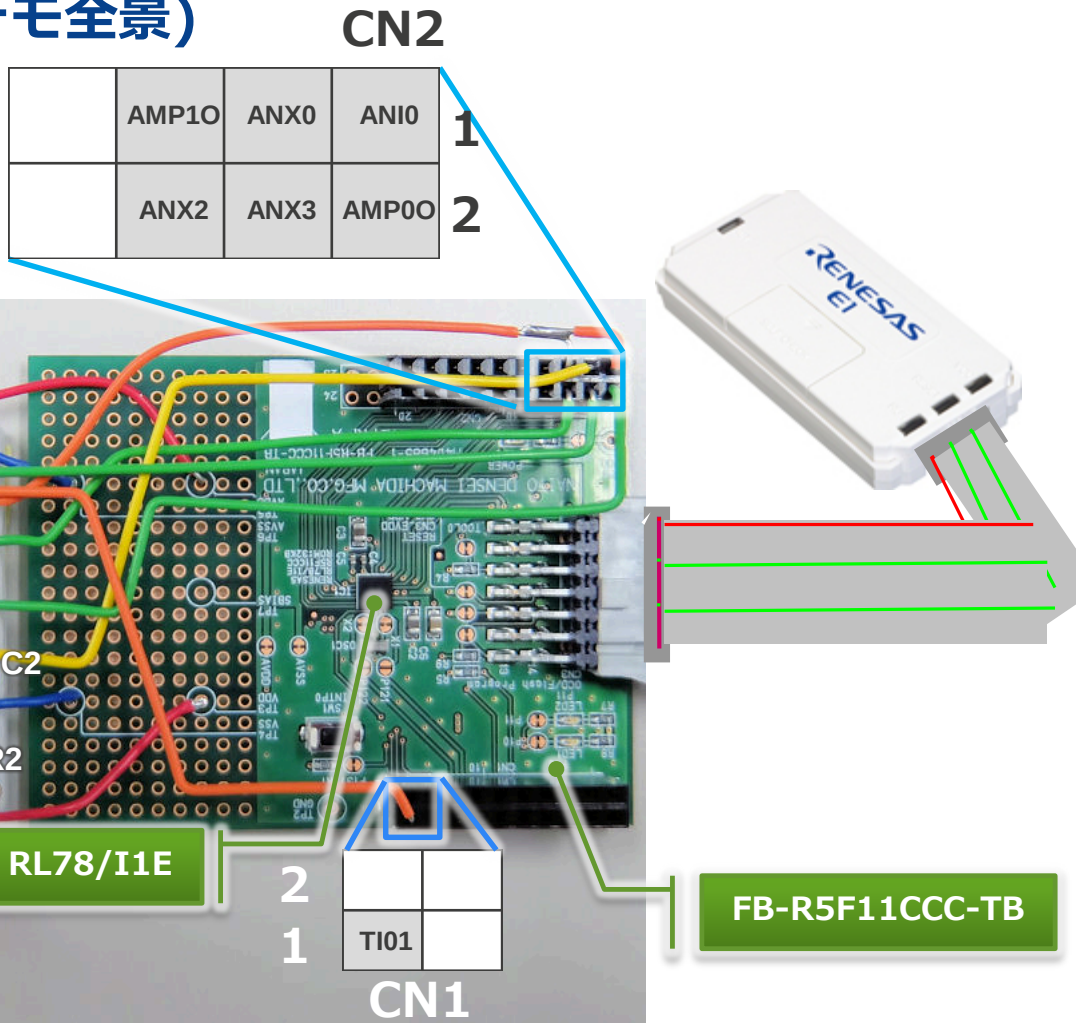


CN2へ接続

CN1へ接続

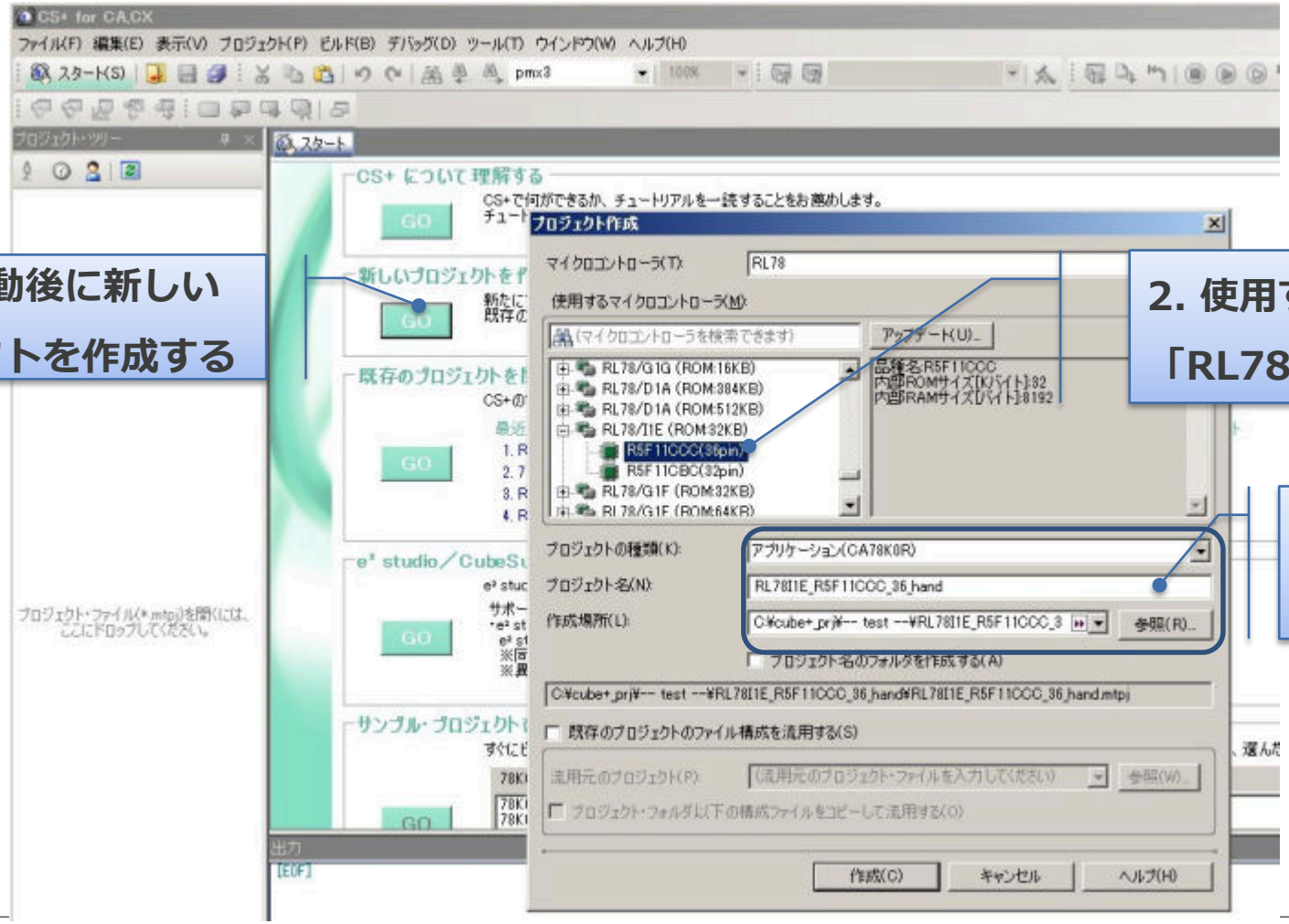
RL78/I1Eのアンプを2つ使う

AE-NJL5501R



※ AE-NJL5501Rは、株式会社秋月電子通商様の製品です
http://akizukidenshi.com/catalog/g/gK-09433/

CS+でプロジェクト作成



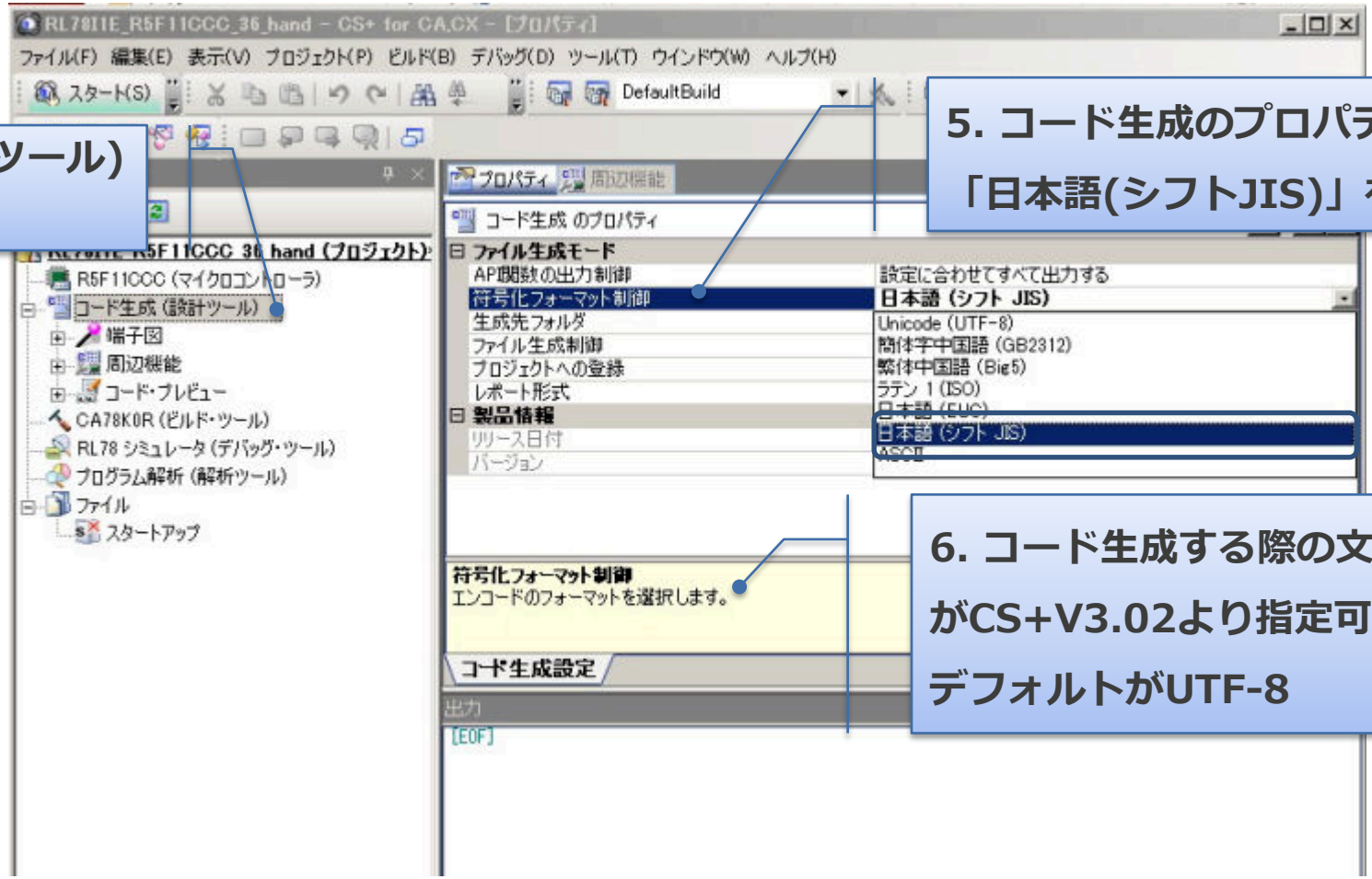
1. CS+起動後に新しいプロジェクトを作成する

2. 使用するマイクロコントローラ「RL78/I1E R5F11CCC(36pin)」を選択

3. プロジェクト名、作成場所を入力

※ 開発環境としてCS+ for CA, CXで作成していますが、CS+ for CCでも同じ手順で作成できます。

CS+でプロジェクト作成

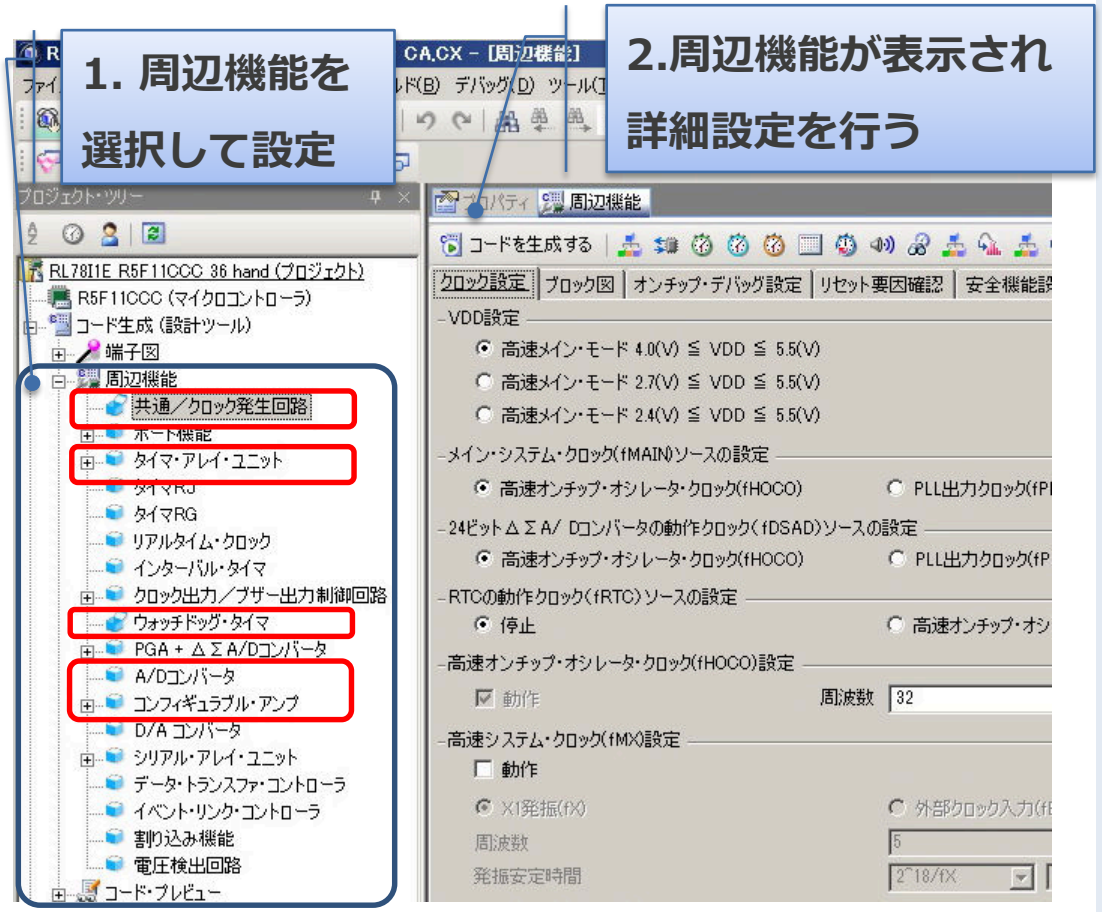


4. コード生成(設計ツール)のプロパティを開く

5. コード生成のプロパティを開き「日本語(シフトJIS)」を選択

6. コード生成する際の文字フォーマットがCS+V3.02より指定可能。CS+の場合、デフォルトがUTF-8

コード生成で周辺機能設定　(このデモで使用する周辺機能)



このデモで設定する周辺機能

・共通/クロック発生回路はデフォルトの設定

→ メイン・システム・クロック、高速オンチップ・オシレータ・クロック(fHOCO)

高速オンチップ・オシレータ(fHOCO) 32MHz

CPUと周辺クロック fHOCOを使用、 オンチップ・デバッグの設定

・タイマ・アレイ・ユニット

→ インターバルタイマ、入力パルス間隔の設定

・ウォッチドッグタイマ

→ デフォルトが「使用する」設定なので未使用とする

・A/Dコンバータ

→ アンプ出力をA/D変換する

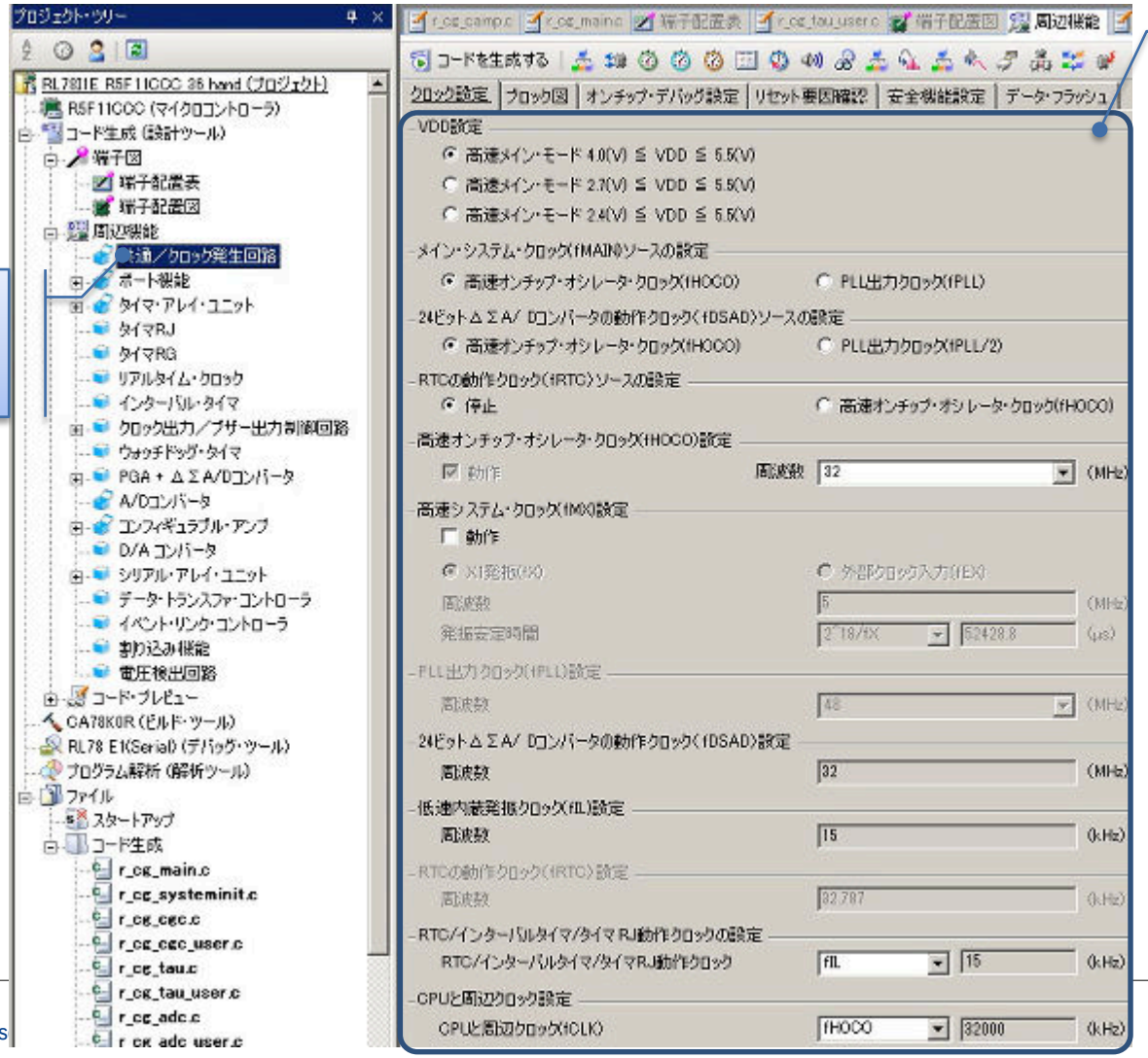
・コンフィギュラブル・アンプ

→ 汎用アンプx2として使う

コード生成で周辺機能設定

（ クロック発生回路はデフォルトで使用 ）

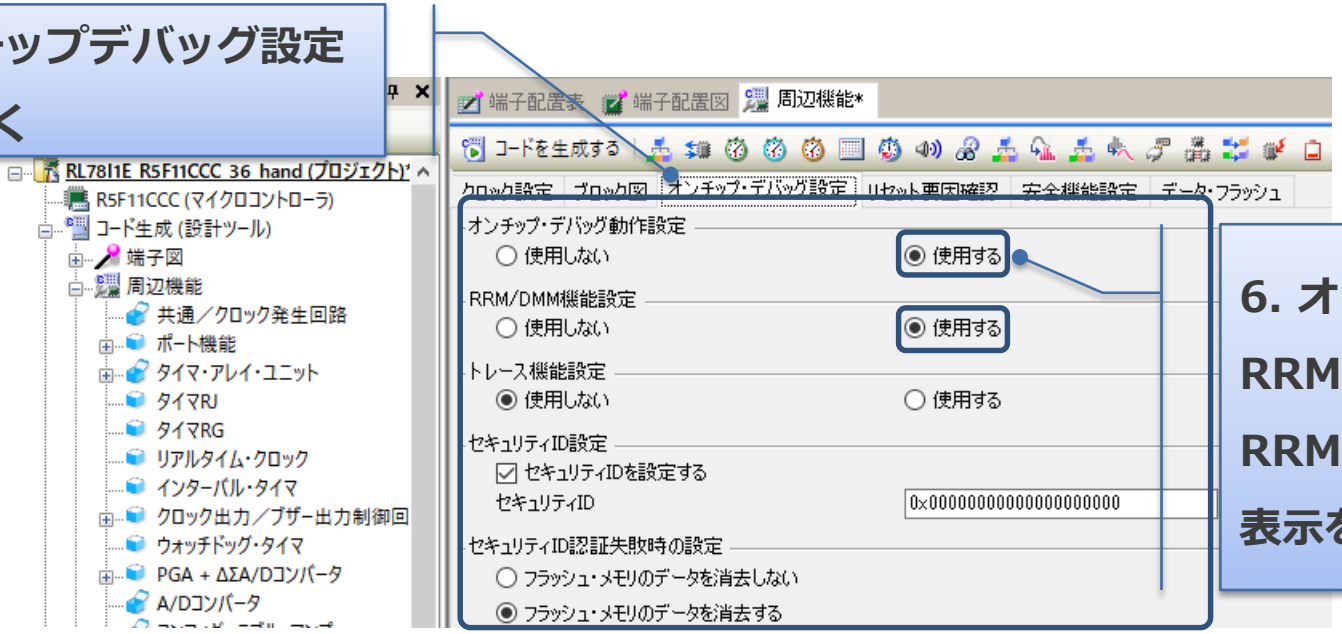
3.クロック発生回路をクリック



4.デフォルトで使用するので変更しない

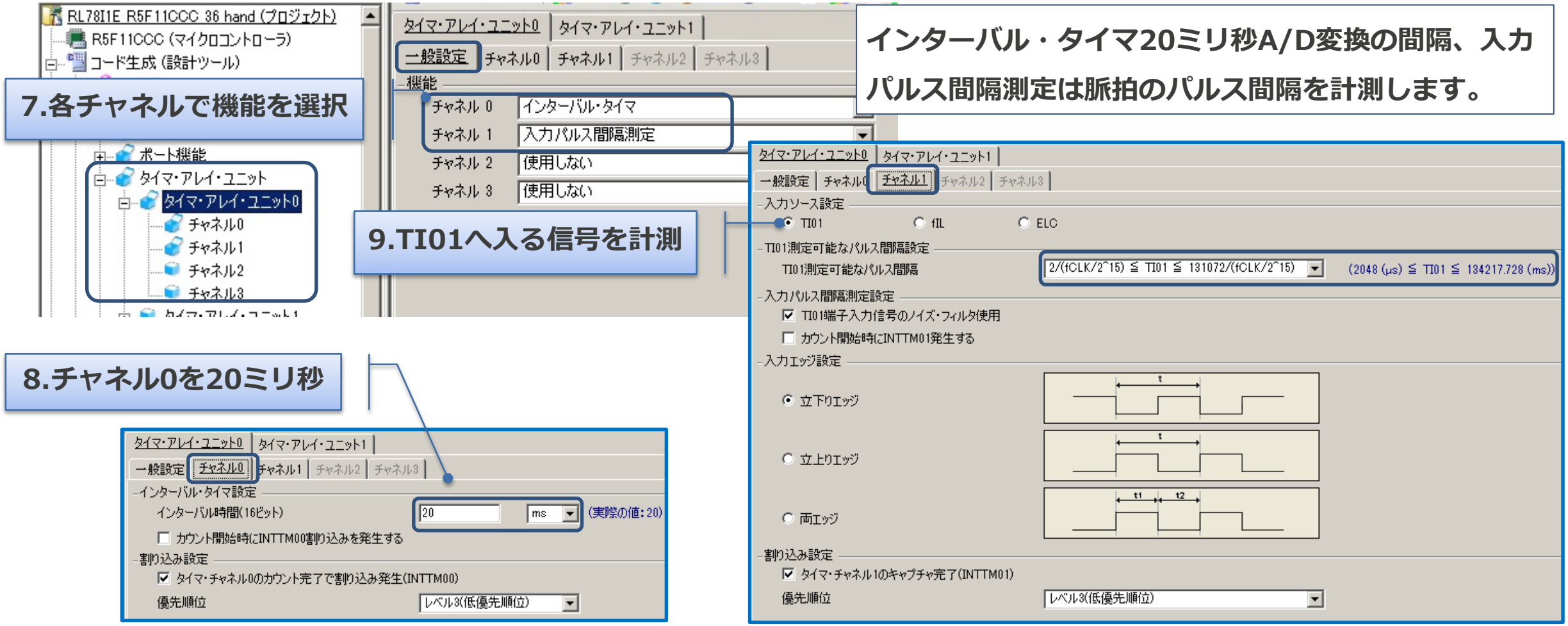
コード生成で周辺機能設定　（ オンチップ・デバッグ設定 ）

5. オンチップデバッグ設定タブを開く

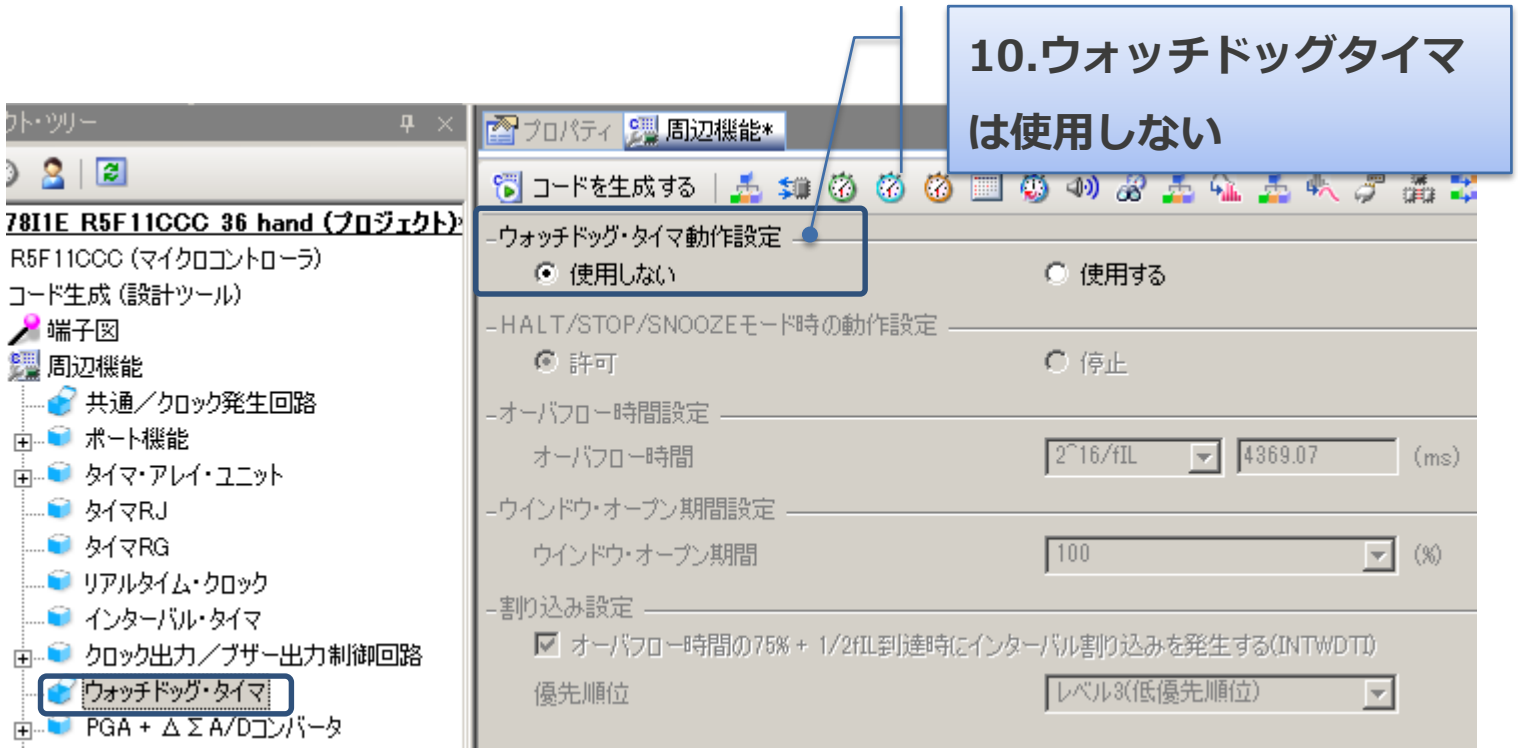


6. オンチップデバッグを使用する。
RRM/DMM機能を使用する。
RRMとはプログラム実行中に変数表示を行う機能のこと

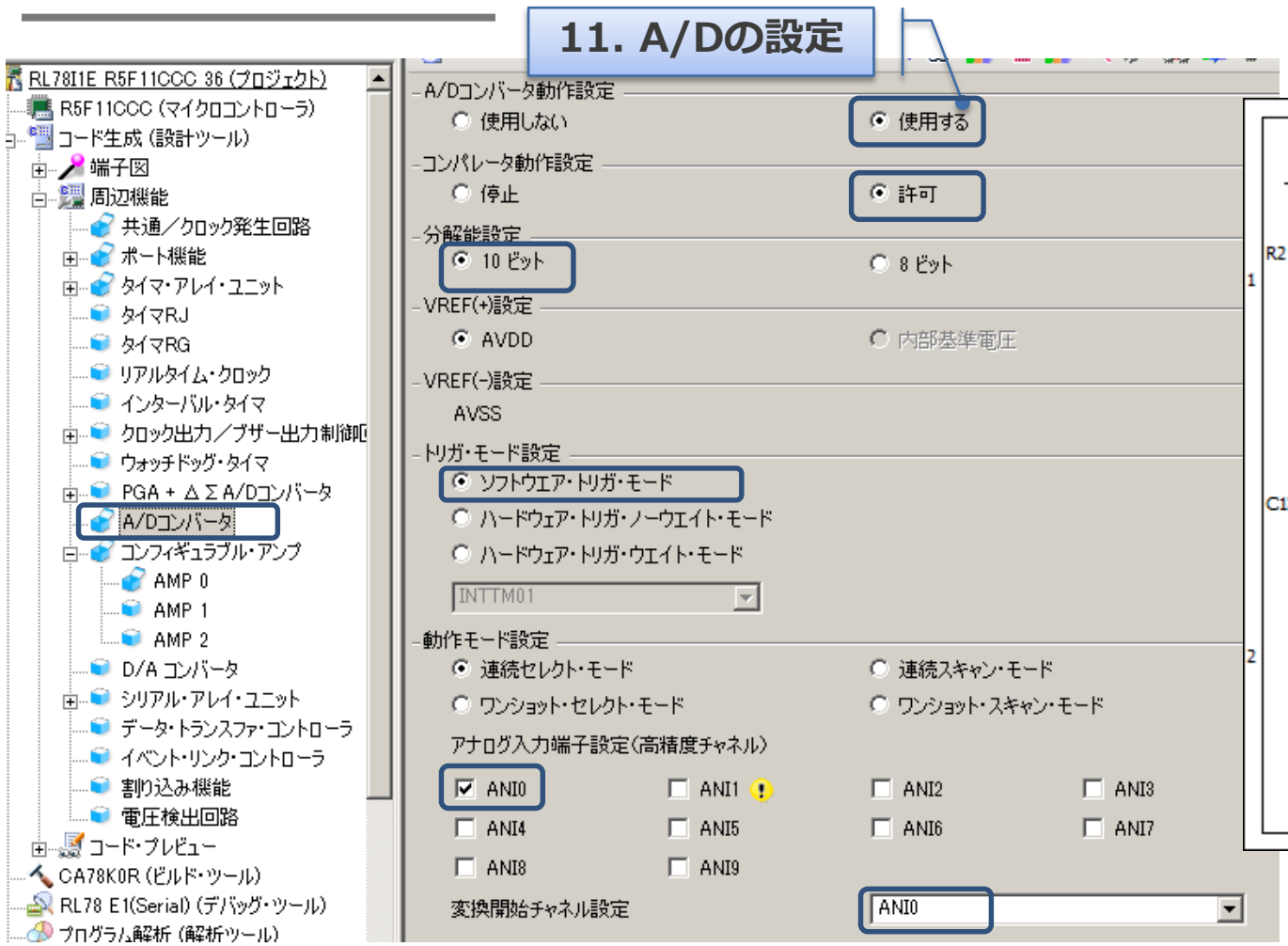
コード生成で周辺機能設定 （ タイマ・アレイ・ユニットの設定 ）

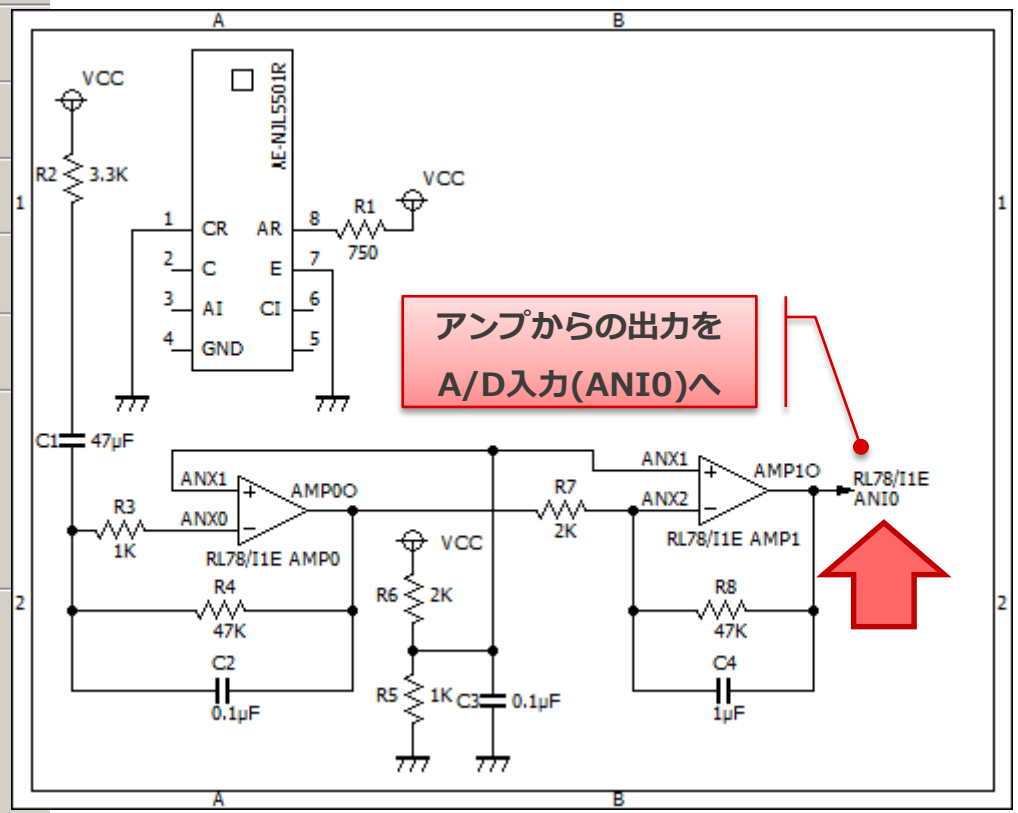


コード生成で周辺機能設定　（ ウォッチドッグタイマは未使用 ）

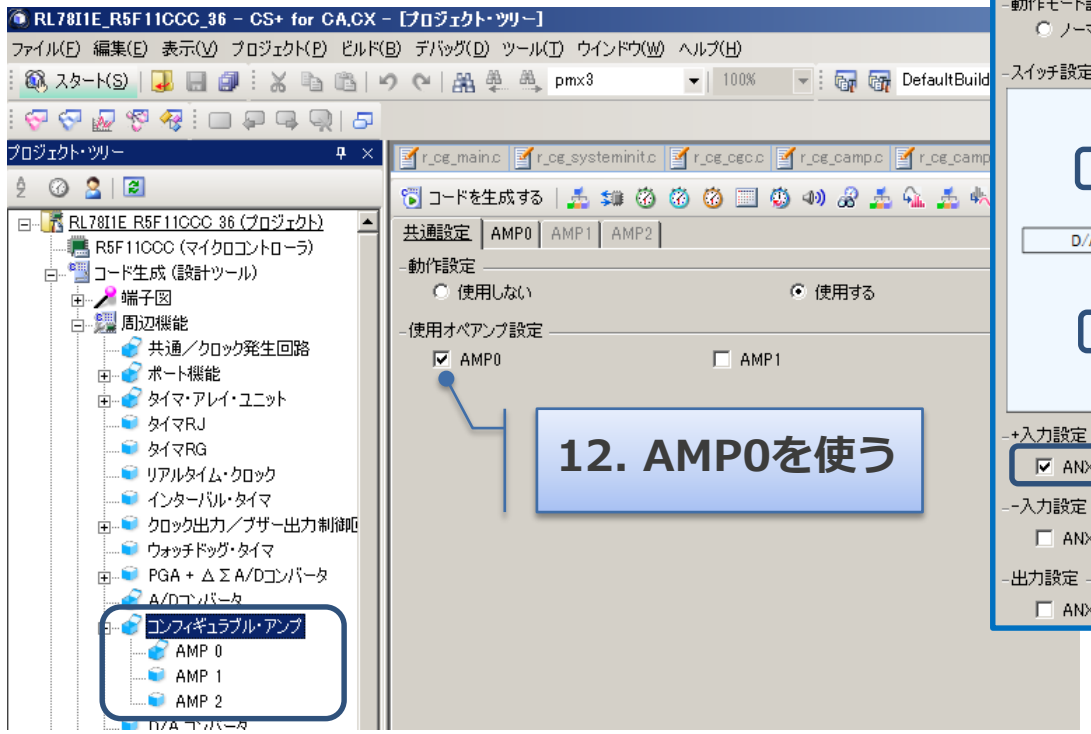


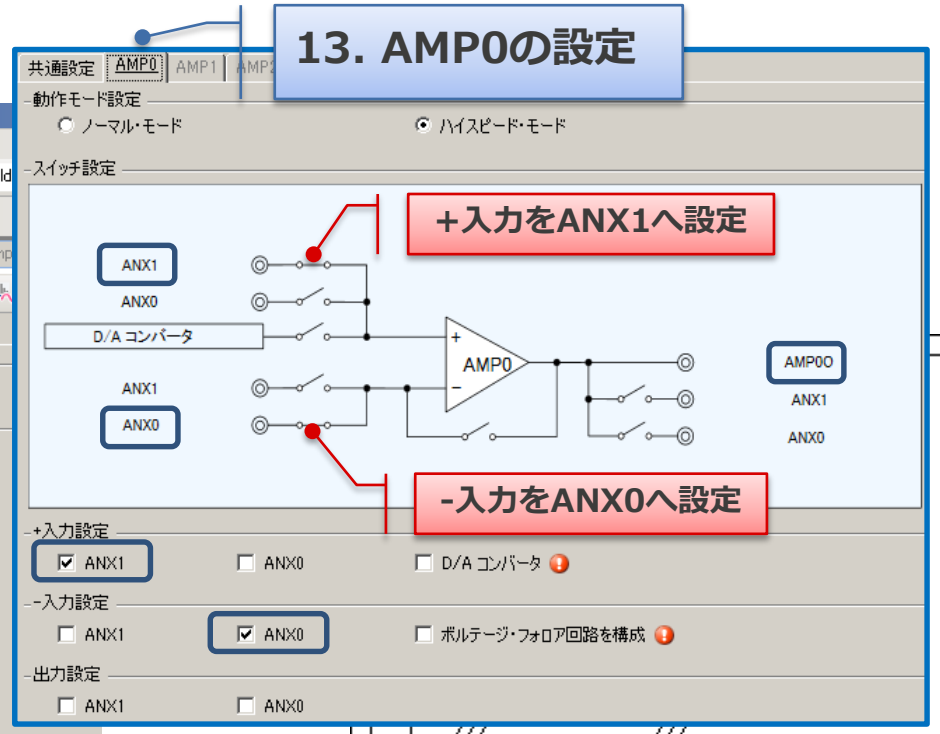
コード生成で周辺機能設定 （ A/Dコンバータの設定 ）

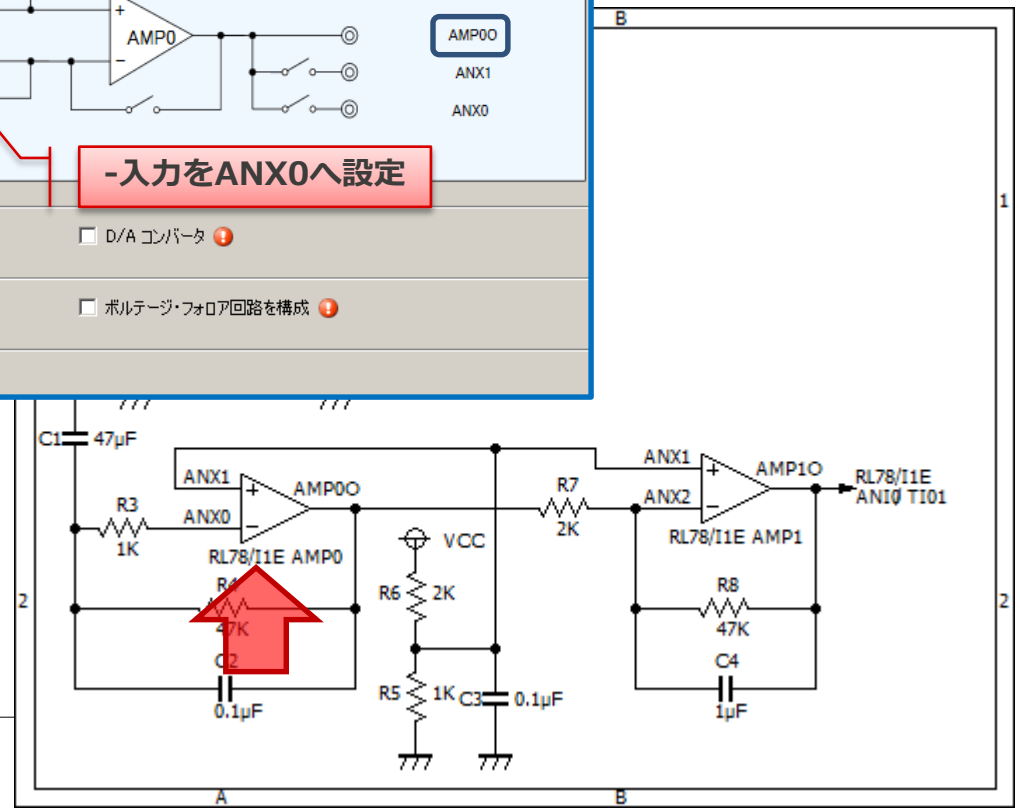




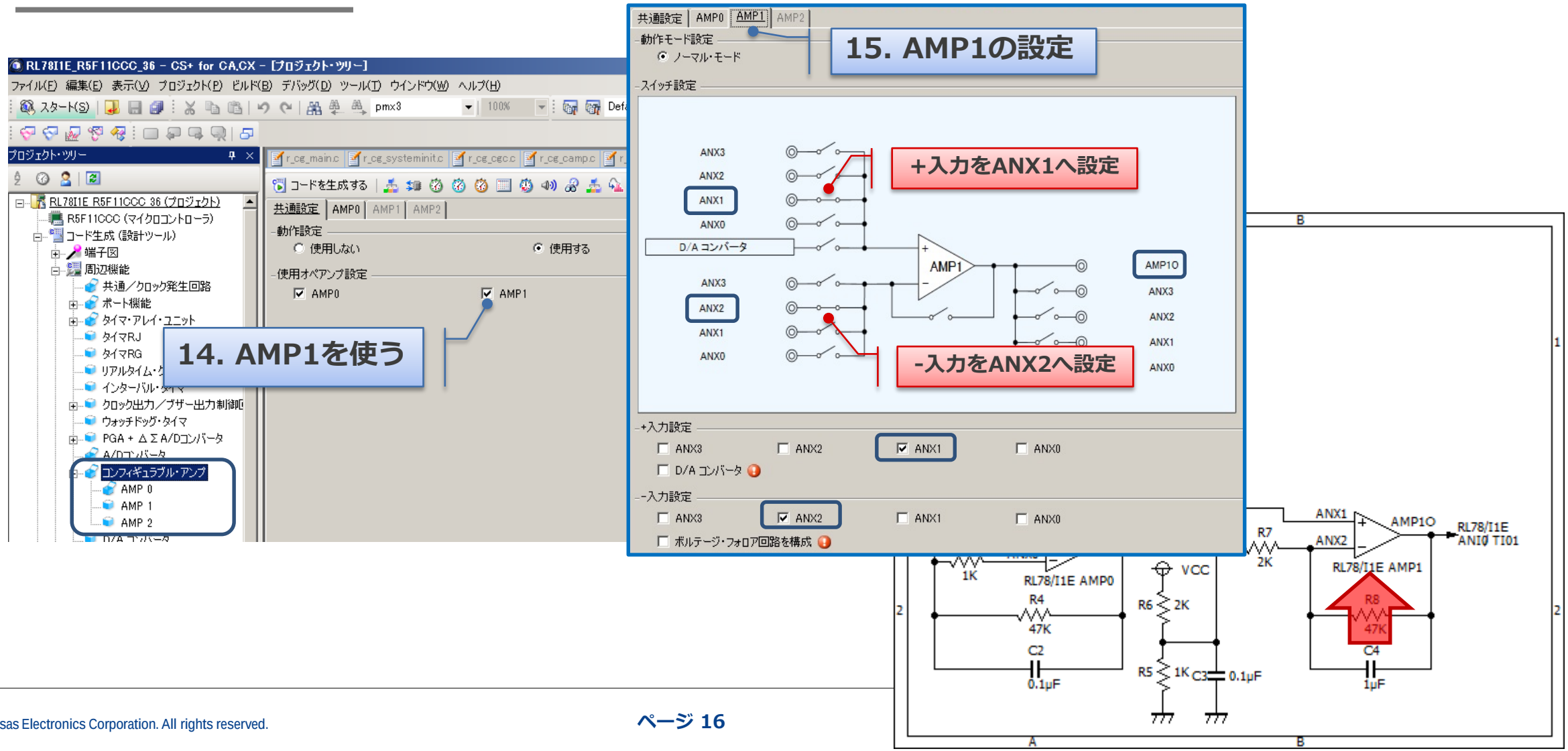
コード生成で周辺機能設定 （ コンフィギュラブル・アンプ0の設定 ）



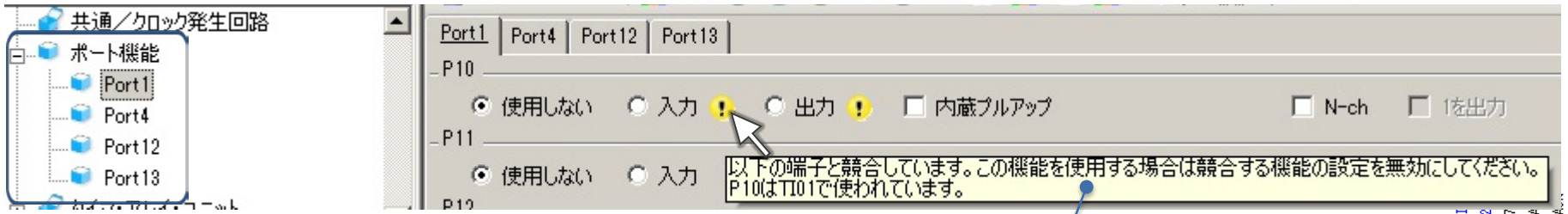




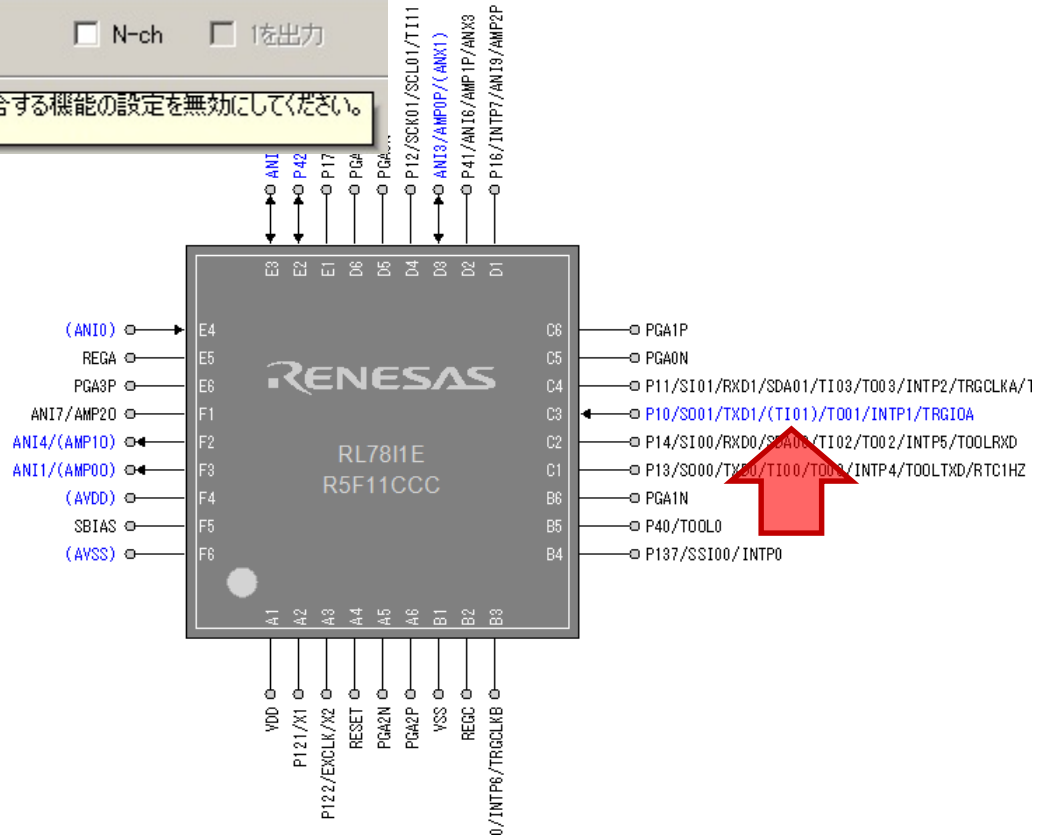
コード生成で周辺機能設定 （ コンフィギュラブル・アンプ1の設定 ）



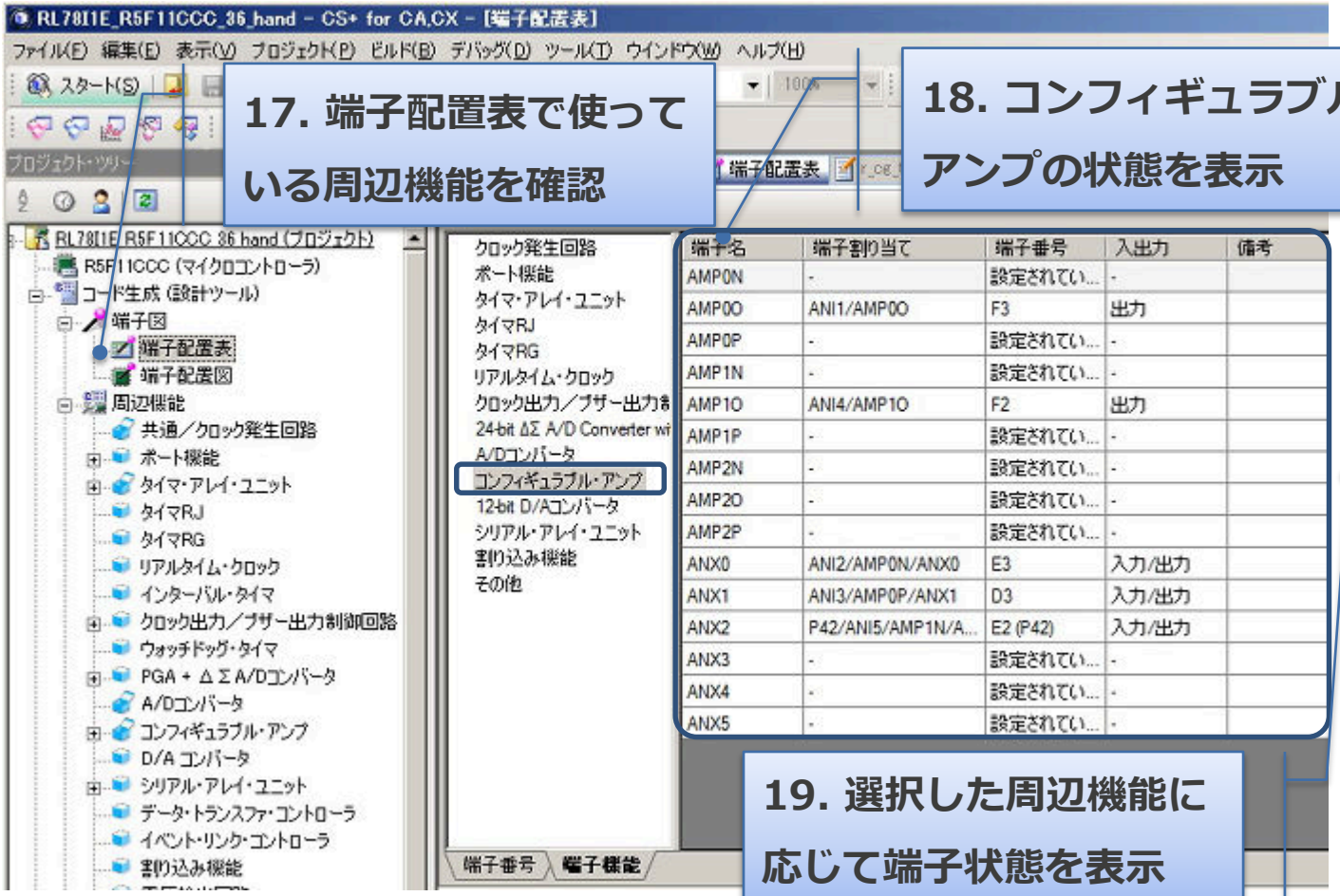
コード生成で周辺機能設定 （ ポート10の確認 ）



16. P10の「 ! 」にマウスカーソルを合わせると、TI01で使われていることが表示される。



コード生成で周辺機能設定 （ 端子配置表で確認 ）



17. 端子配置表で使っている周辺機能を確認

18. コンフィギュラブル・アンプの状態を表示

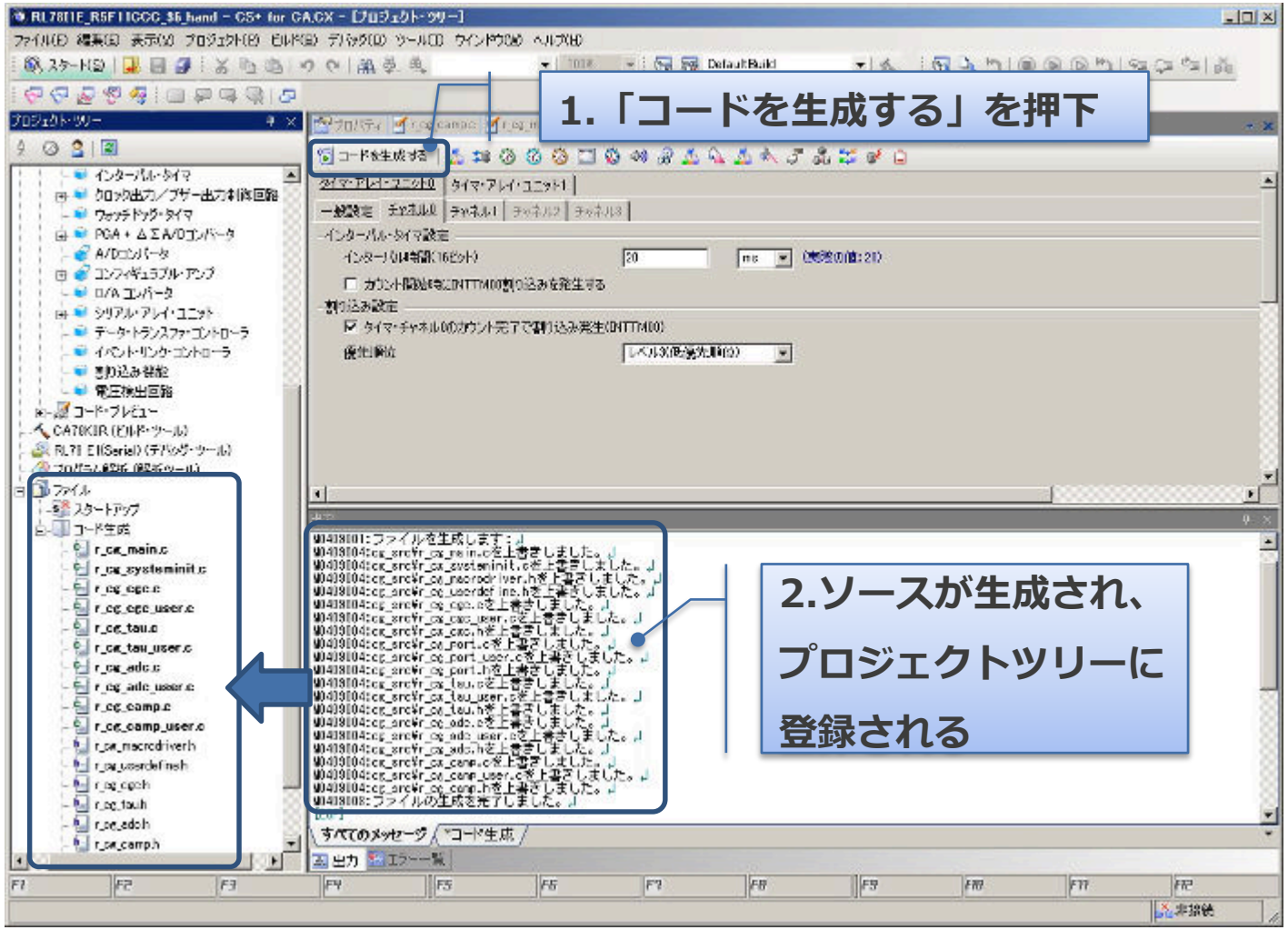
19. 選択した周辺機能に応じて端子状態を表示

20. A/Dコンバータの状態を表示

クロック発生回路	端子名	端子割り当て	端子番号	入出力
ポート機能	ANI0	ANI0	E4	入力
タイマ・アレイ・ユニット	ANI1	-	設定されてい...	-
タイマRJ	ANI2	-	設定されてい...	-
タイマRG	ANI3	-	設定されてい...	-
リアルタイム・クロック	ANI4	-	設定されてい...	-
クロック出力／ブザー出力制	ANI5	-	設定されてい...	-
24-bit ΔΣ A/D Converter wi	ANI6	-	設定されてい...	-
A/Dコンバータ	ANI7	-	設定されてい...	-
コンフィギュラブル・アンプ	ANI8	-	設定されてい...	-
12-bit D/Aコンバータ	ANI9	-	設定されてい...	-
シリアル・アレイ・ユニット	AVDD	AVDD	F4	-
割り込み機能	AVSS	AVSS	F6	-
その他				

現在の端子状態の表をエクセルで出力することも可能です。

ソースコードを自動生成（生成されるソースファイルの種類）



生成されるソースファイル

・周辺機能の初期化と制御APIを含むもの

r_cg_周辺機能.c / .h

・周辺機能の割り込み関数を含むもの

r_cg_周辺機能_user.c

・main()関数があるファイル

r_cg_main.c

・コード生成で使う変数型の定義など

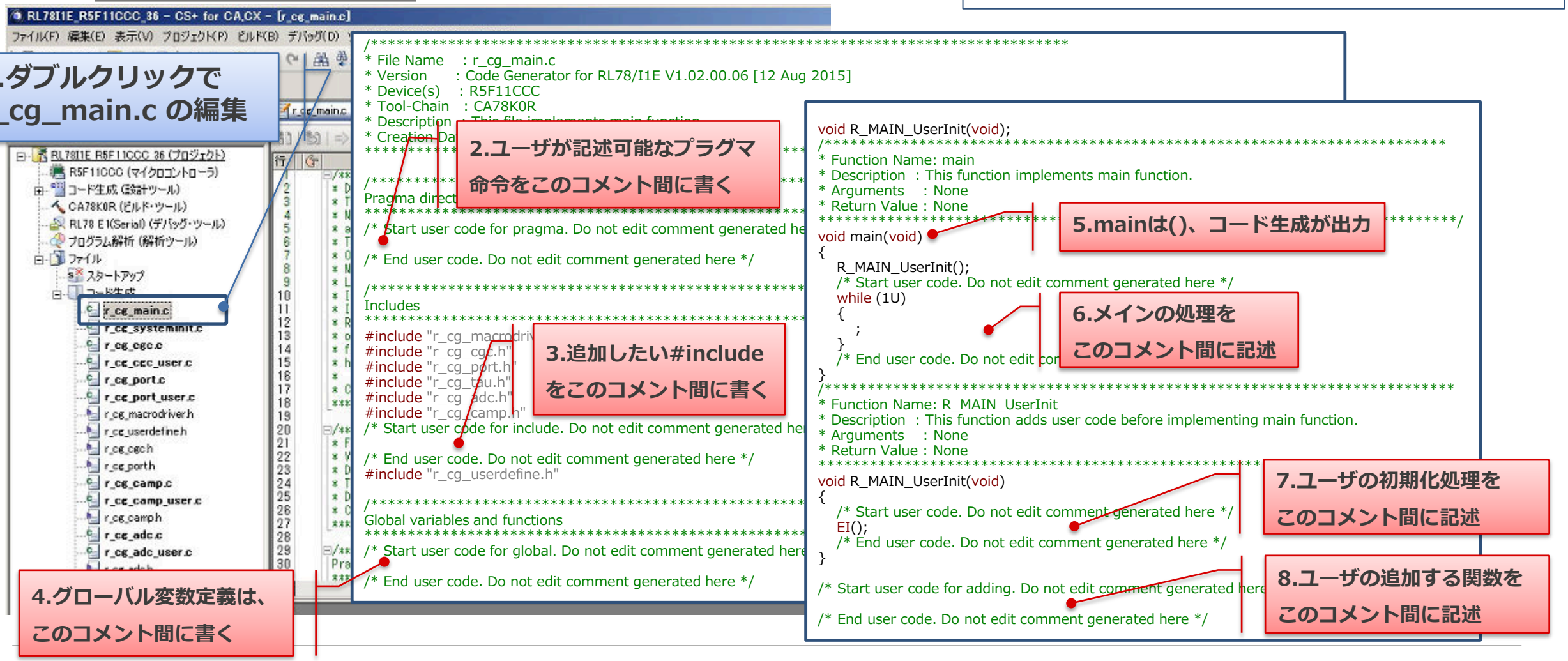
r_cg_macrodriver.h

・ユーザ用の共通定義するためのファイル

r_cg_userdefine.h

プログラム編集（ソースの記述方法）

指定コメント間にユーザコードを記述すれば、「コード生成」を実行しても、編集したユーザコードが消えることがありません。



プログラム編集 （r_cg_main.c）

プログラム追記部分

```
/***********************************************************
Global variables and functions
***********************************************************/
/* Start user code for global. Do not edit comment generated here */

volatile uint16_t gAdResult;                    // AD変換結果
volatile uint16_t gtAdResult[ D_PLUSEMAX ];     // 過去のAD変換結果を格納
volatile uint16_t gPulse;                       // 1分間の脈拍数

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

void R_MAIN_UserInit(void);
/***********************************************************
* Function Name: main
* Description  : This function implements main function.
* Arguments    : None
* Return Value : None
***********************************************************/
void main(void)
{
    R_MAIN_UserInit();
    /* Start user code. Do not edit comment generated here */
    while (1U)
    {
        ;
    }
    /* End user code. Do not edit comment generated here */
}
```

```
/***********************************************************
* Function Name: R_MAIN_UserInit
* Description  : This function adds user code before implementing main function.
* Arguments    : None
* Return Value :None
***********************************************************/
void R_MAIN_UserInit(void)
{
    /* Start user code. Do not edit comment generated here */
    R_CAMP0_Start();                    // アンプ０開始
    R_CAMP1_Start();                    // アンプ1開始
    R_TAU0_Channel0_Start();            // タイマチャネル０開始
    R_TAU0_Channel1_Start();            // タイマチャネル1開始
    EI();
    /* End user code. Do not edit comment generated here */
}
```

このページ以降のプログラム追記部分を記述して

ビルドすれば、デモプログラムが動作します。

プログラム編集 （r_cg_userdefine.h）

プログラム追記部分

```
/***********************************************************
* File Name    : r_cg_userdefine.h
* Version      : Code Generator for RL78/I1E V1.02.00.06 [12 Aug 2015]
* Device(s)    : R5F11CCC
* Tool-Chain   : CA78K0R
* Description  : This file includes user definition.
•  Creation Date: 2015/12/17
•  ***********************************************************/
#ifndef _USER_DEF_H
#define _USER_DEF_H

/***********************************************************************
***********************************************************
User definitions
***********************************************************************
***********************************************************/

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

#define D_PLUSEMAX      8    // ADの変換結果を格納する大きさ

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

プログラム編集 （r_cg_ad_user.c）

プログラム追記部分

```
/***********************************************************
Global variables and functions
***********************************************************/
/* Start user code for global. Do not edit comment generated here */

extern volatile uint16_t gAdResult;
extern volatile uint16_t gtAdResult[ D_PLUSEMAX ];
extern volatile uint16_t gPulse;

/* End user code. Do not edit comment generated here */
/***********************************************************
* Function Name: r_adc_interrupt
* Description  : None
* Arguments    : None
* Return Value : None
***********************************************************/
__interrupt static void r_adc_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    uint8_t i;

    R_ADC_Stop();
    R_ADC_Get_Result( &gAdResult );

    // 過去 D_PLUSEMAX 分のA/D変換結果を gtAdResult バッファへ格納する
    for ( i = (D_PLUSEMAX - 1); i > 0; i-- )
    {
        gtAdResult[ i ] = gtAdResult[ i - 1 ];
    }
    gtAdResult[ 0 ] = gAdResult;

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

A/D変換終了割り込み関数
(38μ秒ごとに変換が完了)

A/D変換を停止させているが、チャネル０タイマでA/D変換を20ミリ秒ごとに起動している。

A/D変換値の取得

defineで定義された個数分、A/D変換結果を
バッファへ代入

プログラム編集 （r_cg_tau_user.c）

プログラム追記部分

```
/***********************************************
Includes
***********************************************/
#include "r_cg_macrodriver.h"
#include "r_cg_tau.h"
/* Start user code for include. Do not edit
#include "r_cg_adc.h"
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"

/***************************
Global variables and functions
***************************
/* For TAU0_ch1 pulse measurement */
volatile uint32_t g_tau0_ch1_width = 0U;
/* Start user code for global. Do not edit comment generated here */

extern volatile uint16_t gPulse;
volatile uint32_t g32ul;

/* End user code. Do not edit comment generated here */
***********************************************
* Function Name: r_tau0_channel0_interrupt
* Description  : This function INTTM00 interrupt service
* Arguments    : None
* Return Value : None
***********************************************
__interrupt static void r_tau0_channel0_interrupt(void)
{
    /* Start user code. Do not edit comment generated here */
    R_ADC_Start();
    /* End user code. Do not edit comment generated here */
}
```

A/DのAPIを使うために追記

カウンタ値が820だとすれば、
820x(977/1)=0.839秒のパルスになる

20ミリ秒毎に呼ばれる
割り込み関数

A/D変換を開始

```
/***********************************
* Function Name: r_tau0_channel1_inte
* Description  : This function INTTM01 i
* Arguments    : None
* Return Value : None
***********************************************/
__interrupt static void r_tau0_channel1_interrupt(void)
{
    if (1U == (TSR01 & _0001_TAU_OVERFLOW_OCCURS))    /* overflow occurs
*/
    {
        g_tau0_ch1_width = (uint32_t)(TDR01 + 1U) + 0x10000U;
    }
    else
    {
        g_tau0_ch1_width = (uint32_t)(TDR01 + 1U);
    }
    /* Start user code. Do not edit comment generated here */

    // 60sec = 60000msec, g_tau0_ch1_width = 977Hz counter
    // TPS0 = _00F0_TAU_CKM1_FCLK_15 → fclk/2^15 → 977Hz (fclk=32MHz)
    g32ul = ( 60000ul * 100 ) / ( g_tau0_ch1_width * 102 );
    gPulse = (uint16_t)g32ul;

    /* End user code. Do not edit com
}

/* Start user cod
/* End user code
```

TI01に入力されたパルスの間隔を
計測する割り込み関数

パルス間隔を計測したカウント値を表す変数

一分間の脈拍数を計算

略号	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TPSm	0	0	PRS m31	PRS m30	0	0	PRS m21	PRS m20	PRS m13	PRS m12	PRS m11	PRS m10	PRS m03	PRS m02	PRS m01	PRS m00

PRS mk3	PRS mk2	PRS mk1	PRS mk0	動作クロック(CKmk)の選択注(k = 0, 1)					
					fCLK = 2 MHz	fCLK = 4 MHz	fCLK = 8 MHz	fCLK = 20 MHz	fCLK = 32 MHz
1	1	1	1	fCLK/2^15	61.0 Hz	122 Hz	244 Hz	610 Hz	977 Hz

TPSmレジスタで選択するクロックの波形は、立ち上がりからfCLKの1周期分だけハイ・レベルになります(m = 1-15)。

プログラムのワンポイント (計算は整数で行うと早い)

```
/***********************************************
* Function Name: r_tau0_channel1_interrupt
* Description  : This function INTTM01 i
* Arguments    : None
* Return Value : None
***********************************/
__interrupt static void r_tau0_channel1_interrupt(void)
{
    if (1U == (TSR01 & _0001_TAU_OVERFLOW_OCCURS))   /* overflow occurs */
    {
        g_tau0_ch1_width = (uint32_t)(TDR01 + 1U) + 0x10000U;
    }
    else
    {
        g_tau0_ch1_w                                  U);
    }
    /* Start user cod                                ated here */

    // 60sec = 60000msec, g_tau0_ch1_width = 977Hz counter
    // TPS0 = _00F0_TAU_CKM1_FCLK_15 → fclk/2^15 → 977Hz (fclk=32MHz)
    P4.3 = 1;
    g32ul = ( 60000ul * 100 ) / ( g_tau0_ch1_width * 102 );
    gPulse = (uint16_t)g32ul;
    P4.3 = 0;

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

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

r_cg_tau_user.c、入力パルス間隔測定の割り込み関数

P4.3を1にしている間の処理を波形で観測

処理が終了したのでP4.3を0にする

一分間の脈拍数を計算するため、下記のコードを使っています。

g32ul → unsigned long

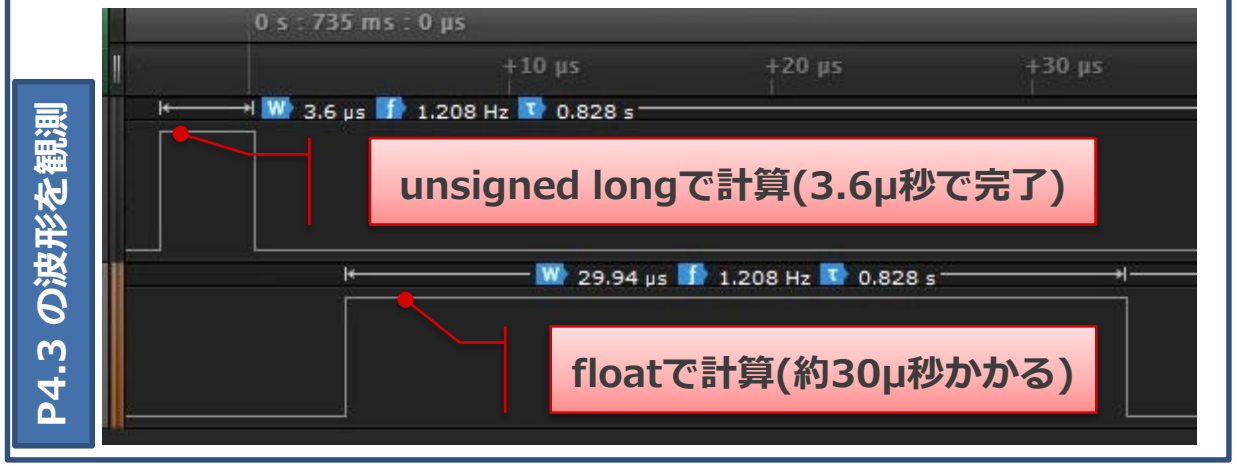
```
g32ul = ( 60000ul * 100 ) / ( g_tau0_ch1_width * 102 );
gPulse = (uint16_t)g32ul;
```

小数を使ったコードにすると下記のようになります。

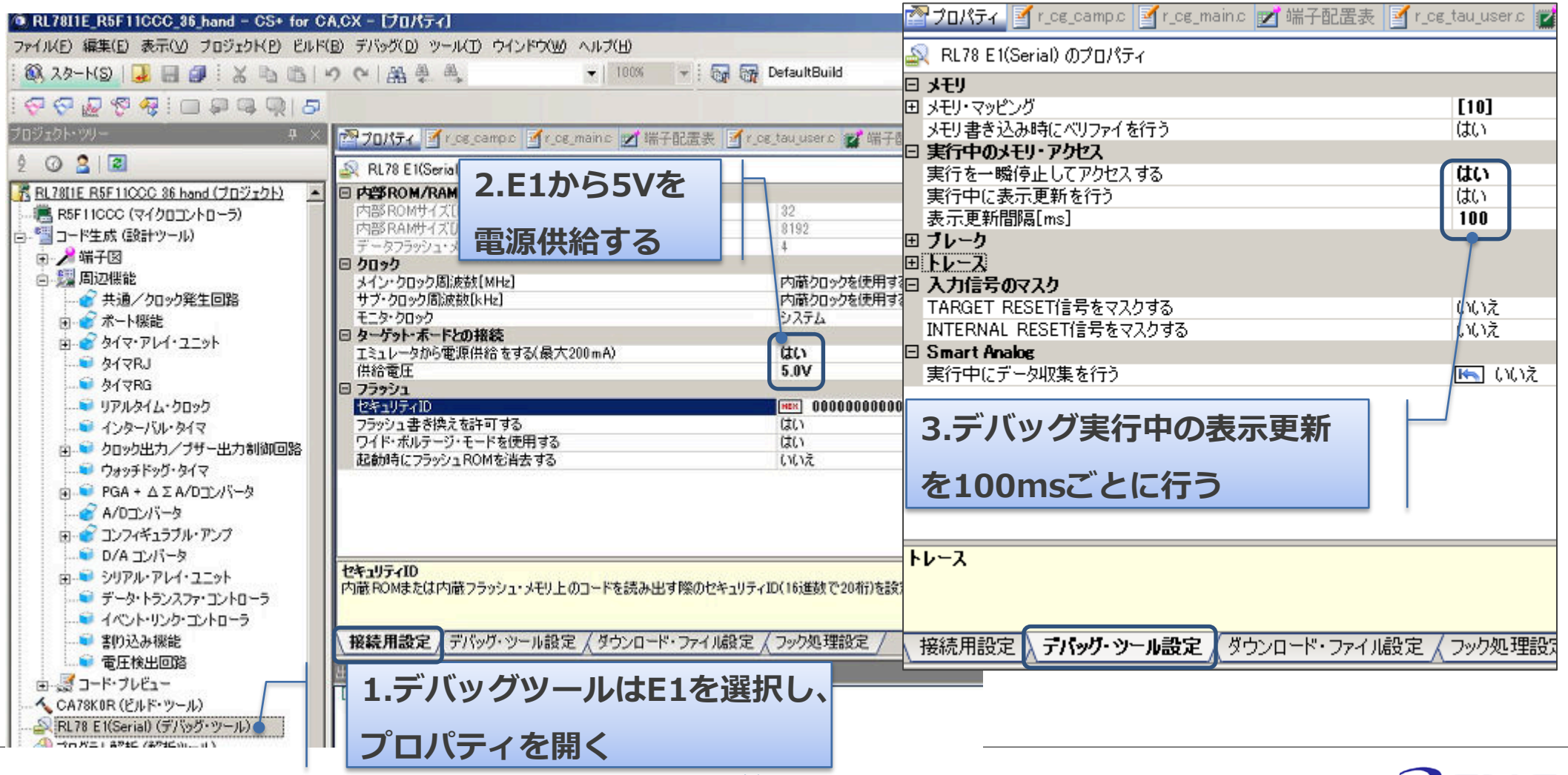
fl → float

```
fl  = 60 / ( 0.00102 * g_tau0_ch1_width );
gPulse = (uint16_t)fl;
```

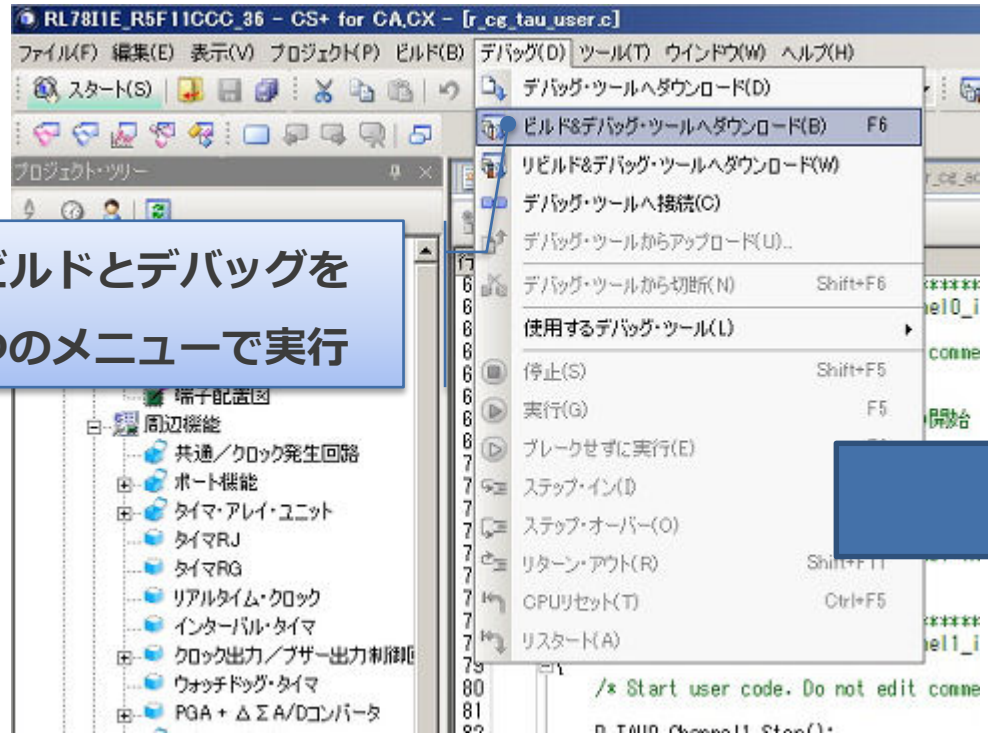
共に計算結果は同じですが、処理時間は8倍以上違います。下図は実際の処理時間。



デバッグツールの設定 (E1エミュレータの設定)

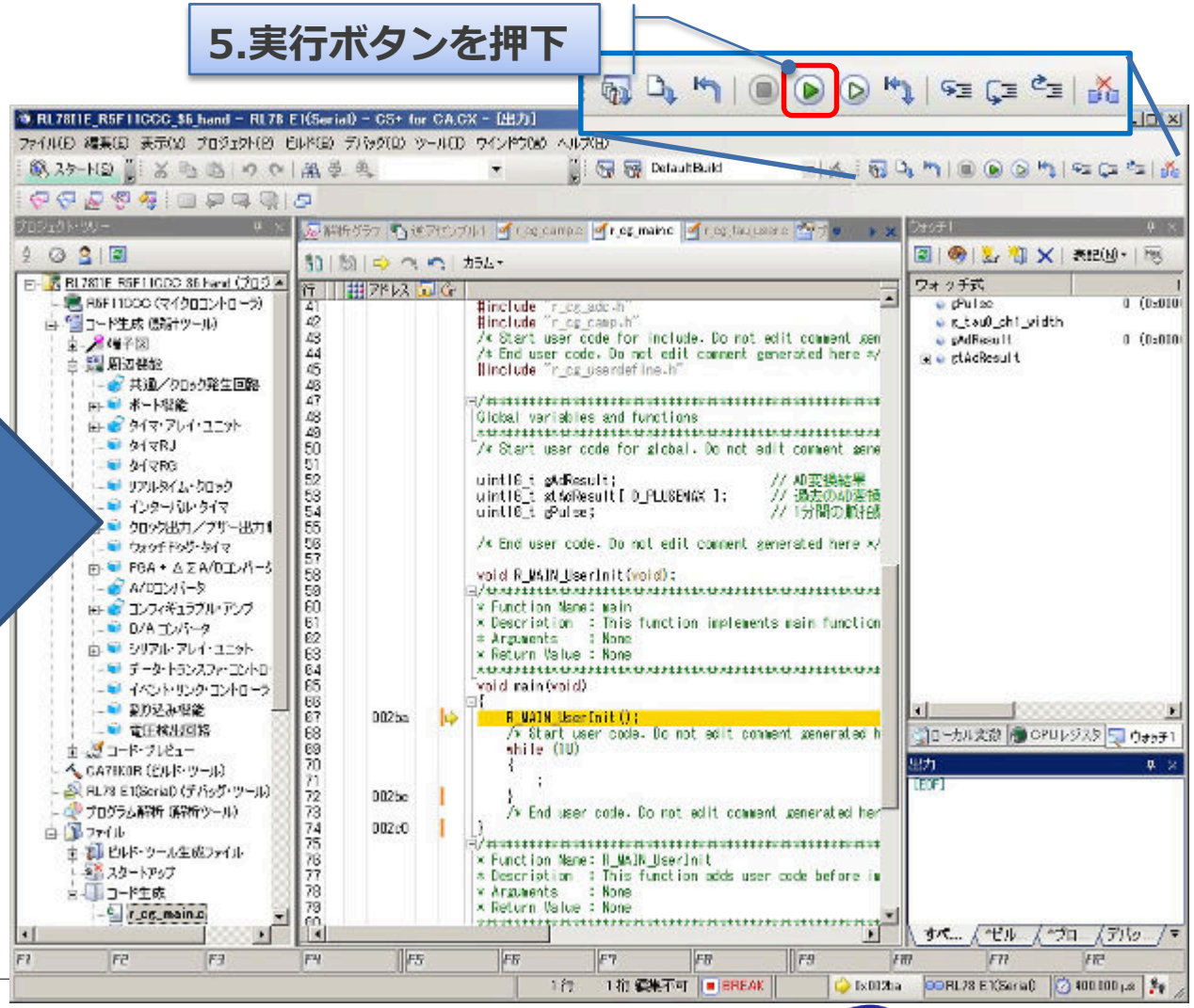


デバッグツールの設定 (ビルドしてダウンロード)

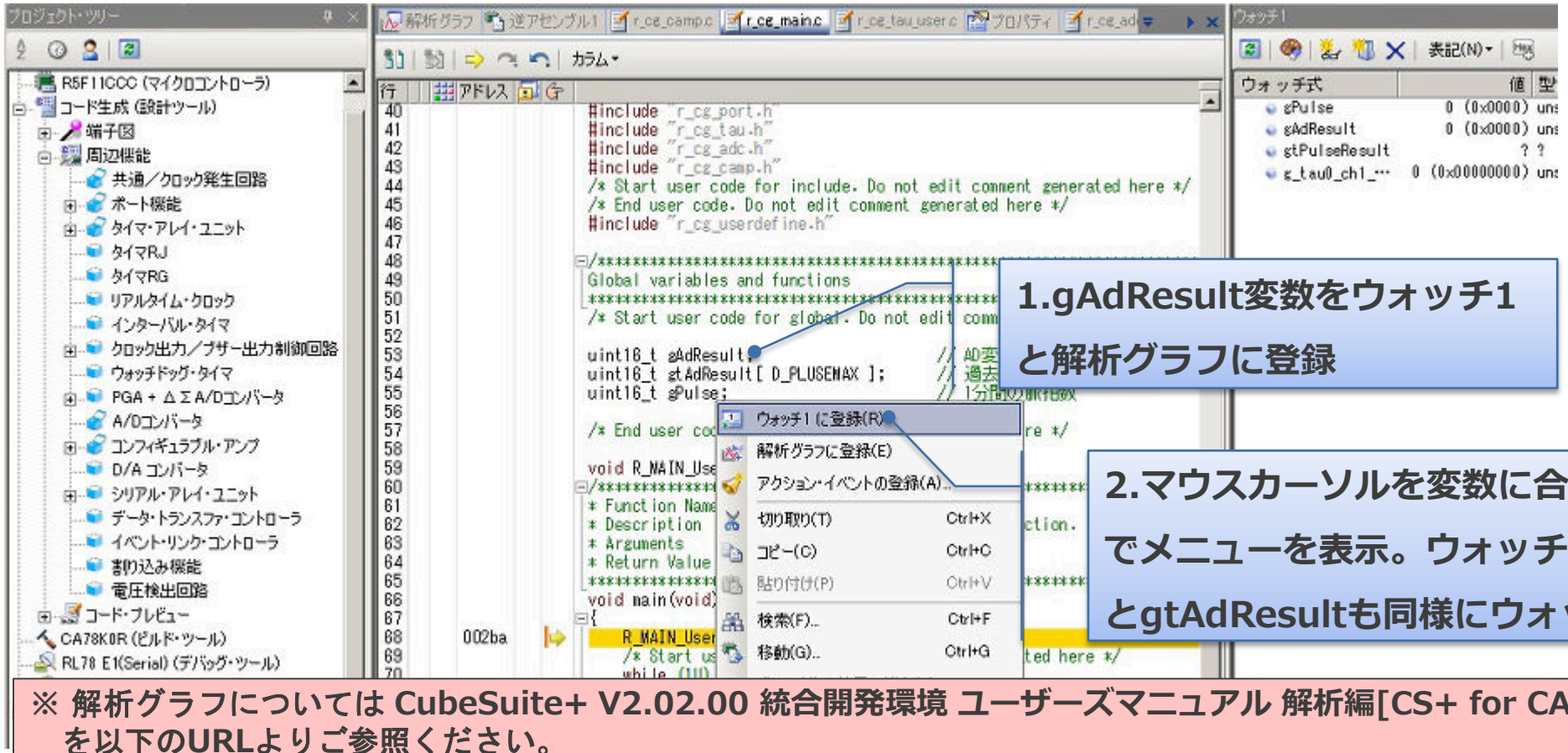


4.ビルドとデバッグを
１つのメニューで実行

5.実行ボタンを押下

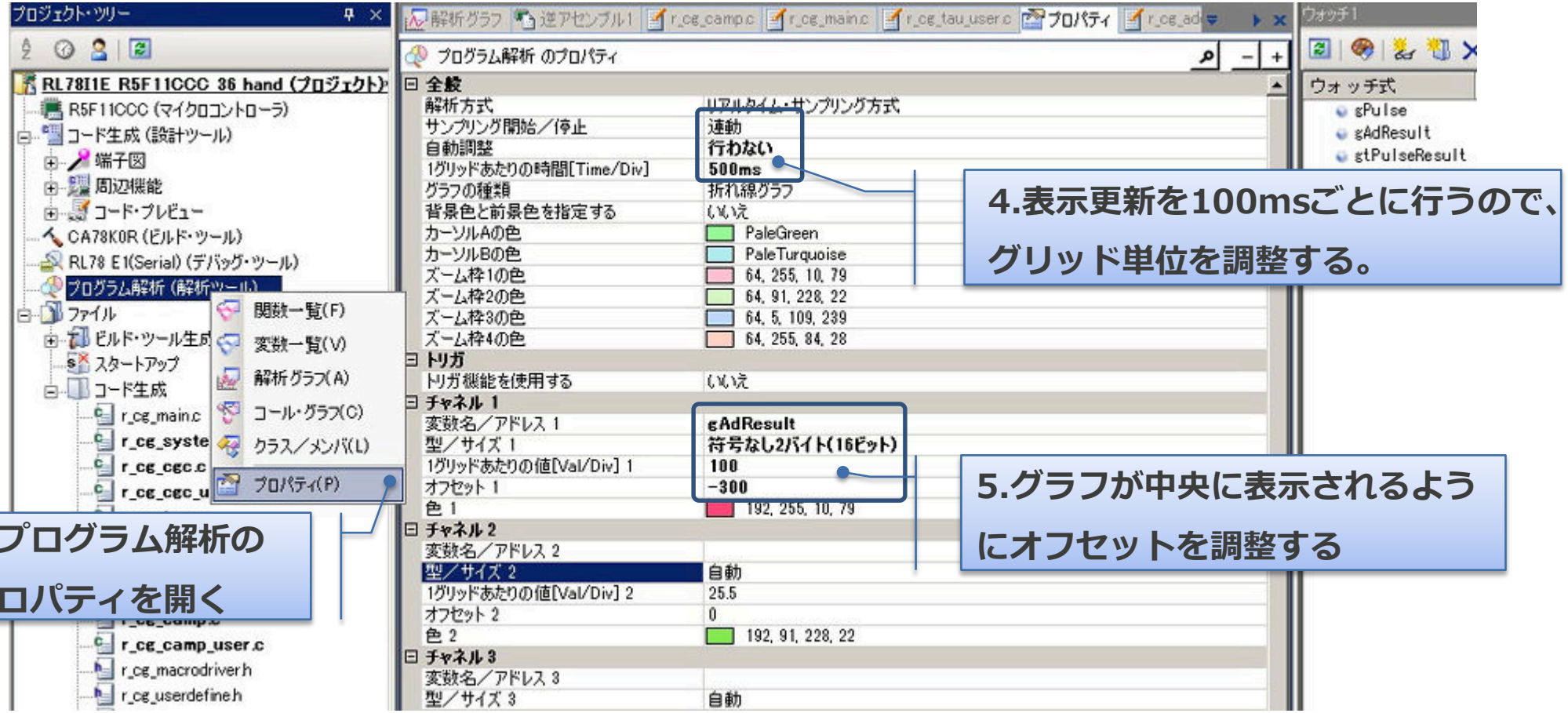


プログラムの実行　（ウォッチ、解析グラフ※の設定）

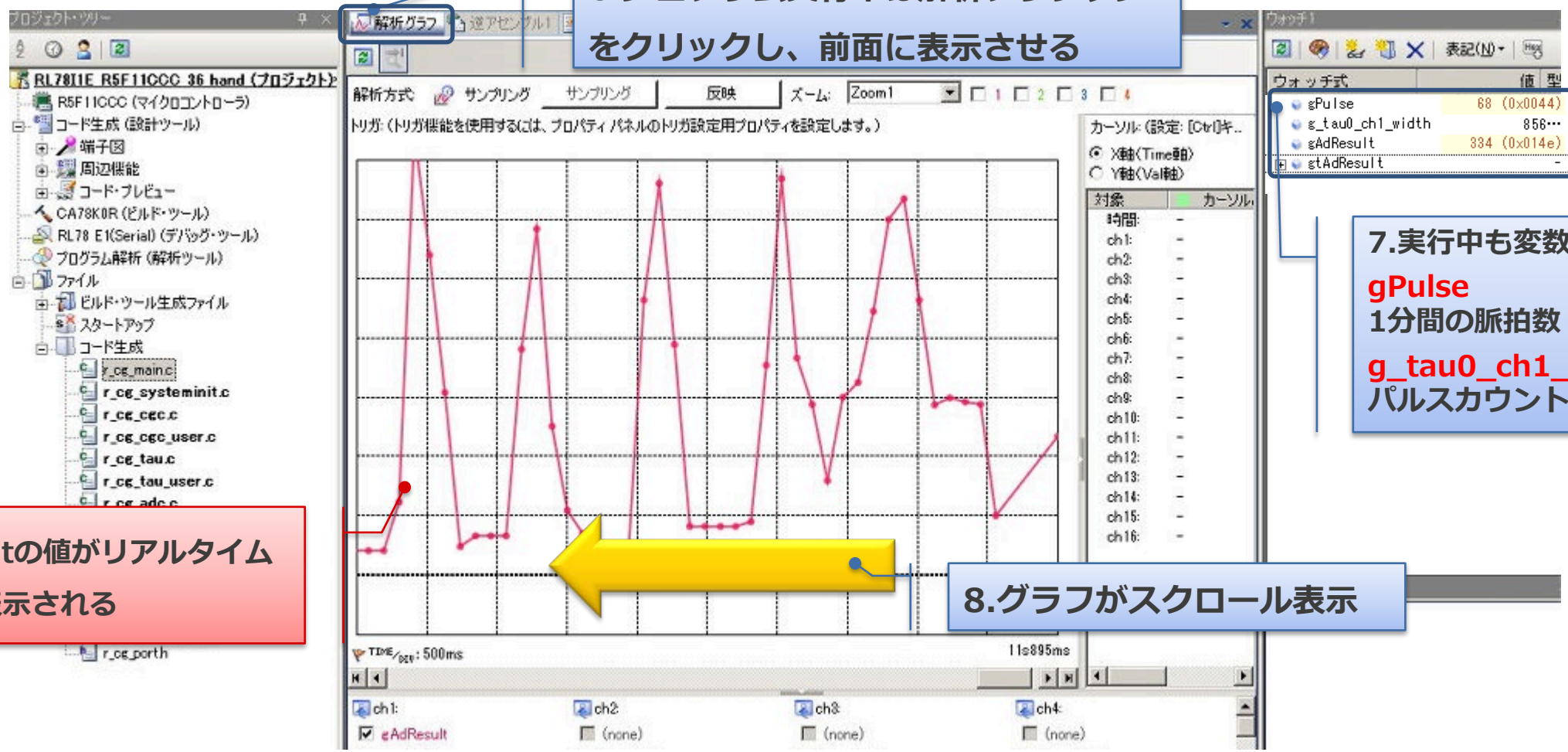


※ 解析グラフについては CubeSuite+ V2.02.00 統合開発環境 ユーザーズマニュアル 解析編[CS+ for CA,CX]を以下のURLよりご参照ください。
http://japan.renesas.com/products/tools/ide/csp/Documentation.jsp

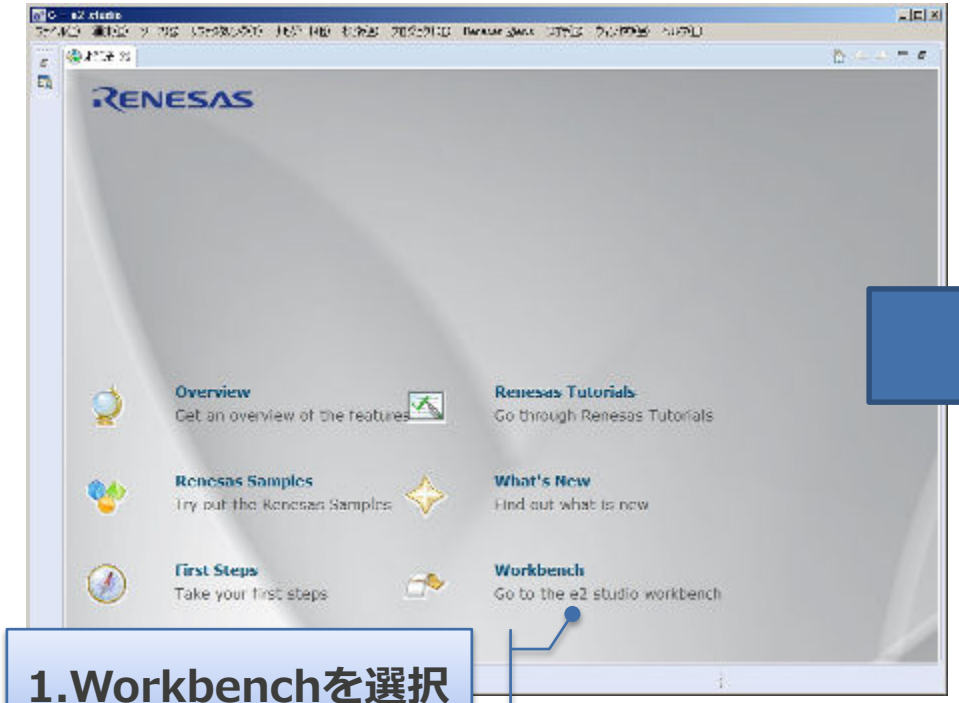
プログラムの実行 （ 解析グラフの詳細設定 ）

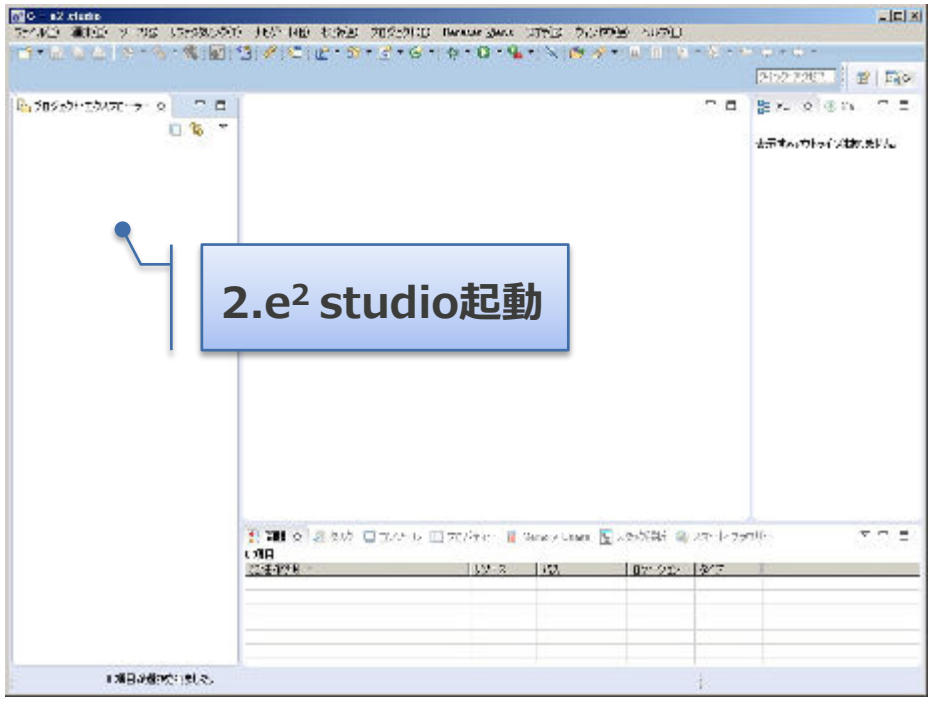


プログラムの実行 （プログラム実行中の表示）

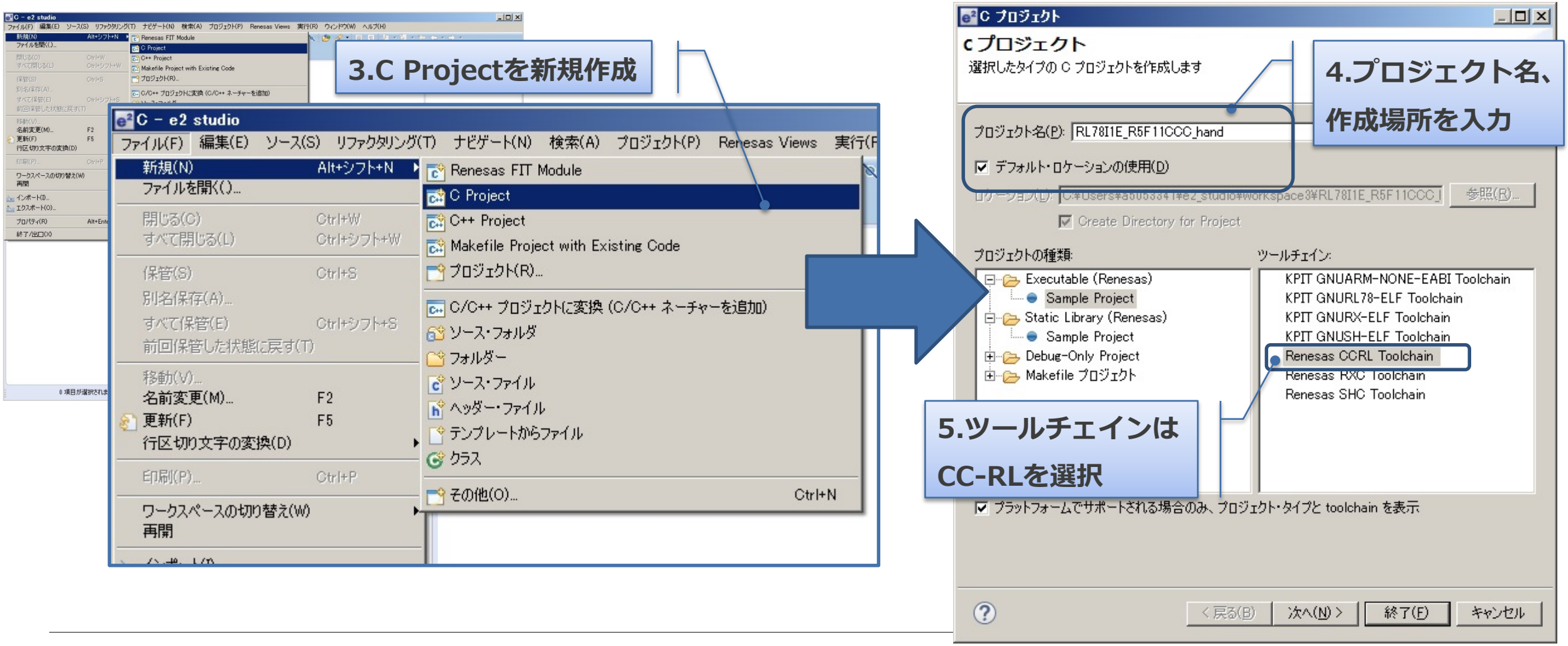


e² studioでプロジェクト作成

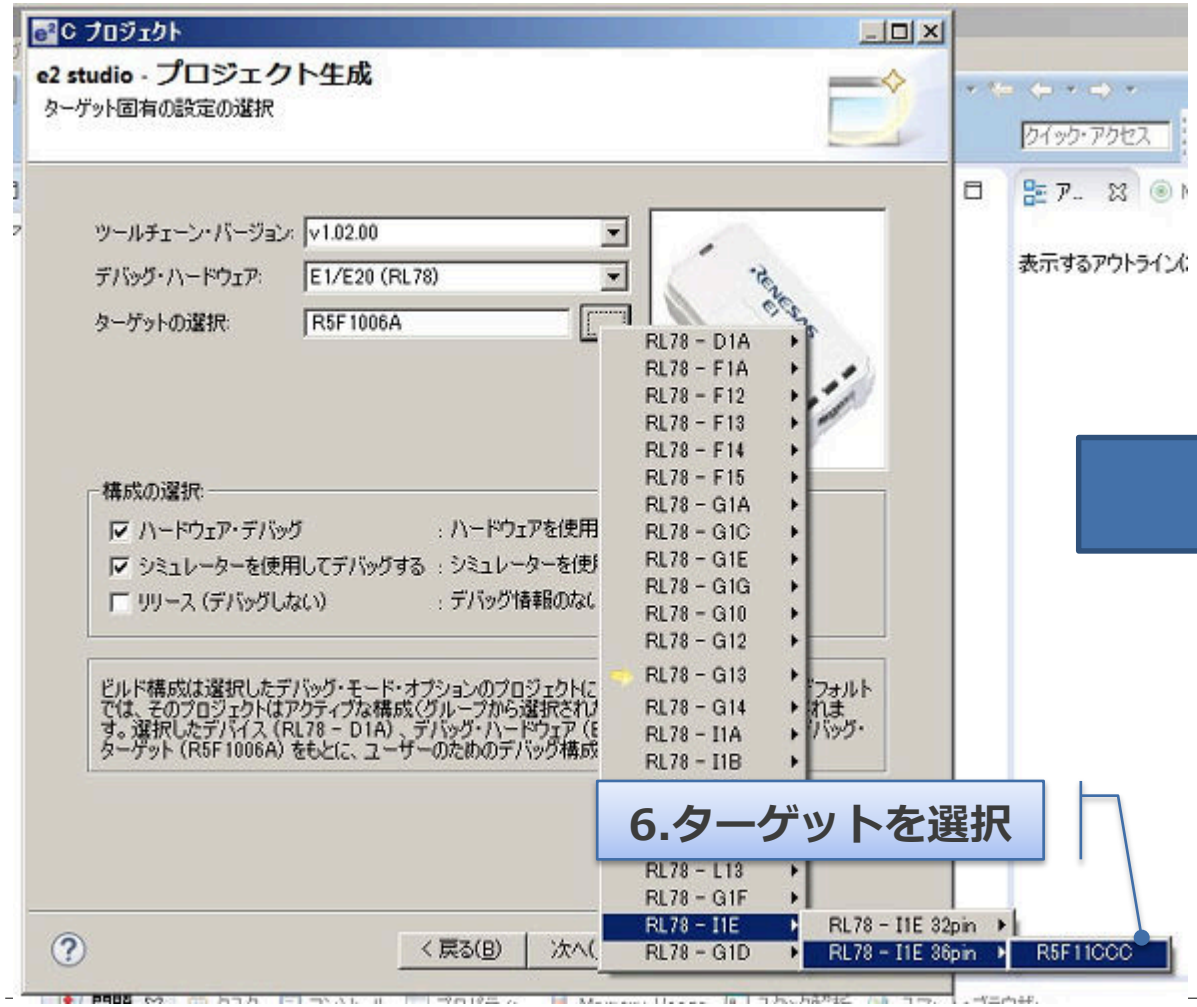


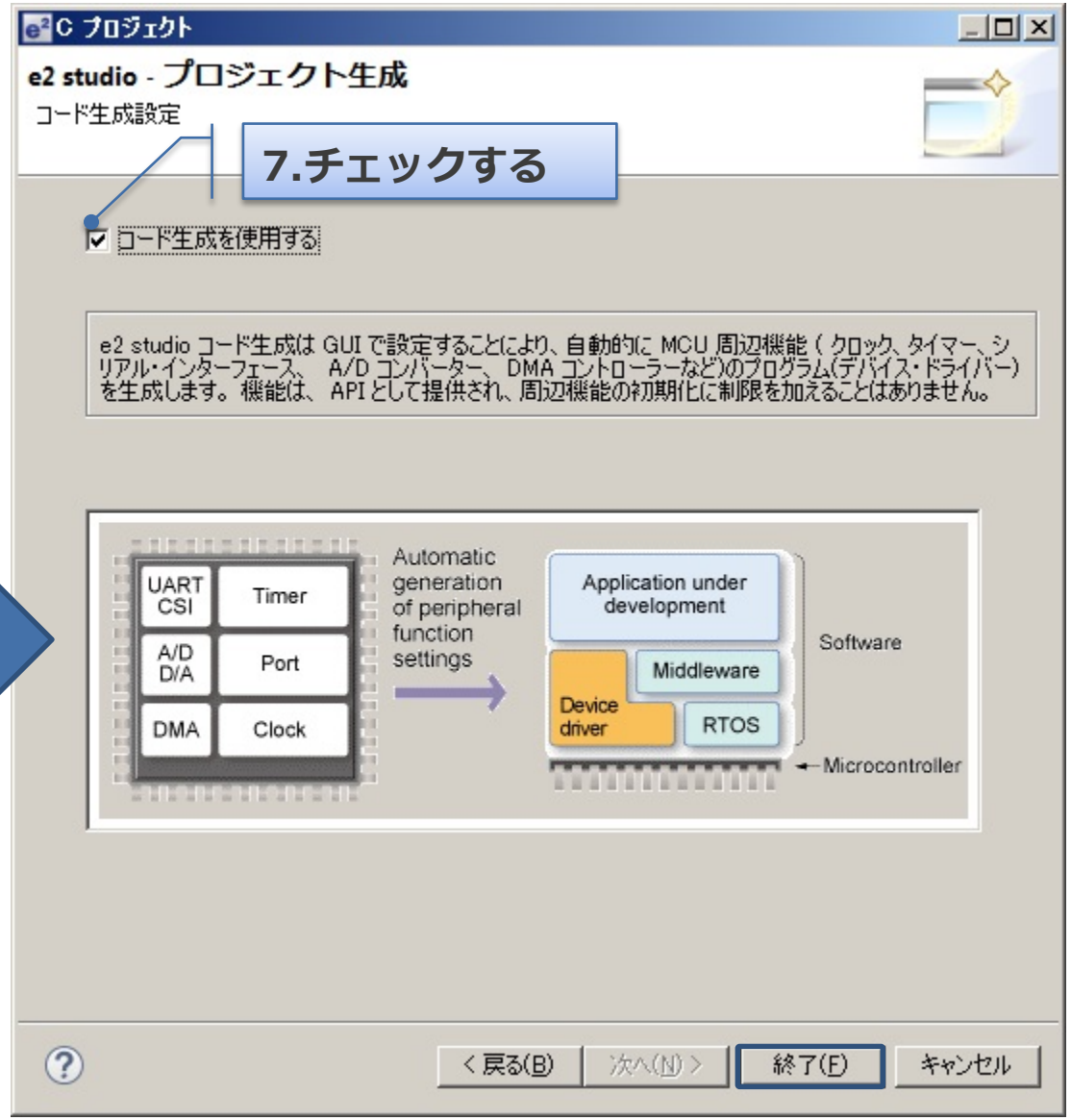


e² studioでプロジェクト作成

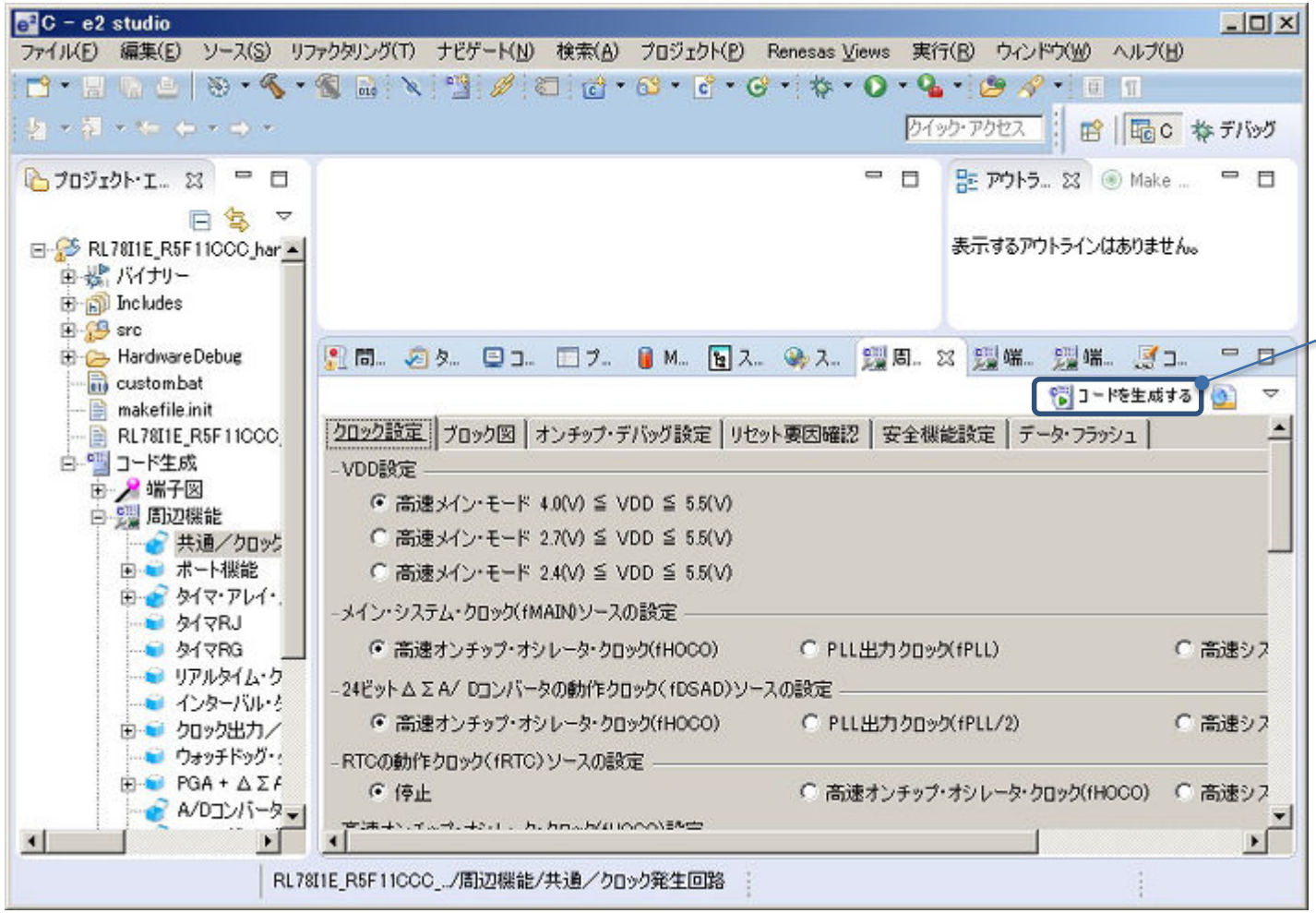


e² studioでプロジェクト作成





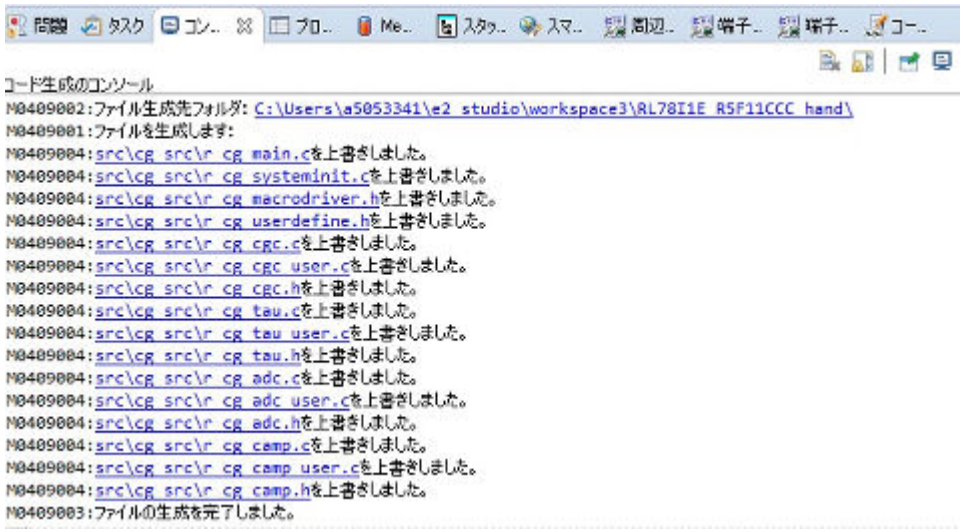
e² studioでプロジェクト作成



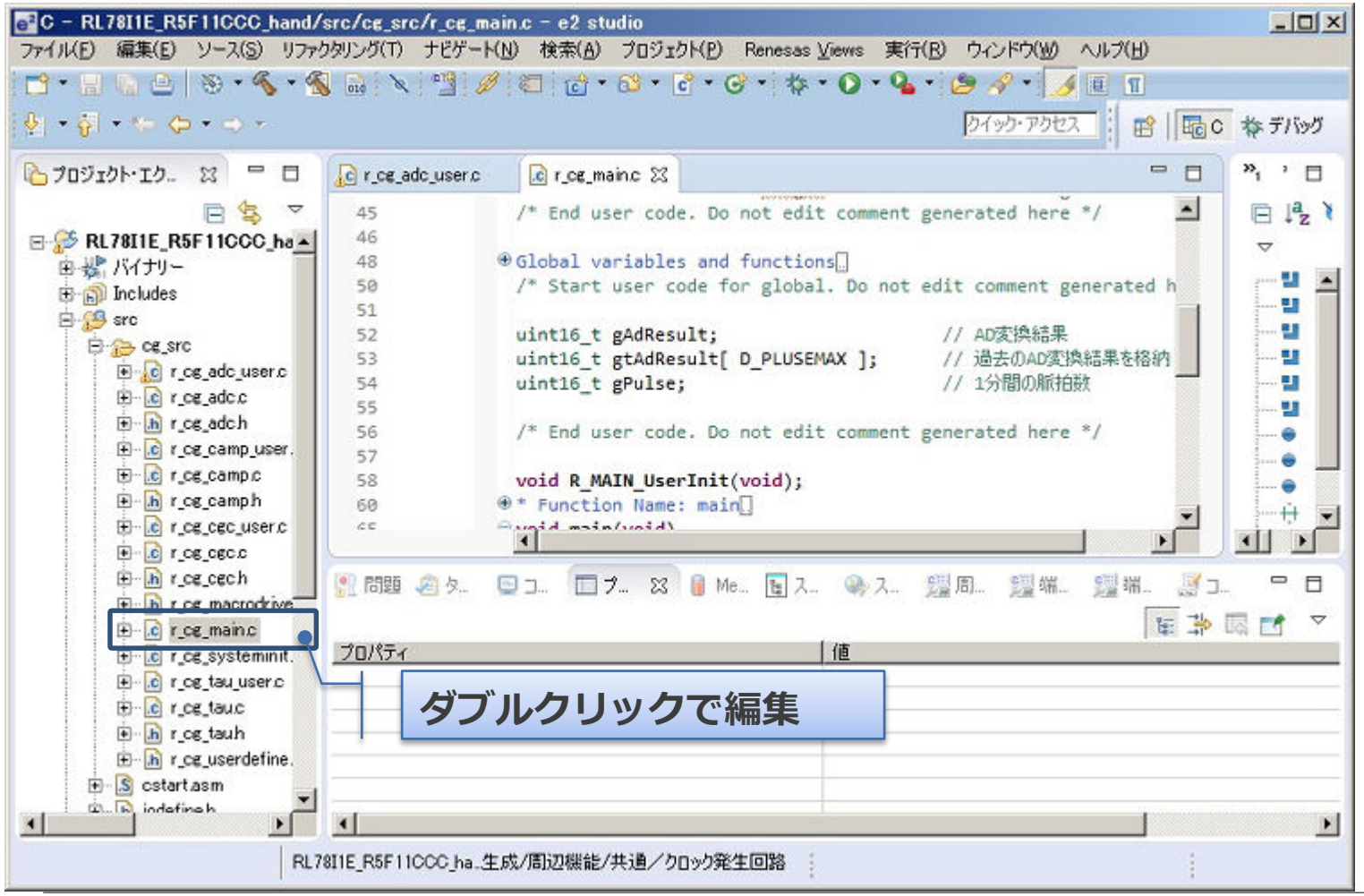
コード生成で設定する内容について

・9～19ページを参照

8.「コードを生成する」を押下



e² studioでプログラム編集



プログラム編集と内容について

・19～24ページを参照

生成されるソースファイル

・周辺機能の初期化と制御APIを含むもの

r_cg_周辺機能.c / .h

・周辺機能の割り込み関数を含むもの

r_cg_周辺機能_user.c

・main()関数があるファイル

r_cg_main.c

・コード生成で使う変数型の定義など

r_cg_macrodriver.h

・ユーザ用の共通定義するためのファイル

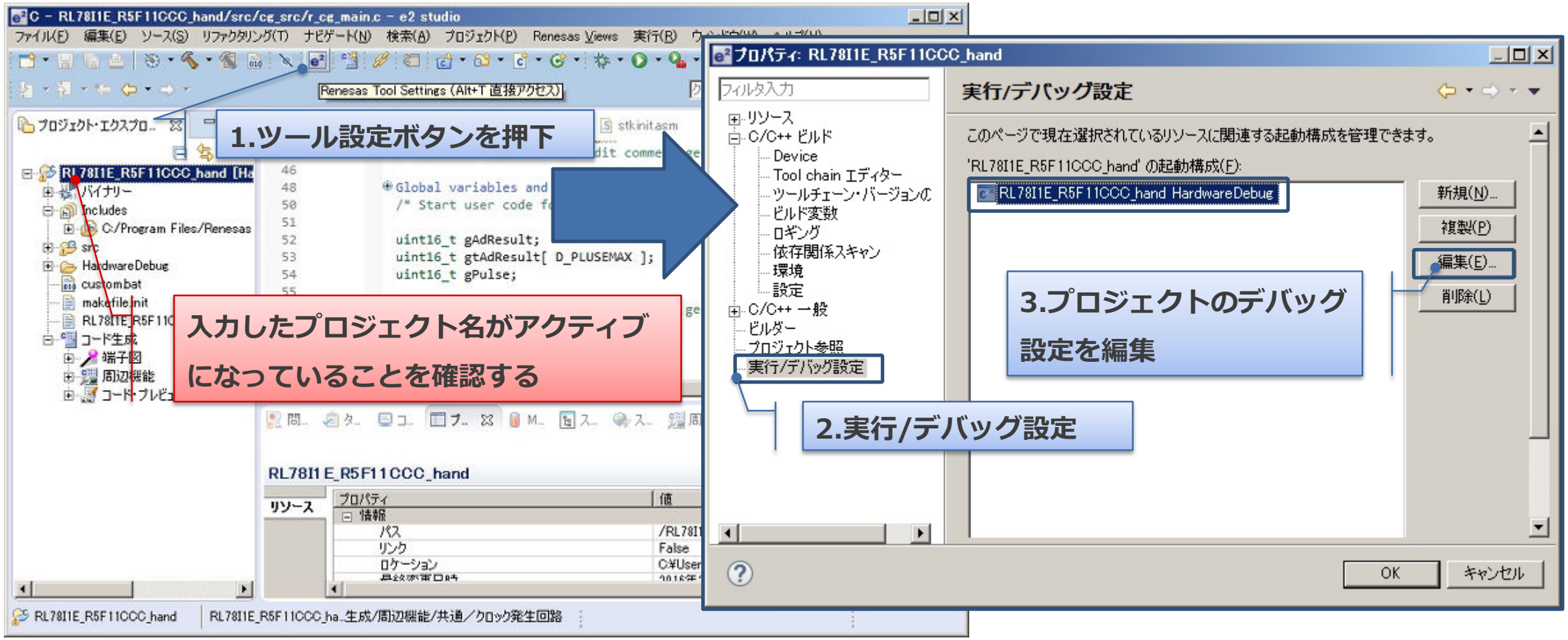
r_cg_userdefine.h

CC-RLデフォルトで用意されるファイル

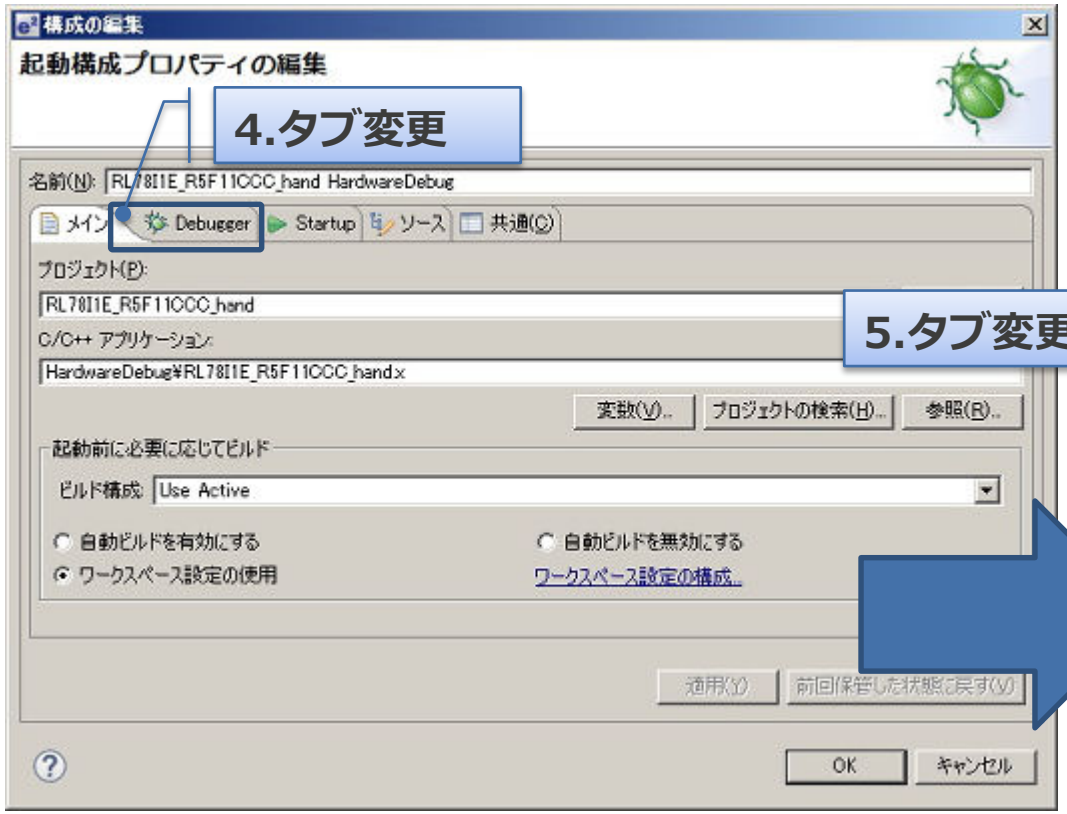
cstart.asm, stkinit.asm → CC-RL用スタートアップ

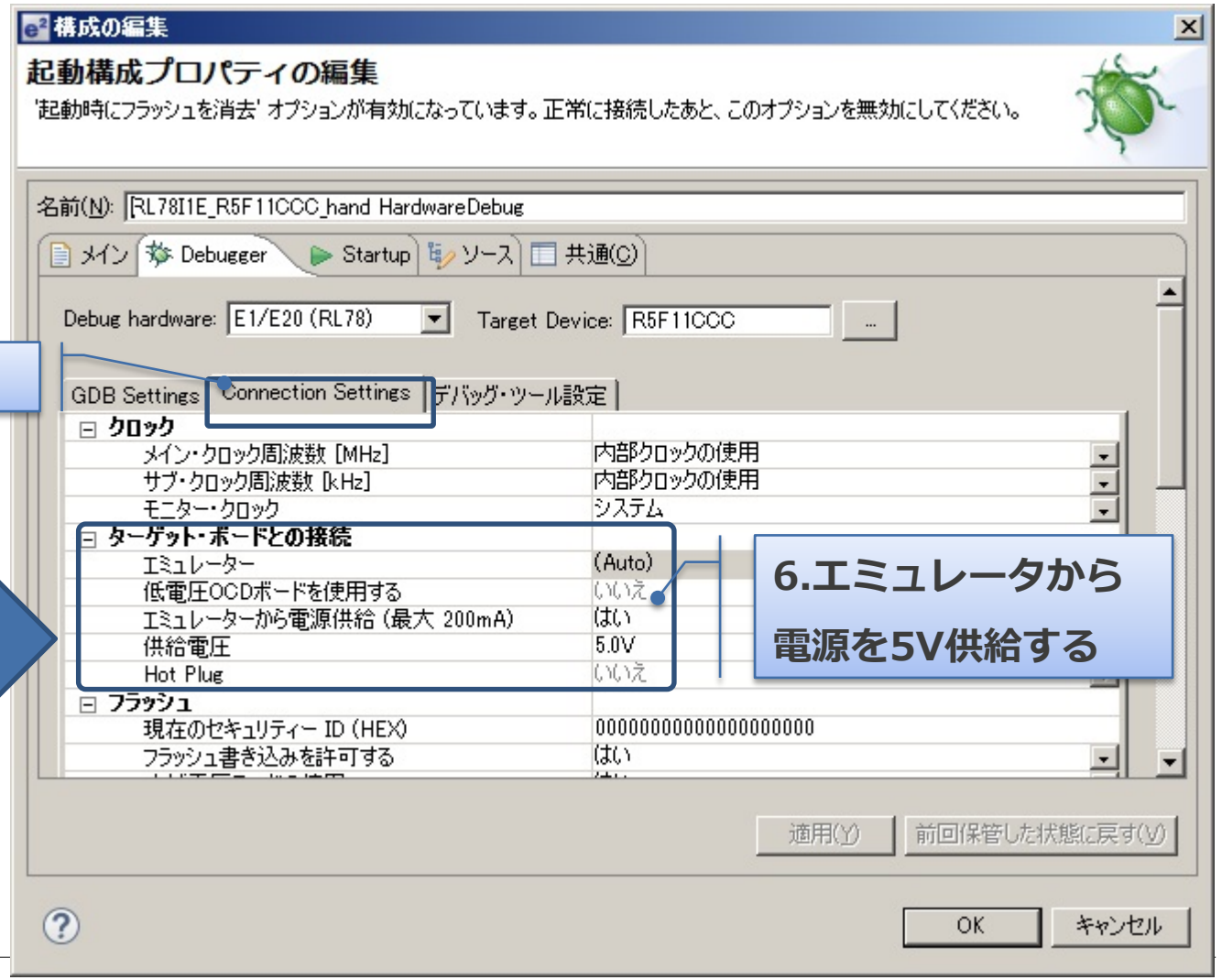
iodefine.h → 周辺レジスタ定義ファイル

e² studioでデバッグツールの設定

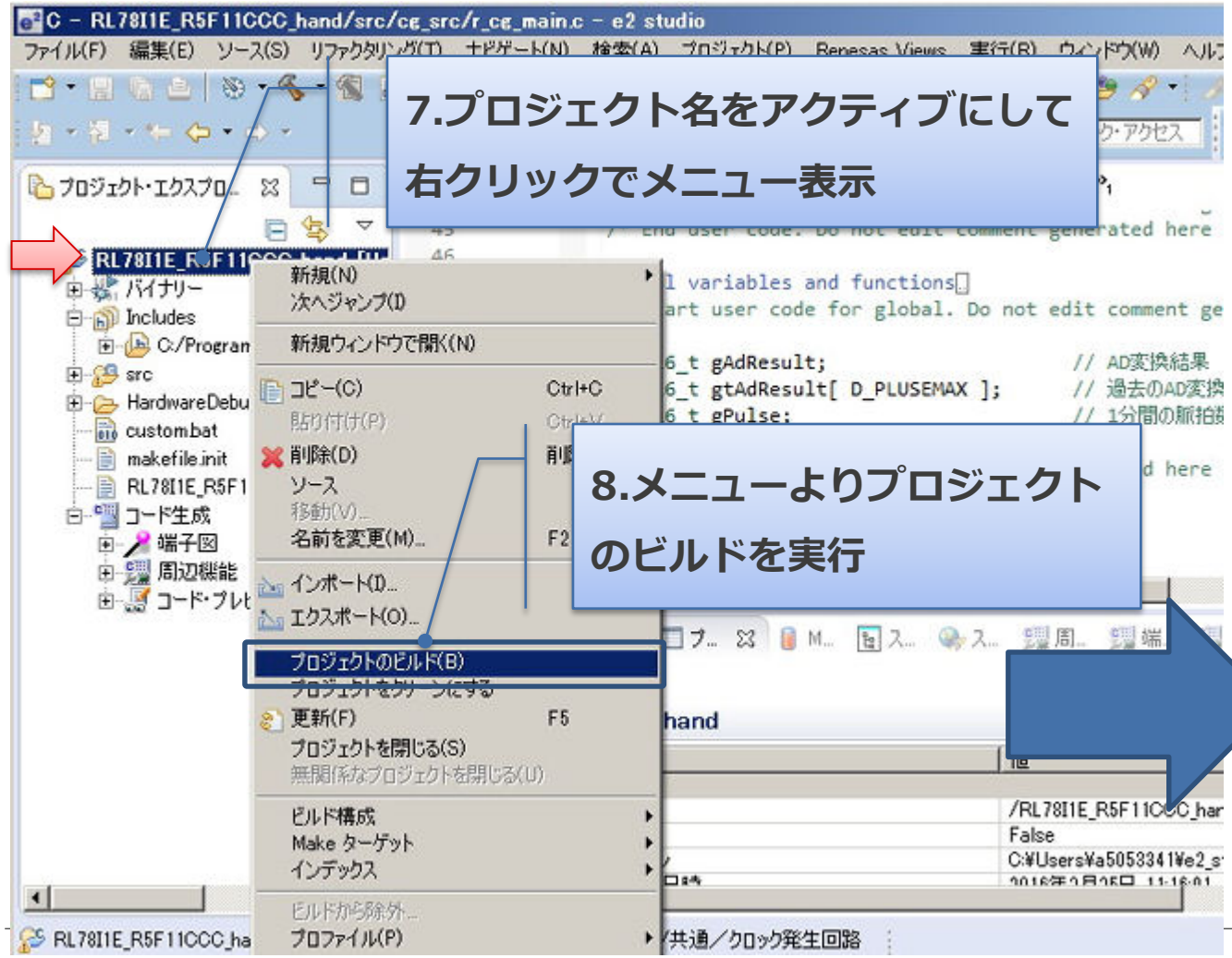


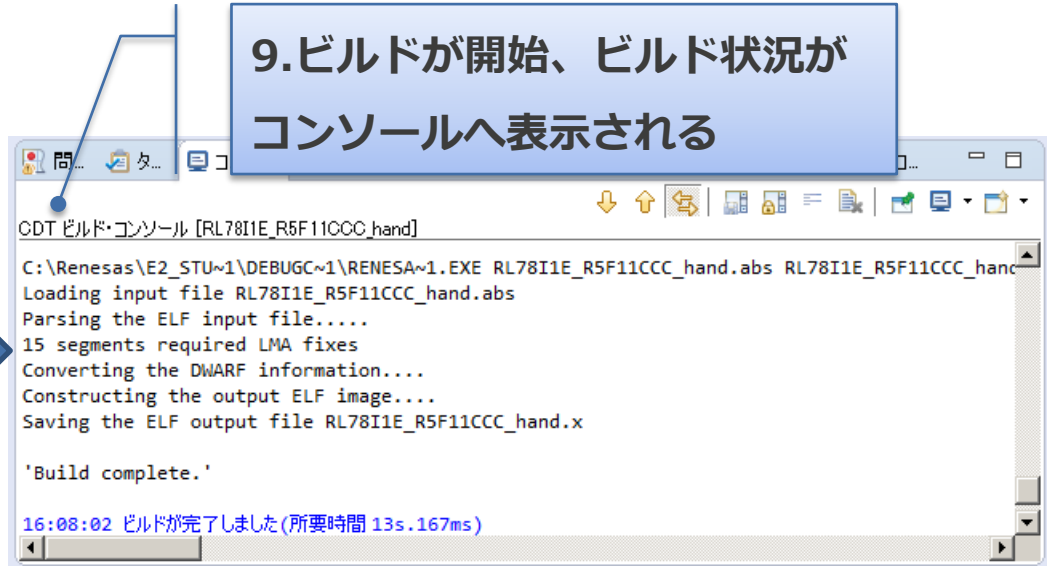
e² studioでデバッグツールの設定



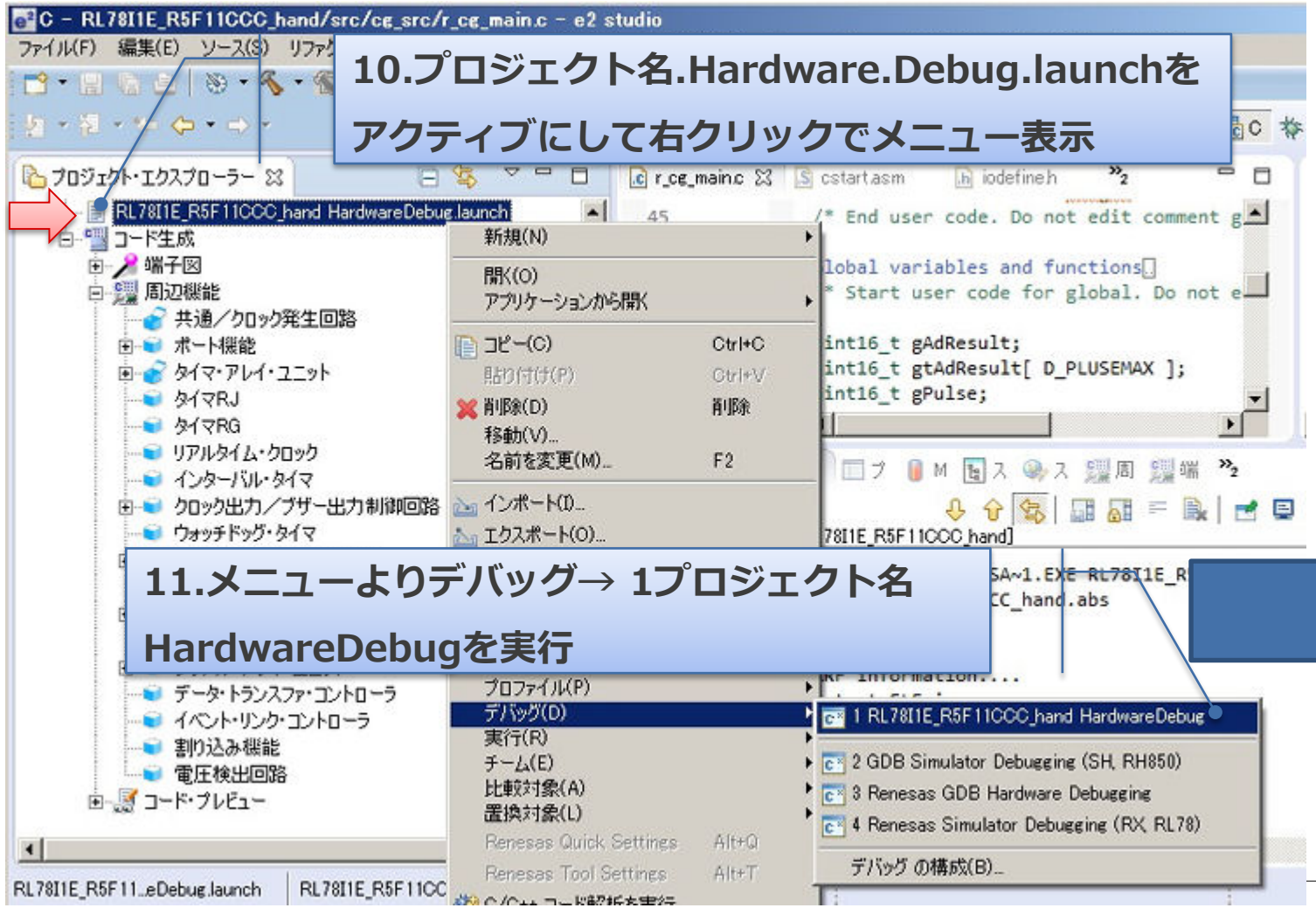


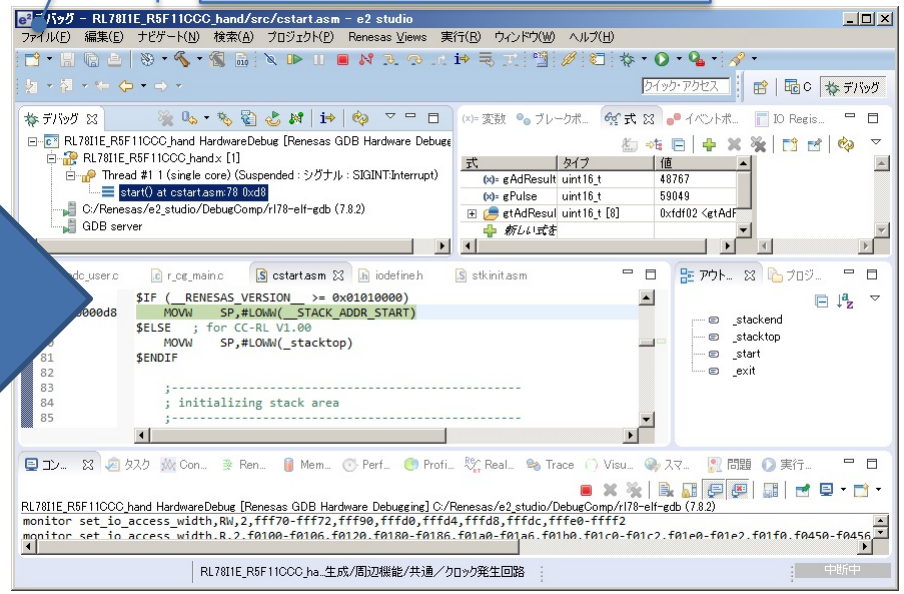
e² studioでデバッグツールの設定 (プロジェクトのビルド)



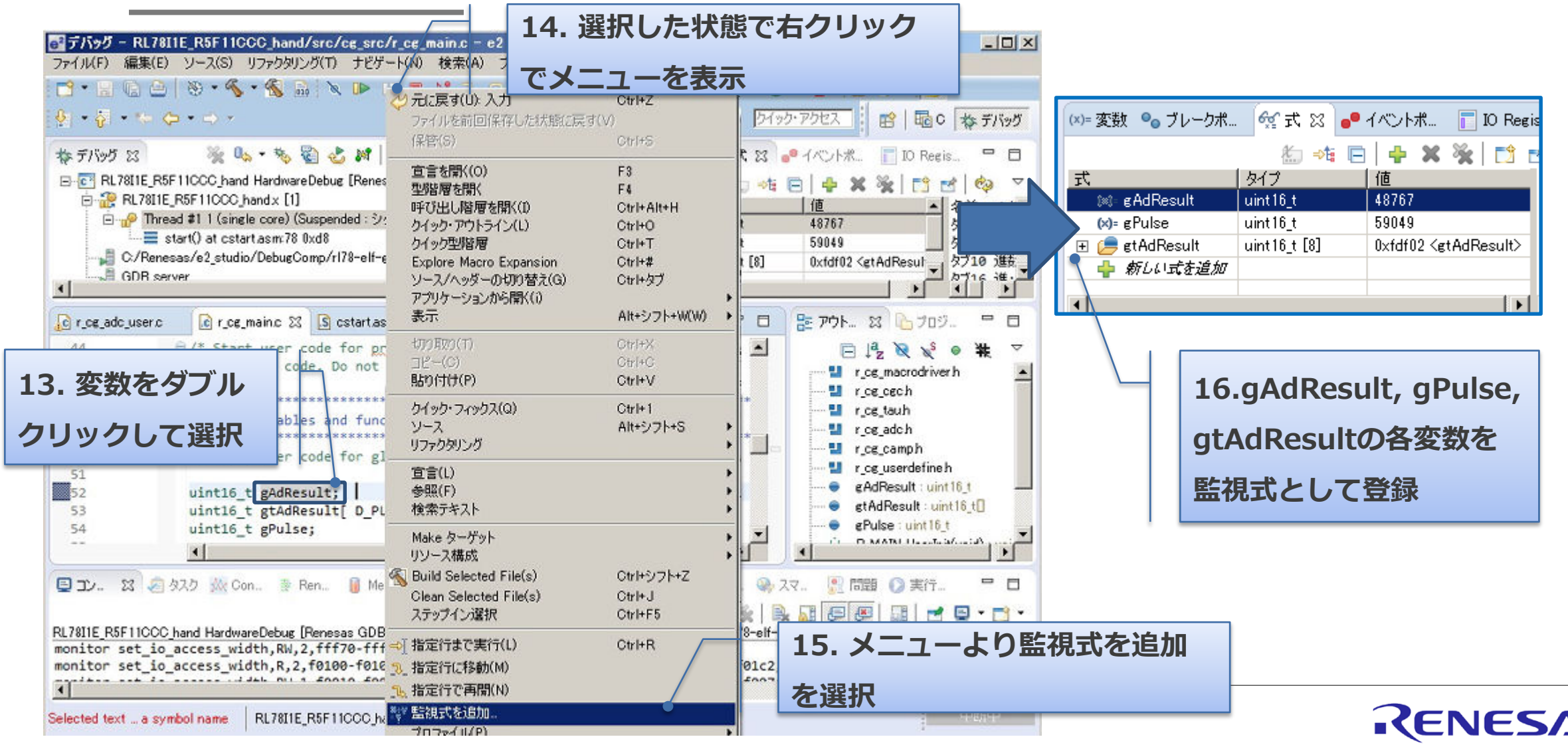


e² studioでデバッグツールの設定 (プロジェクトのダウンロード)



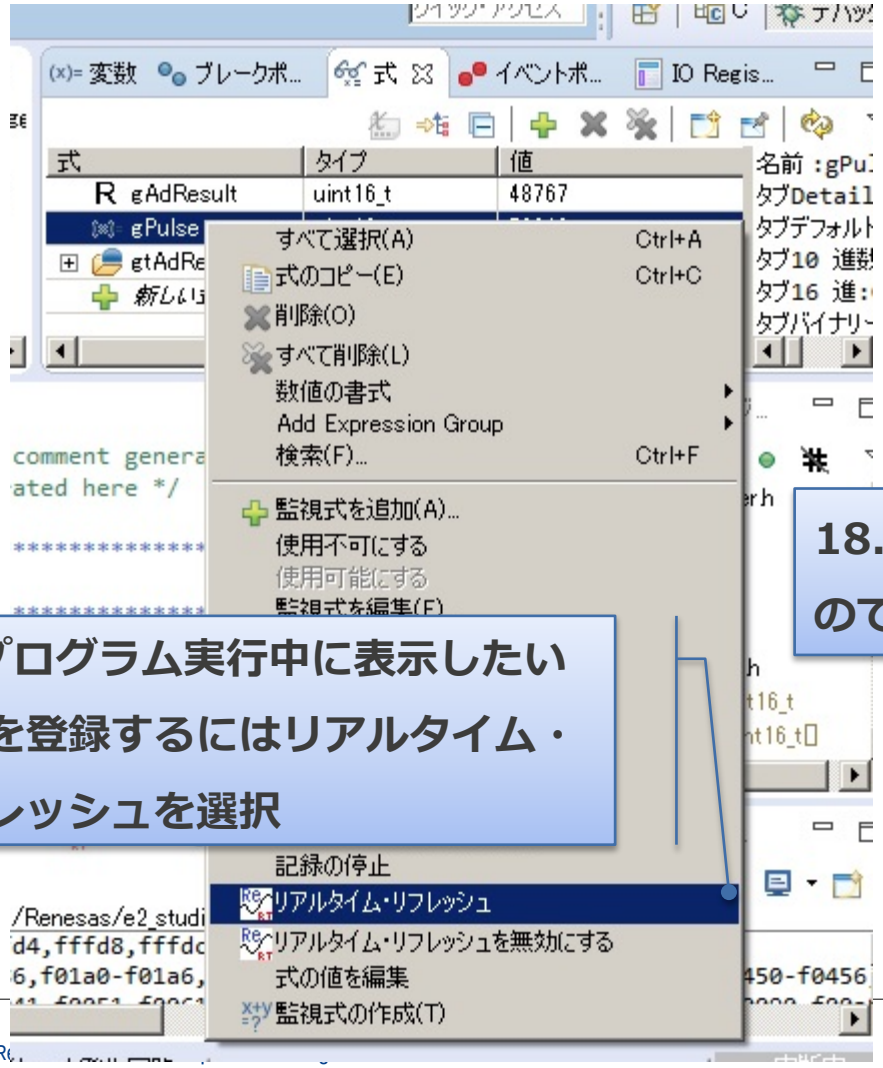


e² studioでデバッグツールの設定 (監視式:実行時に表示したい変数の追加)

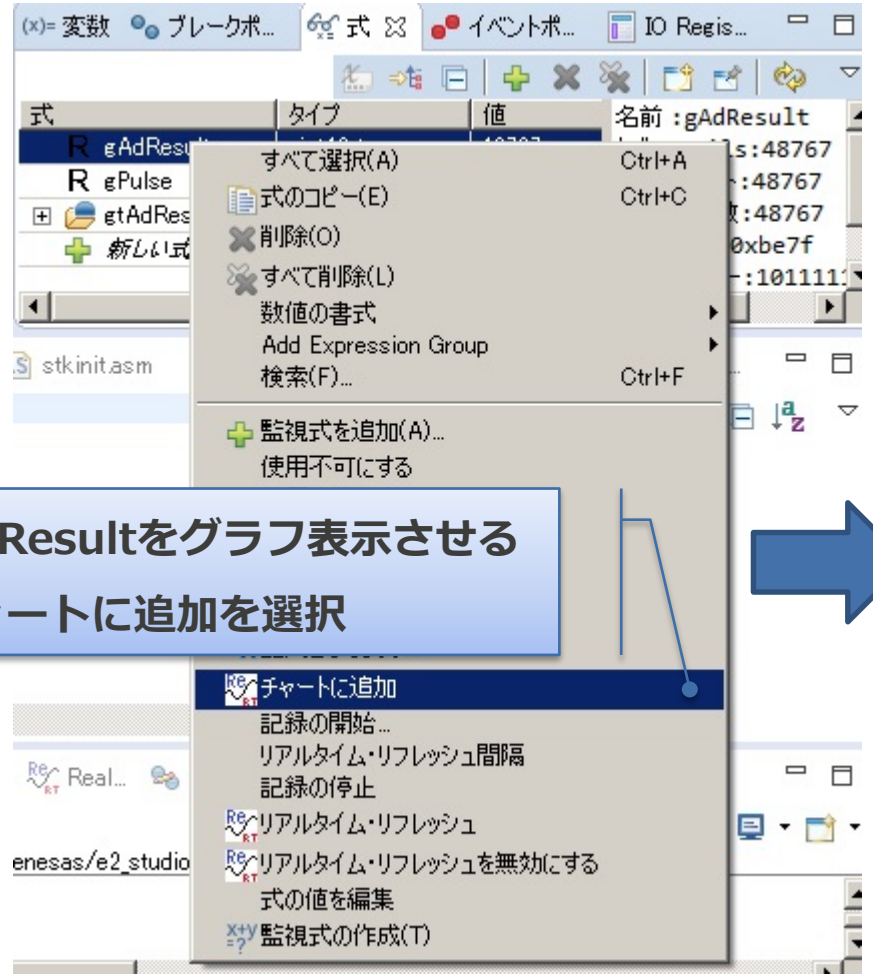


e² studioでデバッグツールの設定 (監視式の設定)

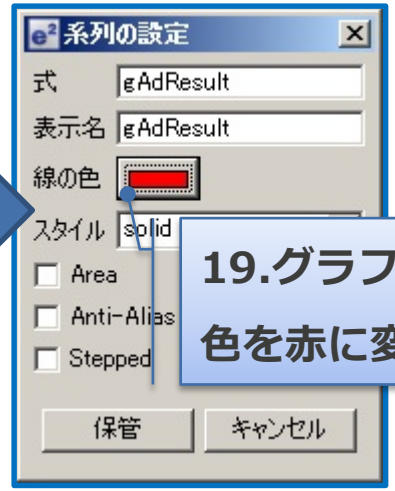
※この設定はデバッガ起動時に
毎回行う必要があります



17.プログラム実行中に表示したい
変数を登録するにはリアルタイム・
リフレッシュを選択

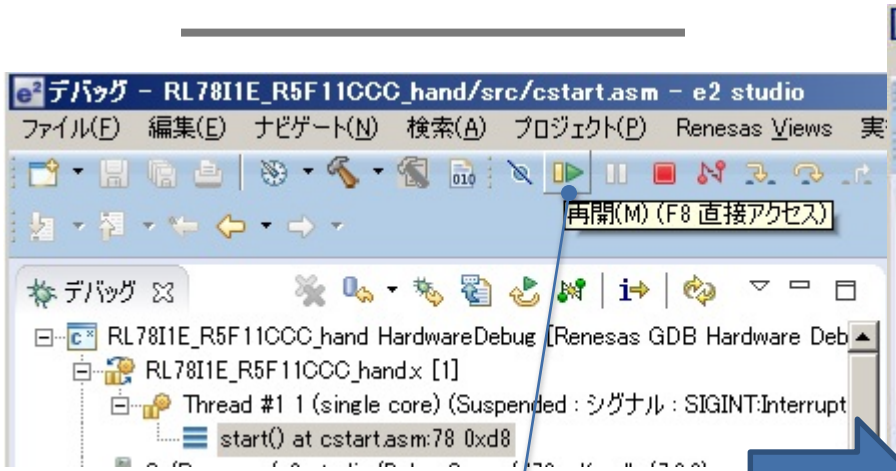


18.gAdResultをグラフ表示させる
のでチャートに追加を選択

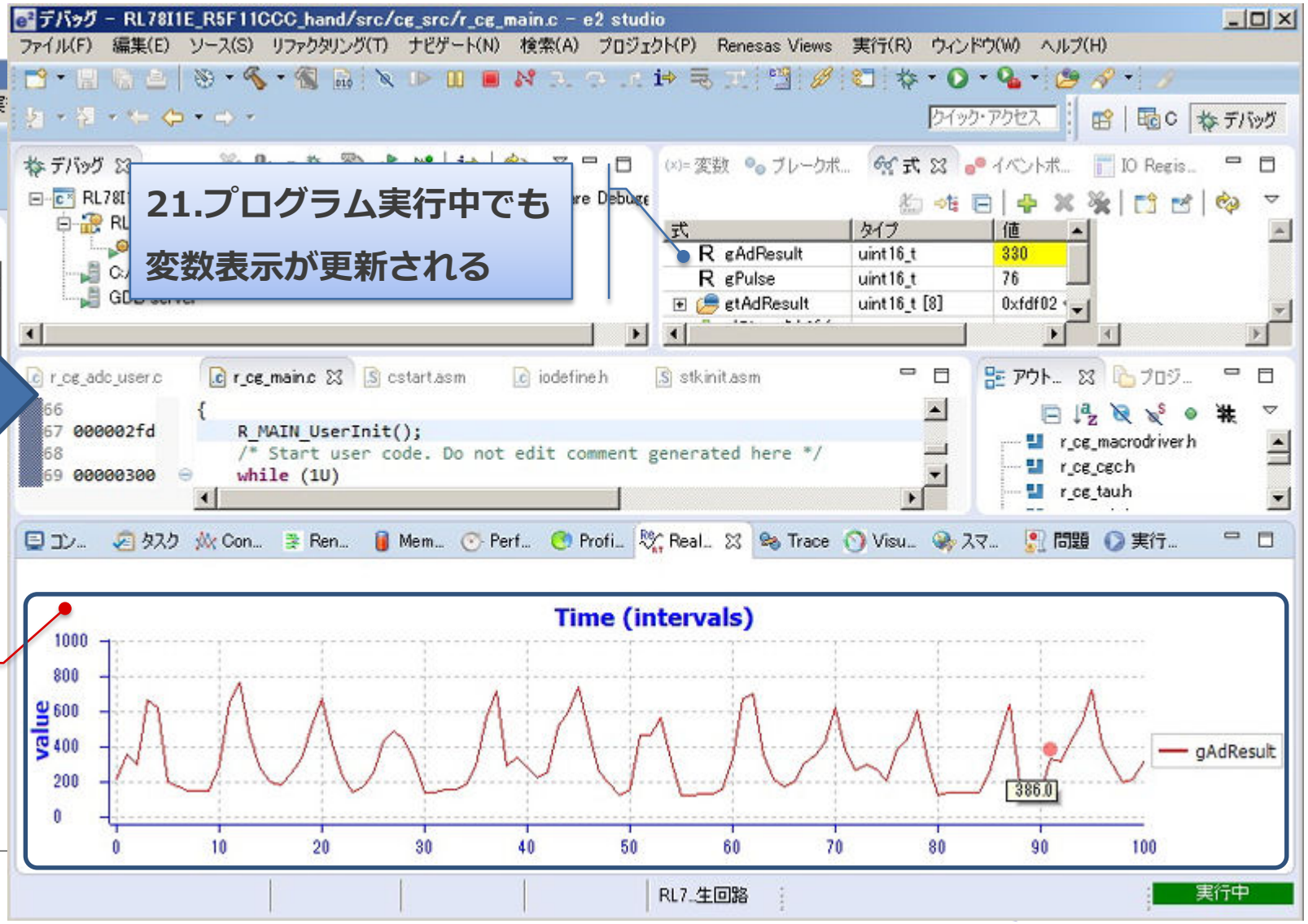


19.グラフ表示
色を赤に変更

e² studioでデバッグツールの設定 (プログラムの実行)



20.プログラムを実行



21.プログラム実行中でも
変数表示が更新される

gAdResult変数がリアル
タイムでグラフ表示される

最後に

他にもコード生成のガイドがあるので活用してください。

http://documentation.renesas.com/doc/products/tool/doc/r20ut3230jj0100_cg_guide.pdf

改訂記録

Rev	発行日	内容	備考
1.00	2016.3.10	新規発行	
1.10	2018.6.4	ページ11 オンチップデバッグ設定を追記 ページ22 r_cg_userdefine.hソース記載を追記	