

RL78/I1Eコード生成

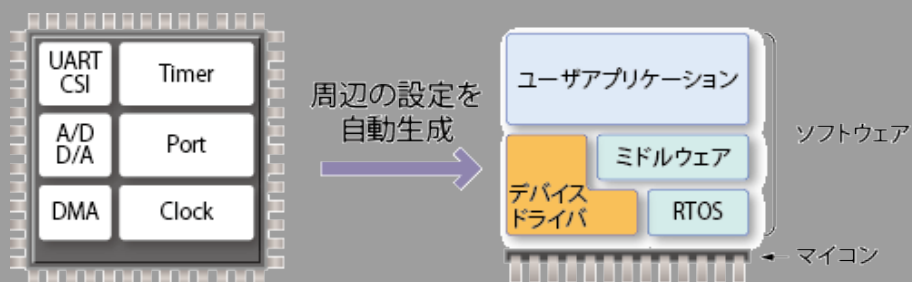
RL78/I1E + 脈拍センサデモ コンフィギュラブル・アンプ使用例

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。

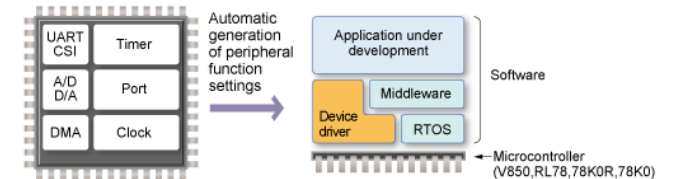
2018.06.04

目次

▪ コード生成概要	ページ 03
▪ 汎用アンプを使ったデモ概要	ページ 05
▪ CS+でプロジェクト作成	ページ 07
▪ コード生成で周辺機能設定	ページ 09
▪ ソースコードを自動生成	ページ 19
▪ プログラム編集	ページ 20
▪ デバッグツールの設定	ページ 26
▪ プログラムの実行	ページ 28
▪ e ² studioでプロジェクト作成	ページ 31
▪ e ² studioでプログラム編集	ページ 35
▪ e ² studioでデバッグツールの設定	ページ 36

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、シリアルのあるレート、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



Peripheral Function	Macro	SubMacro	Setting	Status
UART	UART_NAME	UART_NAME	UART_NAME	設定済
CSI	CSI_NAME	CSI_NAME	CSI_NAME	設定済
Timer	TIMER_NAME	TIMER_NAME	TIMER_NAME	設定済
A/D D/A	AD_CONVERTER_NAME	AD_CONVERTER_NAME	AD_CONVERTER_NAME	設定済
Port	PORT_NAME	PORT_NAME	PORT_NAME	設定済
DMA	DMA_NAME	DMA_NAME	DMA_NAME	設定済
Clock	CLOCK_NAME	CLOCK_NAME	CLOCK_NAME	設定済



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+, e ² 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

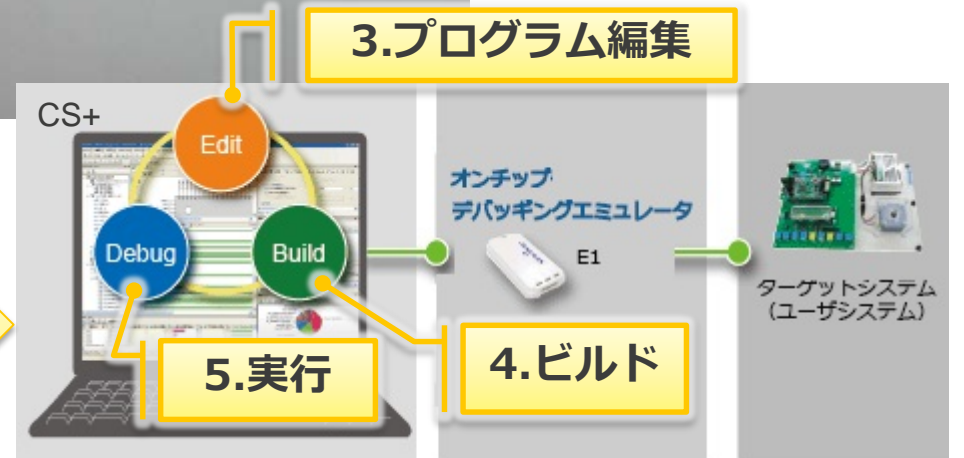
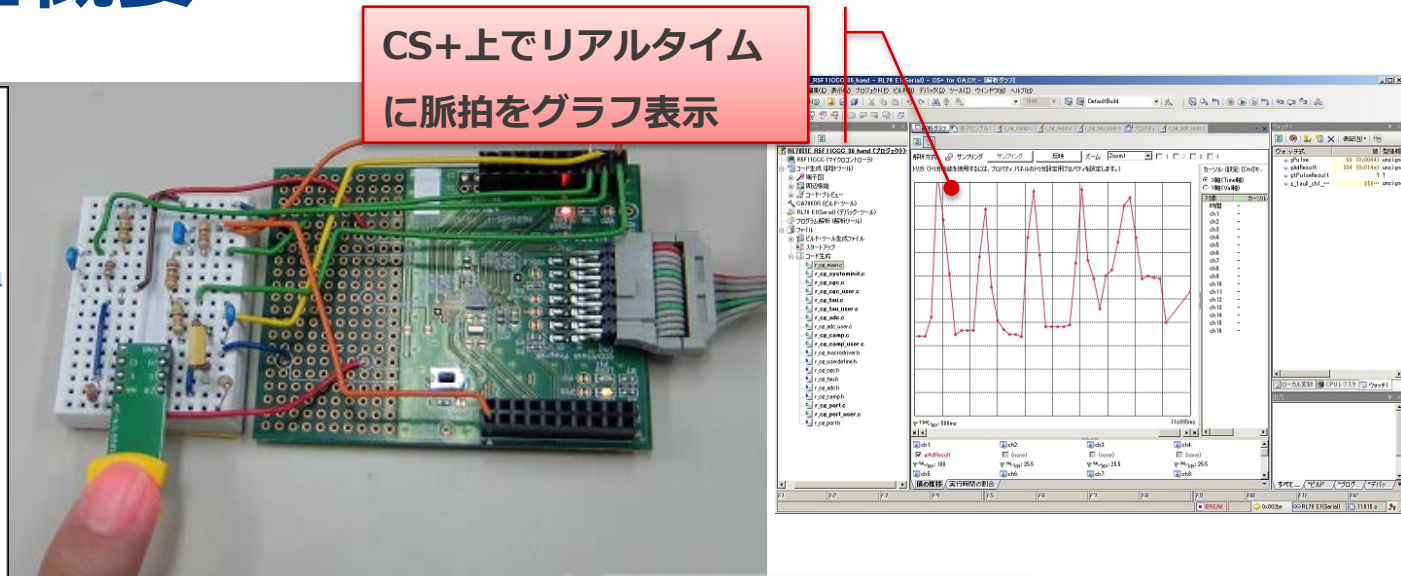
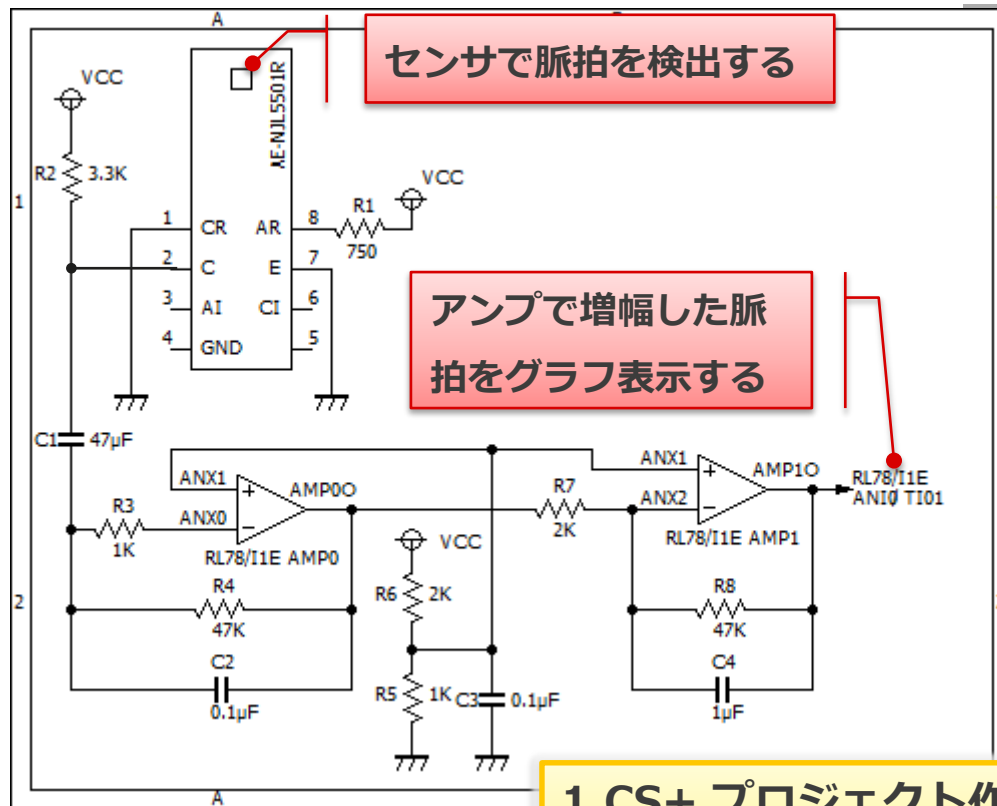
※1 AP4および Applilet は、スタンドアロンツールです。

製品情報の詳細は、以下のURLをご参照ください。

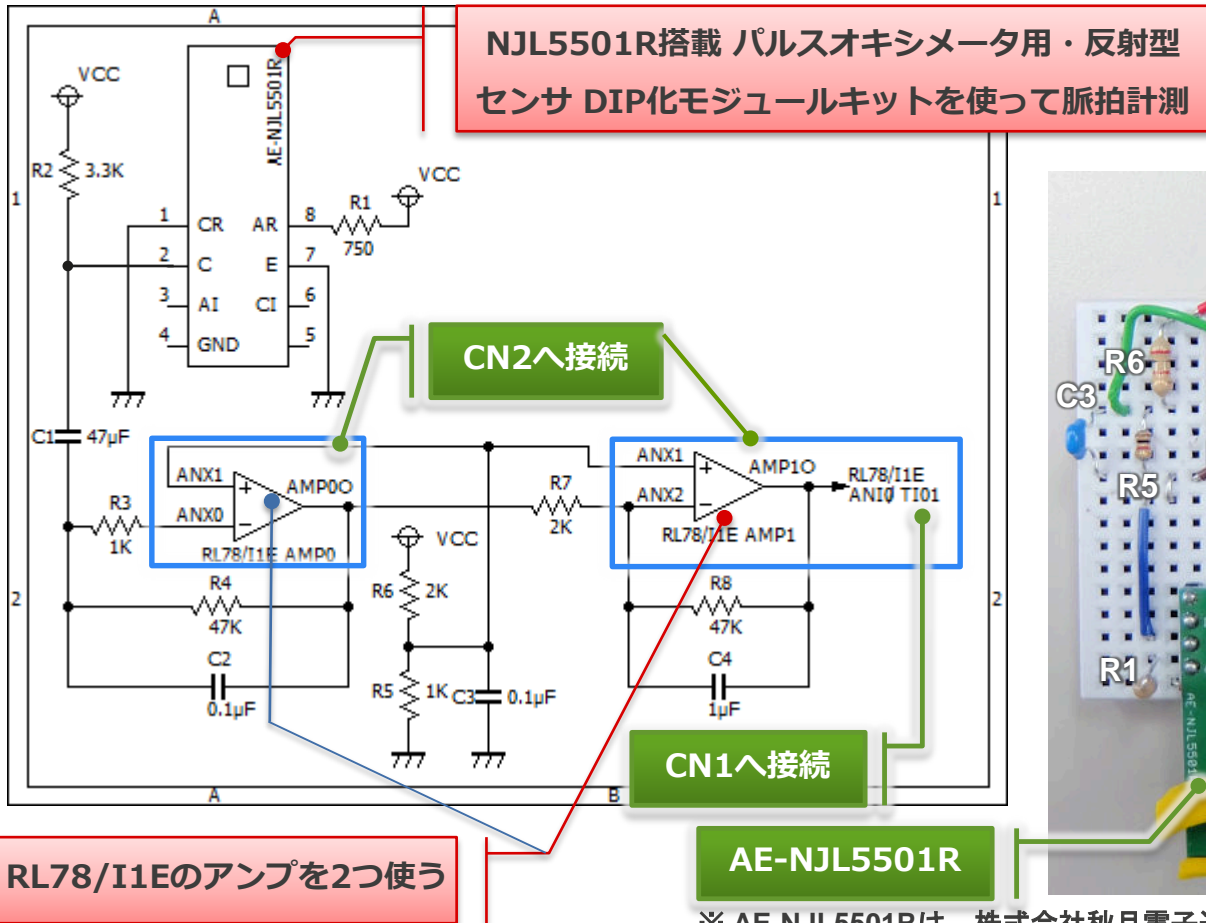
コード生成プラグイン http://japan.renesas.com/cg_p

AP4, Applilet <http://japan.renesas.com/applilet>

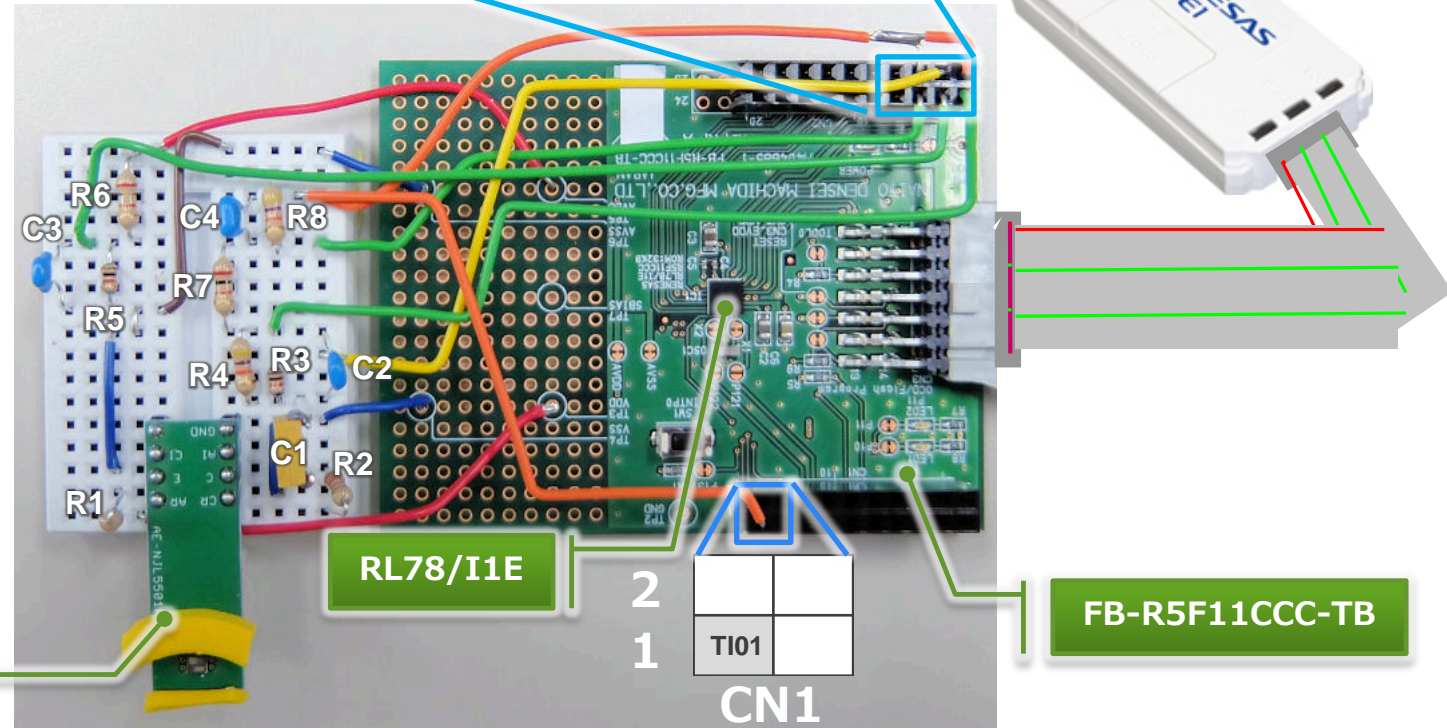
汎用AMPを使ったデモ概要



汎用アンプを使ったデモ概要 (デモ全景)



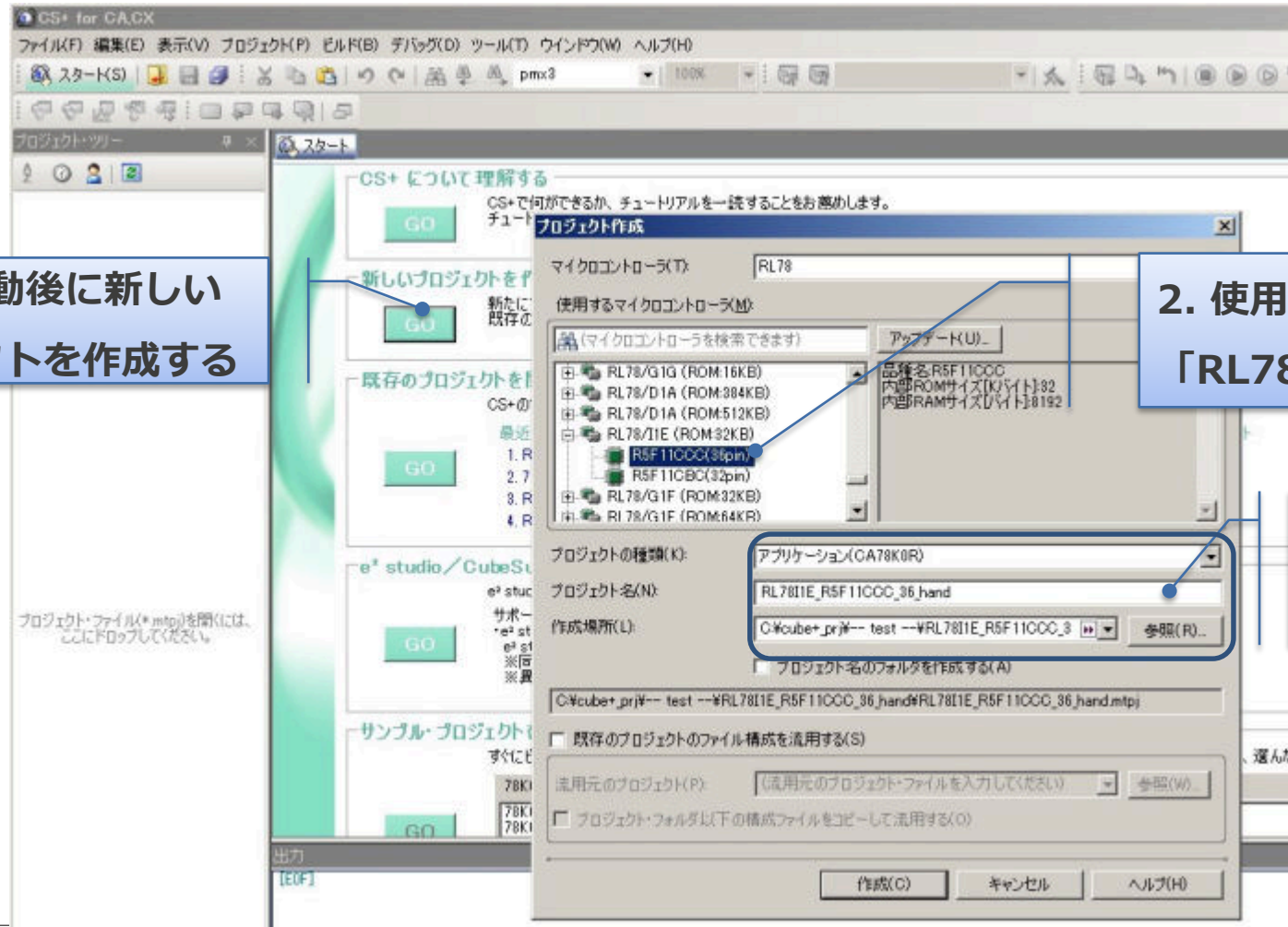
	AMP10	ANX0	ANI0	1
	ANX2	ANX3	AMP00	2



※ 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 R5F1100C(36pin)」を選択

3. プロジェクト名、作成場所を入力

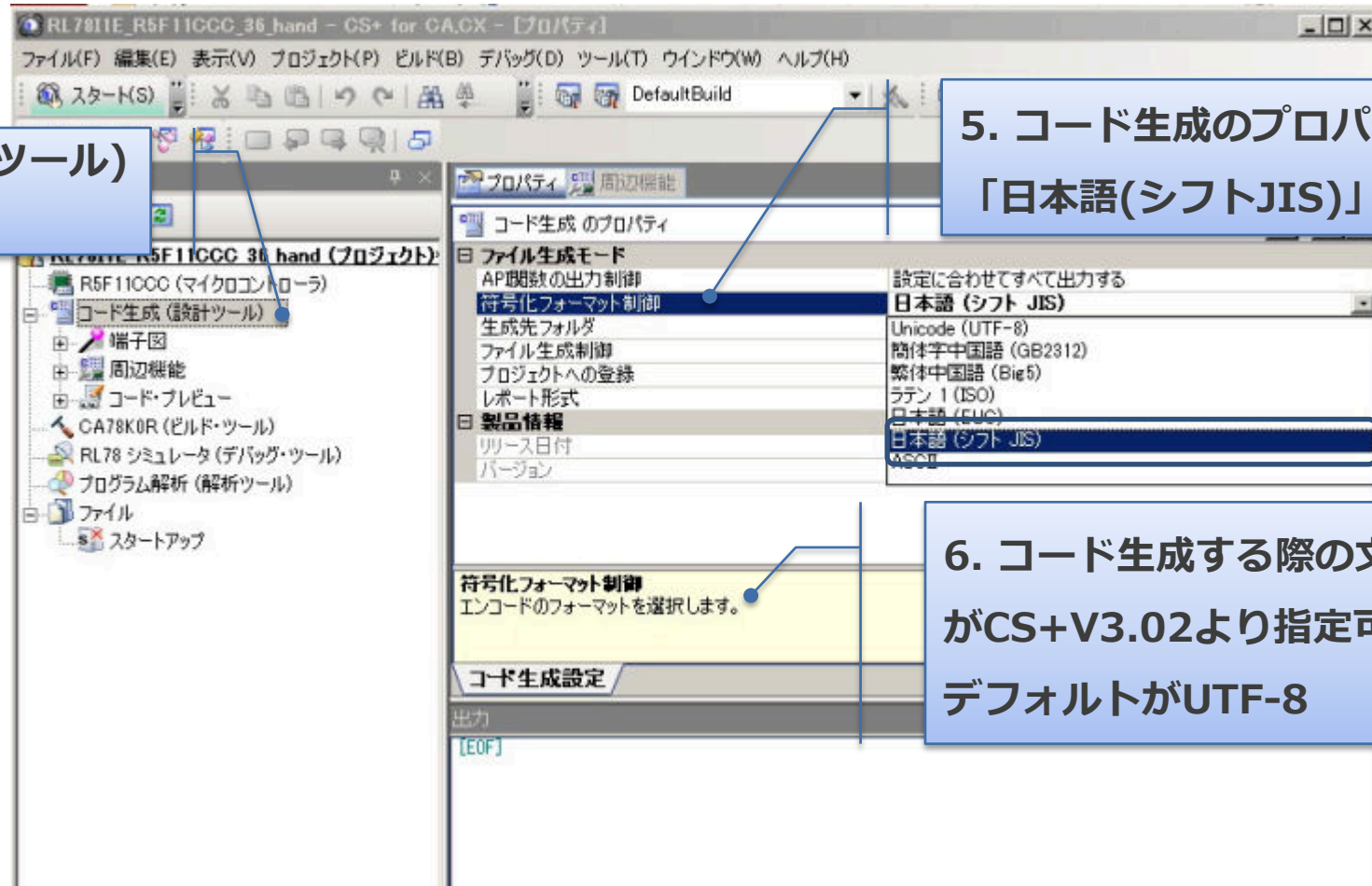
※ 開発環境としてCS+ for CA, CXで作成していますが、CS+ for CCでも同じ手順で作成できます。

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

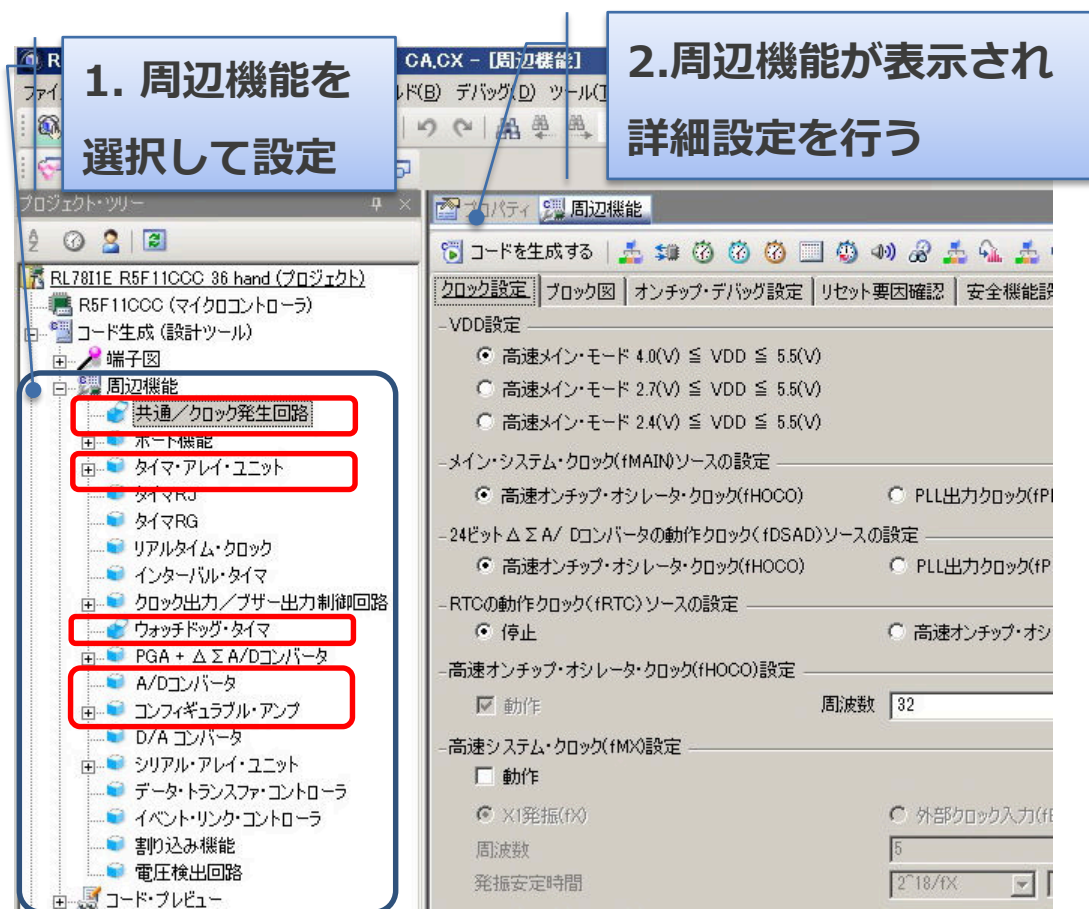
5. コード生成のプロパティを開き
「日本語(シフトJIS)」を選択

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)



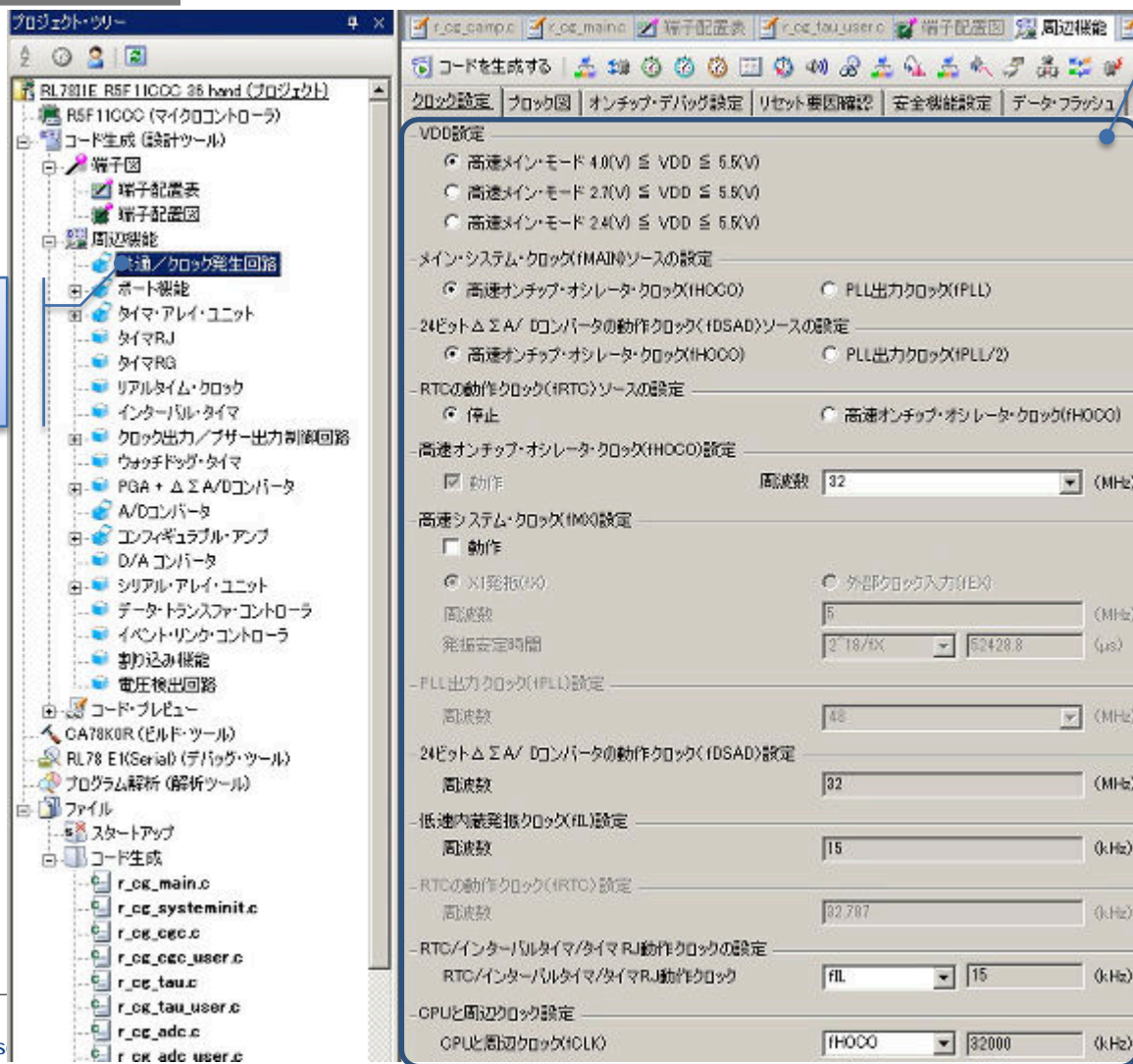
このデモで設定する周辺機能

- ・ **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

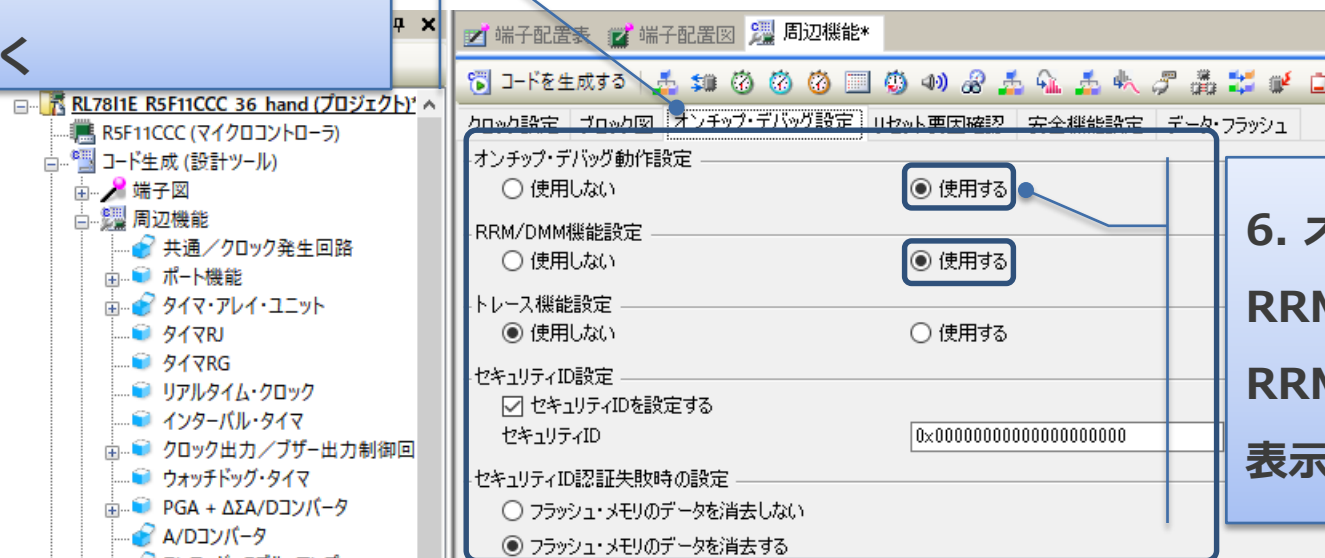


4. デフォルトで使用する
ので変更しない

コード生成で周辺機能設定

(オンチップ・デバッグ設定)

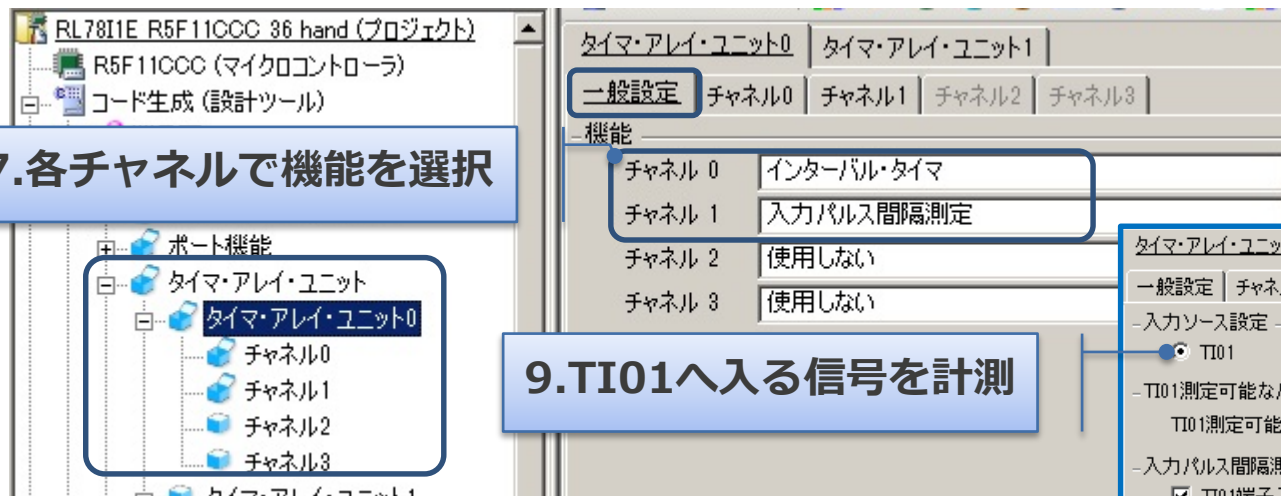
5. オンチップデバッグ設定
タブを開く



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

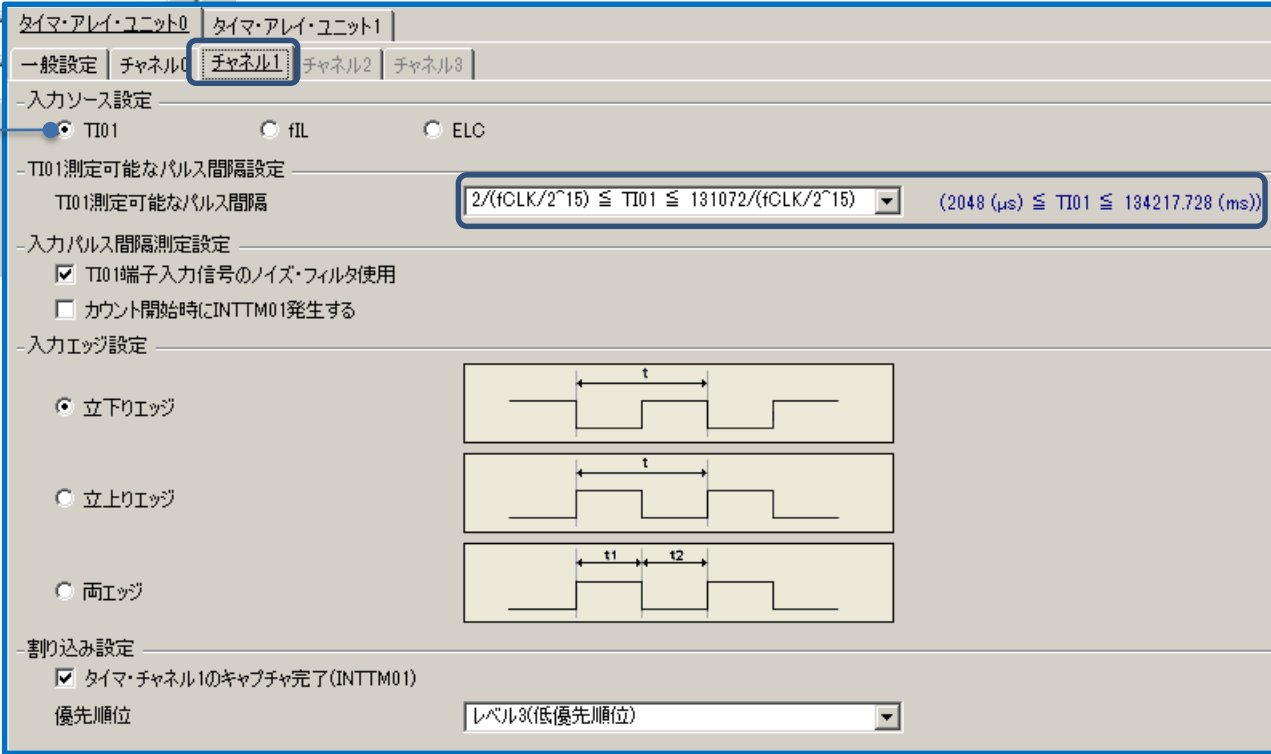
コード生成で周辺機能設定 (タイマ・アレイ・ユニットの設定)

7.各チャンネルで機能を選択

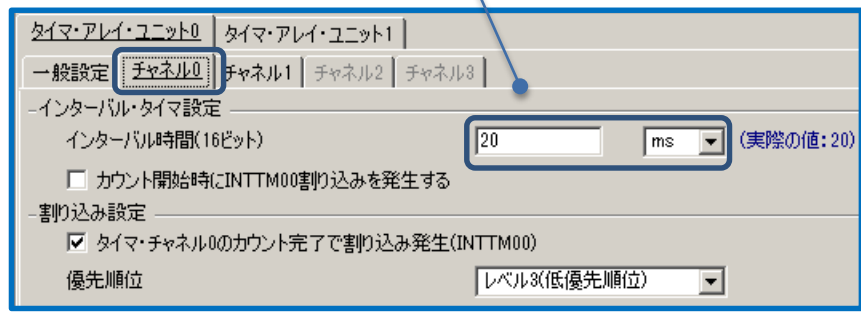


インターバル・タイマ20ミリ秒A/D変換の間隔、入力パルス間隔測定は脈拍のパルス間隔を計測します。

9.TI01へ入る信号を計測



8.チャンネル0を20ミリ秒



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)

10.ウォッチドッグタイマは使用しない

プロパティ 周辺機能*

コードを生成する

-ウォッチドッグ・タイマ動作設定

使用しない 使用する

-HALT/STOP/SNOOZEモード時の動作設定

許可 停止

-オーバーフロー時間設定

オーバーフロー時間 (ms)

-ウィンドウ・オープン期間設定

ウィンドウ・オープン期間 (%)

-割り込み設定

オーバーフロー時間の75% + 1/2fIL到達時に(インターバル)割り込みを発生する(INTWDTI)

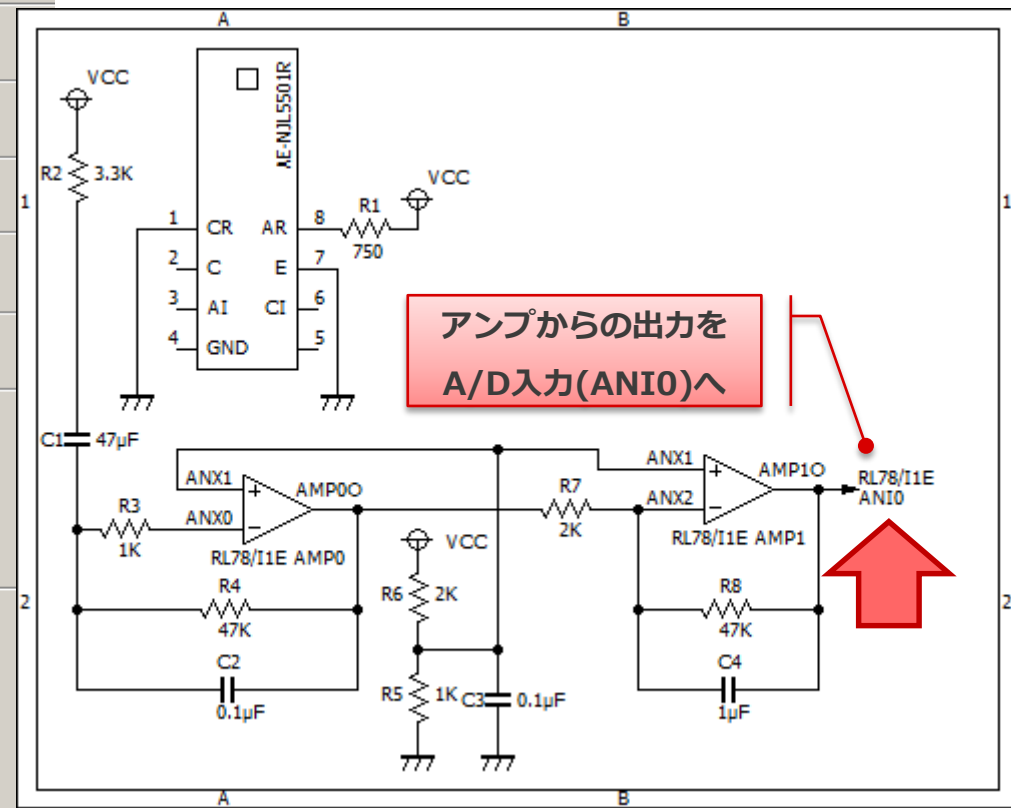
優先順位

コード生成で周辺機能設定 (A/Dコンバータの設定)

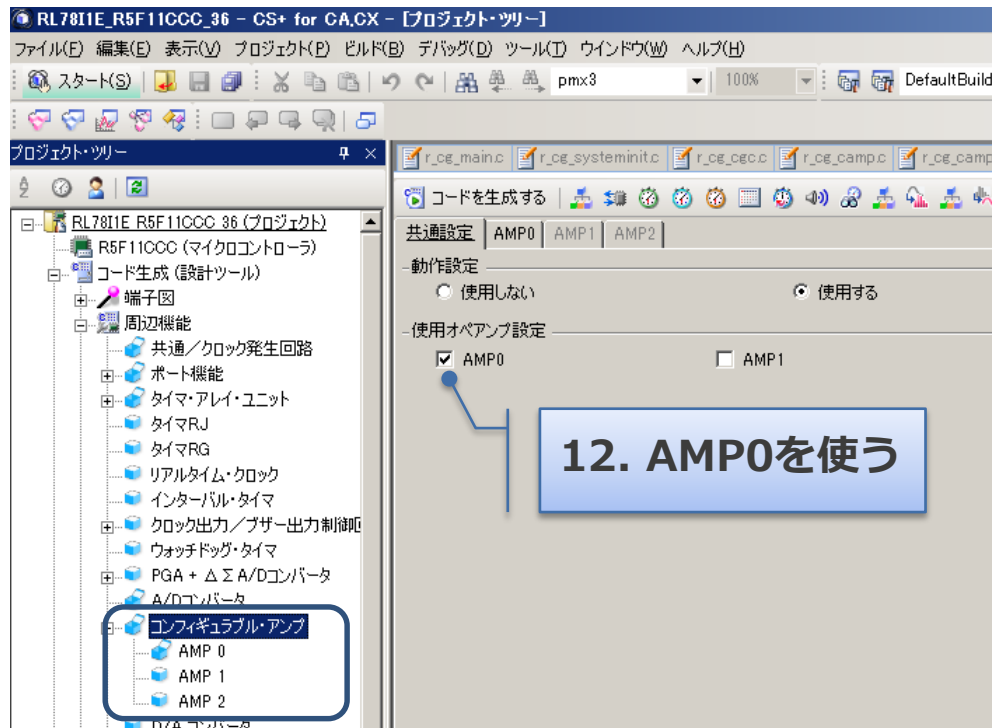
11. A/Dの設定

The screenshot shows the configuration interface for the A/D converter. The left sidebar lists various peripheral functions, with 'A/Dコンバータ' (A/D Converter) selected. The main configuration area is divided into several sections:

- A/Dコンバータ動作設定:** 使用する (Use)
- コンパレータ動作設定:** 許可 (Allow)
- 分解能設定:** 10 ビット (10 bits)
- VREF(+/-)設定:** AVDD (AVDD)
- トリガ・モード設定:** ソフトウェア・トリガ・モード (Software trigger mode)
- 動作モード設定:** 連続セレクト・モード (Continuous select mode)
- アナログ入力端子設定 (高精度チャンネル):** ANI0 (Selected)
- 変換開始チャンネル設定:** ANI0



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)



13. AMP0の設定

共通設定 AMP0 AMP1 AMP2

動作モード設定
 ノーマル・モード ハイスピード・モード

スイッチ設定

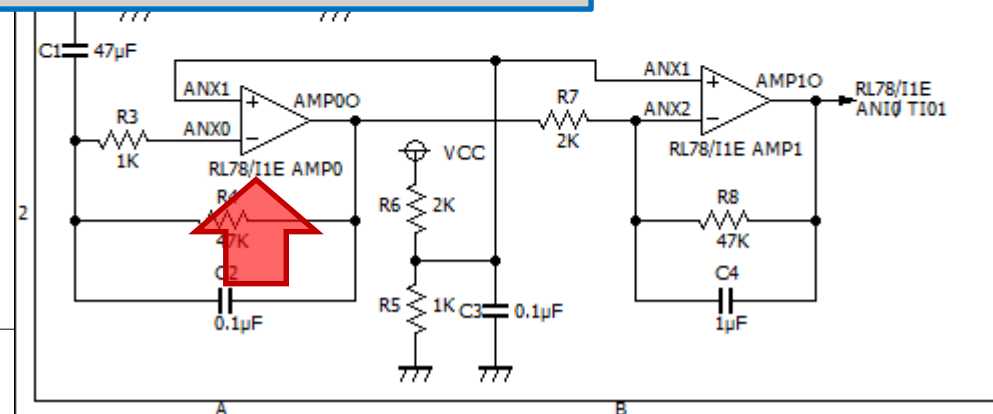
+入力をANX1へ設定

-入力をANX0へ設定

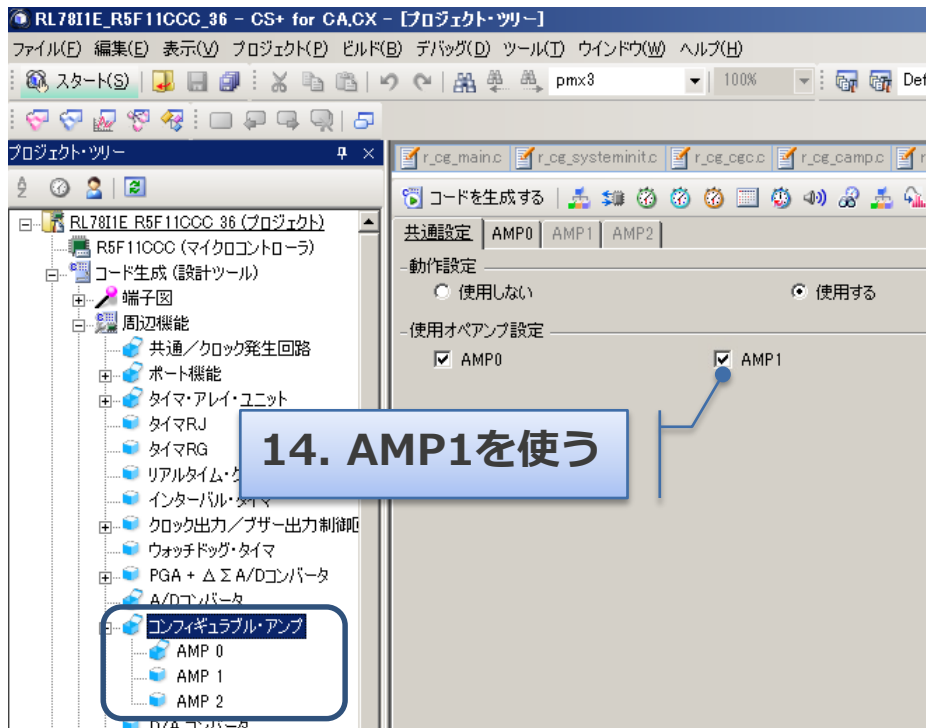
+入力設定
 ANX1 ANX0 D/Aコンバータ

-入力設定
 ANX1 ANX0 ボルテージ・フォロア回路を構成

出力設定
 ANX1 ANX0



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)



14. AMP1を使う

15. AMP1の設定

動作モード設定: ノーマル・モード

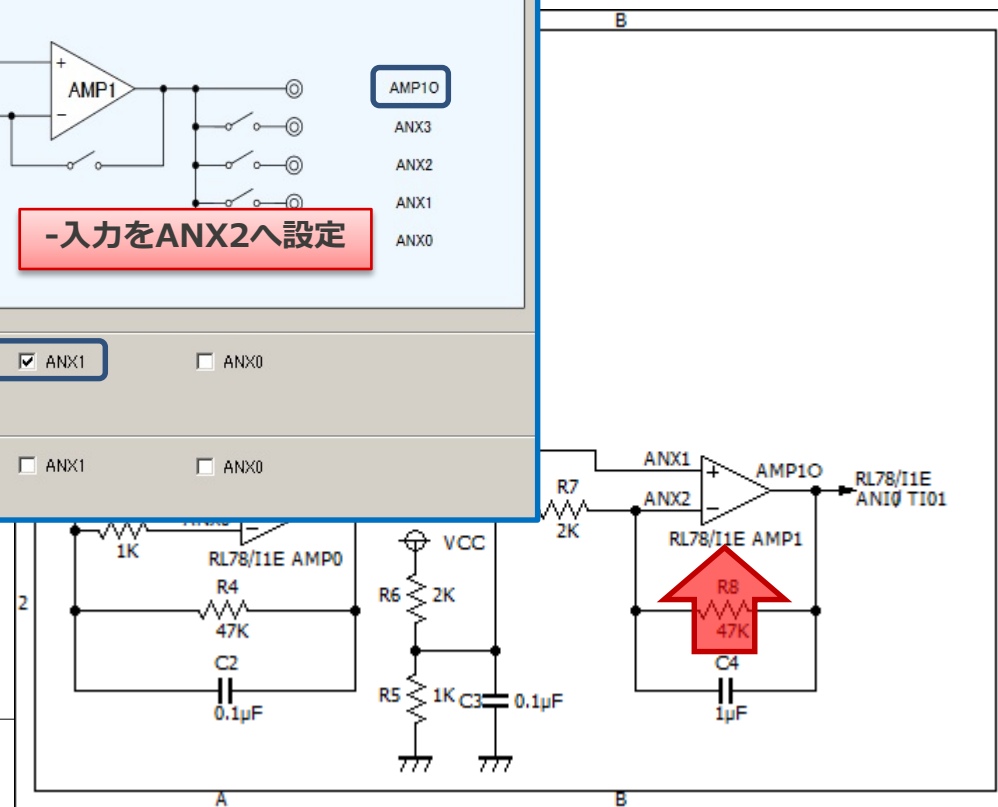
スイッチ設定

+入力をANX1へ設定

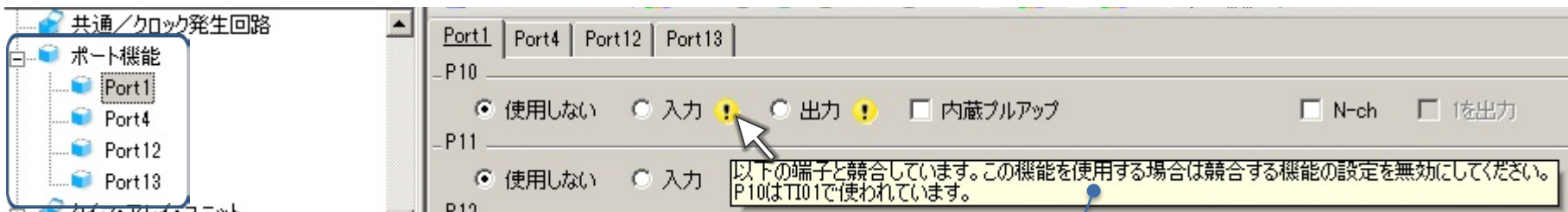
-入力をANX2へ設定

+入力設定: ANX3, ANX2, ANX1, ANX0

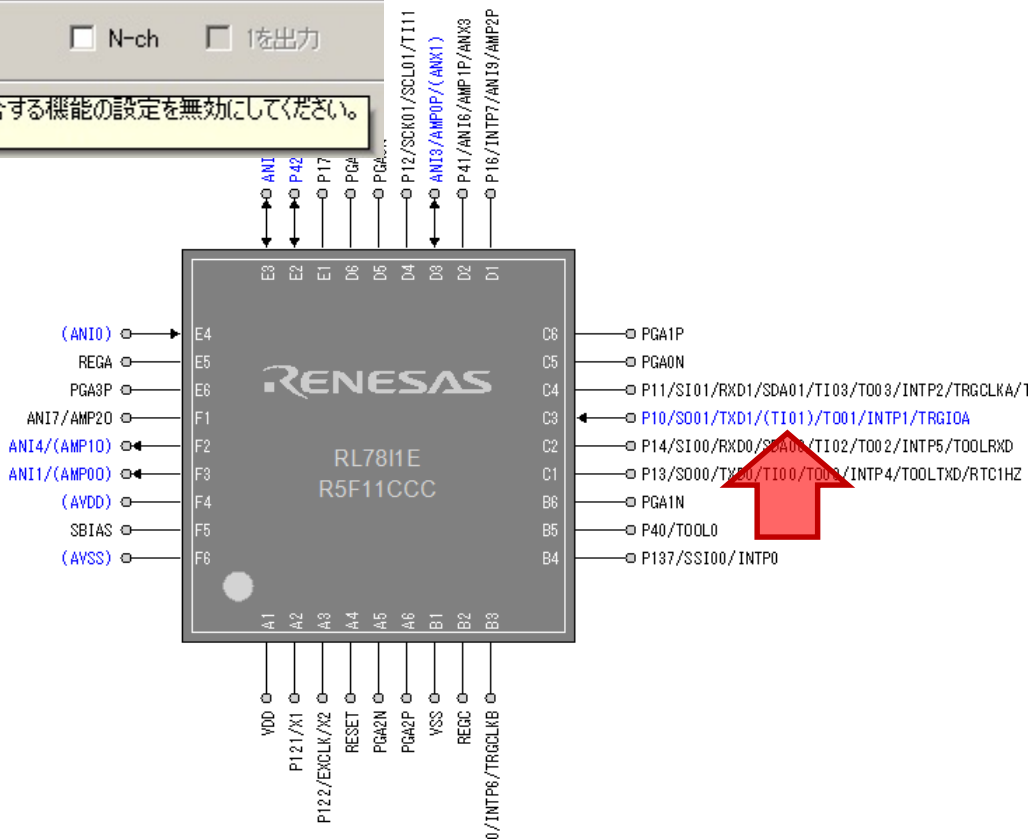
-入力設定: ANX3, ANX2, ANX1, ANX0



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

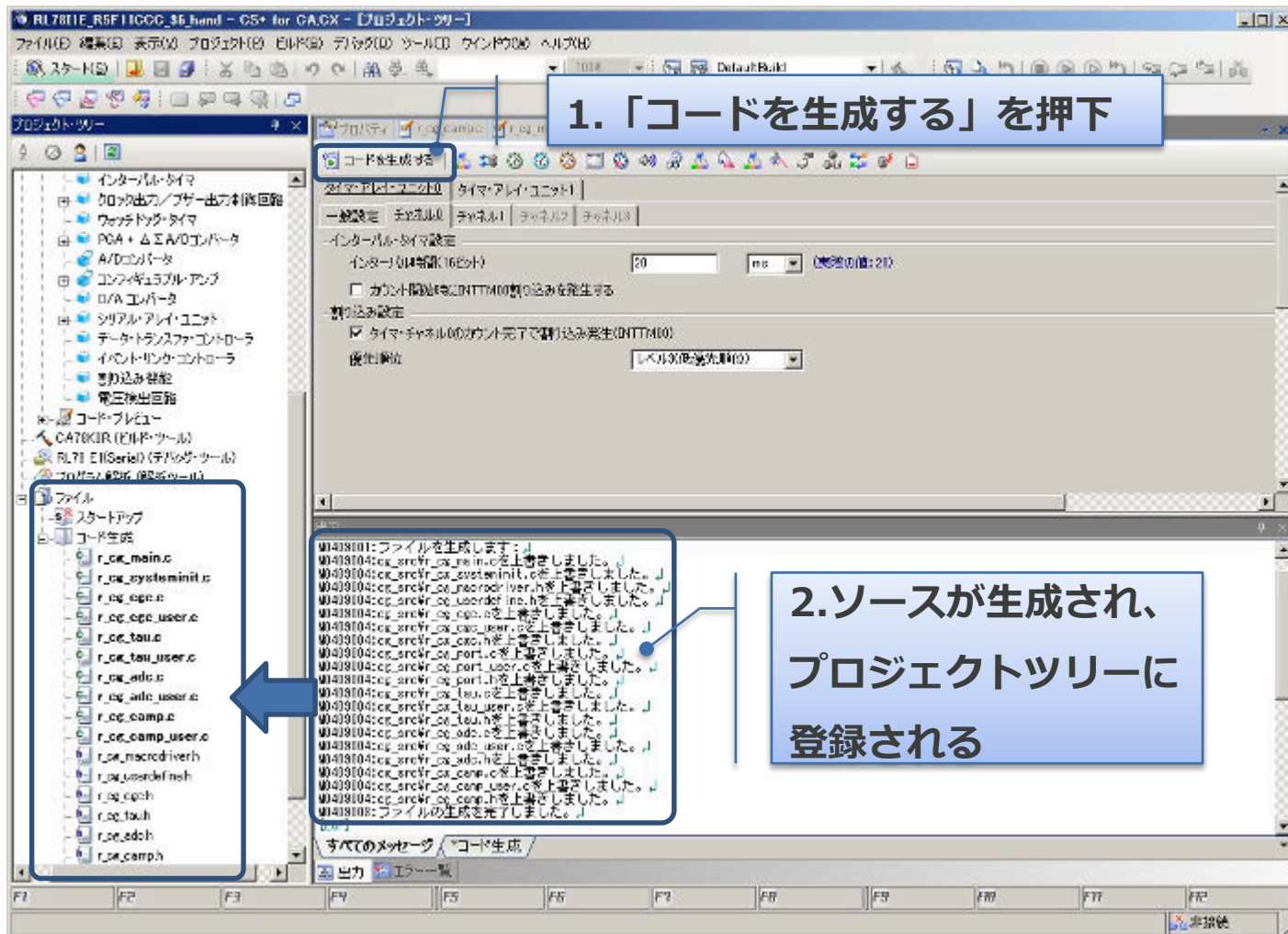
20. A/Dコンバータの状態を表示

端子名	端子割り当て	端子番号	入出力	備考
AMP0N	-	設定されてい...	-	
AMP0O	ANI1/AMP0O	F3	出力	
AMP0P	-	設定されてい...	-	
AMP1N	-	設定されてい...	-	
AMP1O	ANI4/AMP1O	F2	出力	
AMP1P	-	設定されてい...	-	
AMP2N	-	設定されてい...	-	
AMP2O	-	設定されてい...	-	
AMP2P	-	設定されてい...	-	
ANX0	ANI2/AMP0N/ANX0	E3	入力/出力	
ANX1	ANI3/AMP0P/ANX1	D3	入力/出力	
ANX2	P42/ANI5/AMP1N/A...	E2 (F42)	入力/出力	
ANX3	-	設定されてい...	-	
ANX4	-	設定されてい...	-	
ANX5	-	設定されてい...	-	

端子名	端子割り当て	端子番号	入出力
ANI0	ANI0	E4	入力
ANI1	-	設定されてい...	-
ANI2	-	設定されてい...	-
ANI3	-	設定されてい...	-
ANI4	-	設定されてい...	-
ANI5	-	設定されてい...	-
ANI6	-	設定されてい...	-
ANI7	-	設定されてい...	-
ANI8	-	設定されてい...	-
ANI9	-	設定されてい...	-
AVDD	AVDD	F4	-
AVSS	AVSS	F6	-

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はありません。

1.ダブルクリックで
r_cg_main.c の編集

2.ユーザが記述可能なプラグマ
命令をこのコメント間に書く

3.追加したい#include
をこのコメント間に書く

4.グローバル変数定義は、
このコメント間に書く

5.mainは(), コード生成が出力

6.メインの処理を
このコメント間に記述

7.ユーザの初期化処理を
このコメント間に記述

8.ユーザの追加する関数を
このコメント間に記述

```
*****
* File Name      : r_cg_main.c
* Version       : Code Generator for RL78/I1E V1.02.00.06 [12 Aug 2015]
* Device(s)    : R5F11CC0
* Tool-Chain   : CA78K0R
* Description   : This file implements main function
* Creation Date:
*****
/****
* D
* T
* N
* S
* T
* C
* O
* N
* T
* I
* N
* C
* O
* N
* T
/****
Pragma direct
*****
/* Start user code for pragma. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
/****
*****
Includes
*****
#include "r_cg_macrodriver.h"
#include "r_cg_cfg.c"
#include "r_cg_port.h"
#include "r_cg_tau.h"
#include "r_cg_adc.h"
#include "r_cg_camp.h"
/* Start user code for include. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
#include "r_cg_userdefine.h"
/****
*****
Global variables and functions
*****
/* Start user code for global. Do not edit comment generated here */
/* End user code. Do not edit comment generated here */
Pr
****

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 */
    EI();
    /* 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 */
```

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();   // タイマチャンネル0開始
    R_TAU0_Channel1_Start();   // タイマチャンネル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  
***** */
```

A/D変換終了割り込み関数
(38μ秒ごとに変換が完了)

```
__interrupt static void r_adc_interrupt(void)
```

```
{  
/* Start user code. Do not edit comment generated here */  
uint8_t i;
```

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

```
R_ADC_Stop();  
R_ADC_Get_Result( &gAdResult );
```

A/D変換値の取得

```
// 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 ] = gAdResult[ i - 1 ];  
}  
gtAdResult[ 0 ] = gAdResult;
```

defineで定義された個数分、A/D変換結果を
バッファへ代入

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

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 INTTMO0 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、
 $820 \times (977 / 1) = 0.839$ 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_interrupt
* Description : This function INTTMO1 interrupt service
* 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 comment generated here */
}

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

```

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
動作クロック (CKmk) の選択 ¹⁾ (k = 0, 1)																
PRS	PRS	PRS	PRS					fclk =	fclk =	fclk =	fclk =	fclk =				
mk3	mk2	mk1	mk0					2 MHz	4 MHz	8 MHz	20 MHz	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 INTFM01 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
        P4.3を1にしている間
        の処理を波形で観測
        );
    }
    /* Start user code
    /* 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

```

// 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;

```

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。

P4.3の波形を観測

デバッグツールの設定 (E1エミュレータの設定)

1. デバッグツールはE1を選択し、プロパティを開く

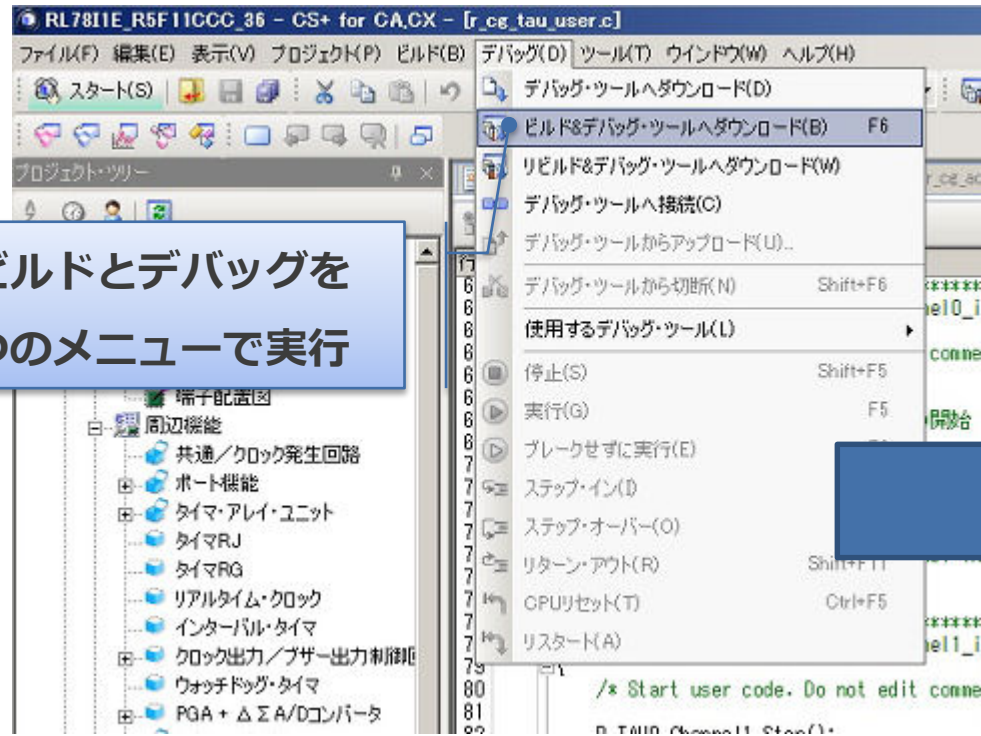
2. E1から5Vを電源供給する

3. デバッグ実行中の表示更新を100msごとに行う

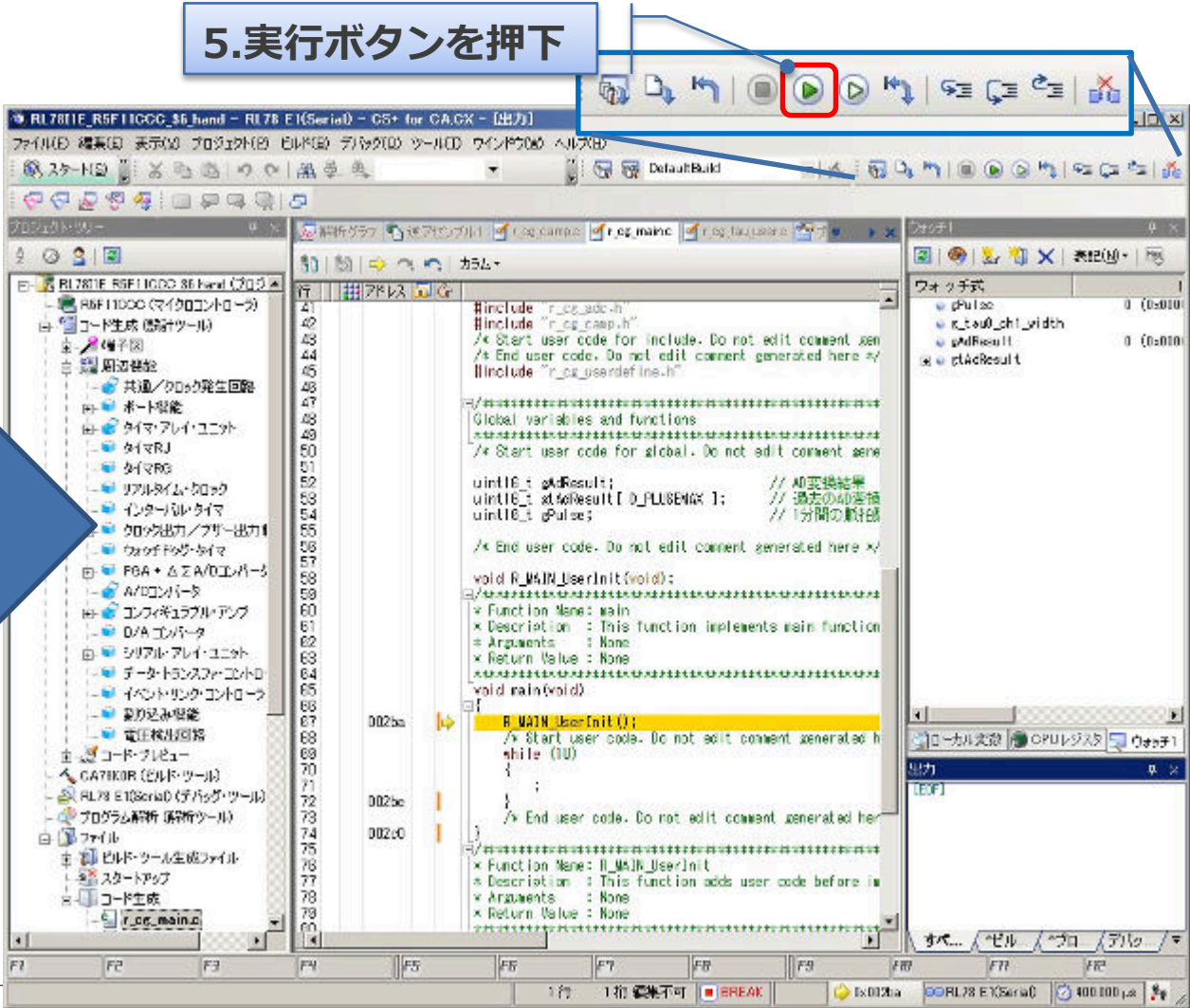
項目	設定
メモリ	
メモリ・マッピング	[10]
メモリ書き込み時にペリファイを行う	はい
実行中のメモリ・アクセス	はい
実行を一瞬停止してアクセスする	はい
実行中に表示更新を行う	はい
表示更新間隔[ms]	100
ブレーク	
トレース	
入力信号のマスク	
TARGET RESET信号をマスクする	いいえ
INTERNAL RESET信号をマスクする	いいえ
Smart Analog	
実行中にデータ収集を行う	<input checked="" type="checkbox"/> いいえ

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)

The screenshot displays the CubeSuite+ IDE interface. On the left is the Project Explorer showing a tree view of hardware components like 'R5F110CC (マイクロコントローラ)'. The main window shows a C program with various include statements and function definitions. A context menu is open over the code, with options like 'ウォッチ1に登録(R)', '解析グラフに登録(E)', and 'アクション・イベントの登録(A)...'. On the right, the Watch window shows a table of variables being monitored:

ウォッチ式	値	型
gPulse	0 (0x0000) uns	
gAdResult	0 (0x0000) uns	
gtPulseResult	??	
g_tau0_ch1_...	0 (0x00000000) uns	

1.gAdResult変数をウォッチ1と解析グラフに登録

2.マウスカースールを変数に合わせて右クリックでメニューを表示。ウォッチ1へ登録。gPulseとgtAdResultも同様にウォッチ1へ登録。

※ 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)

3. プログラム解析のプロパティを開く

4. 表示更新を100msごとに行うので、グリッド単位を調整する。

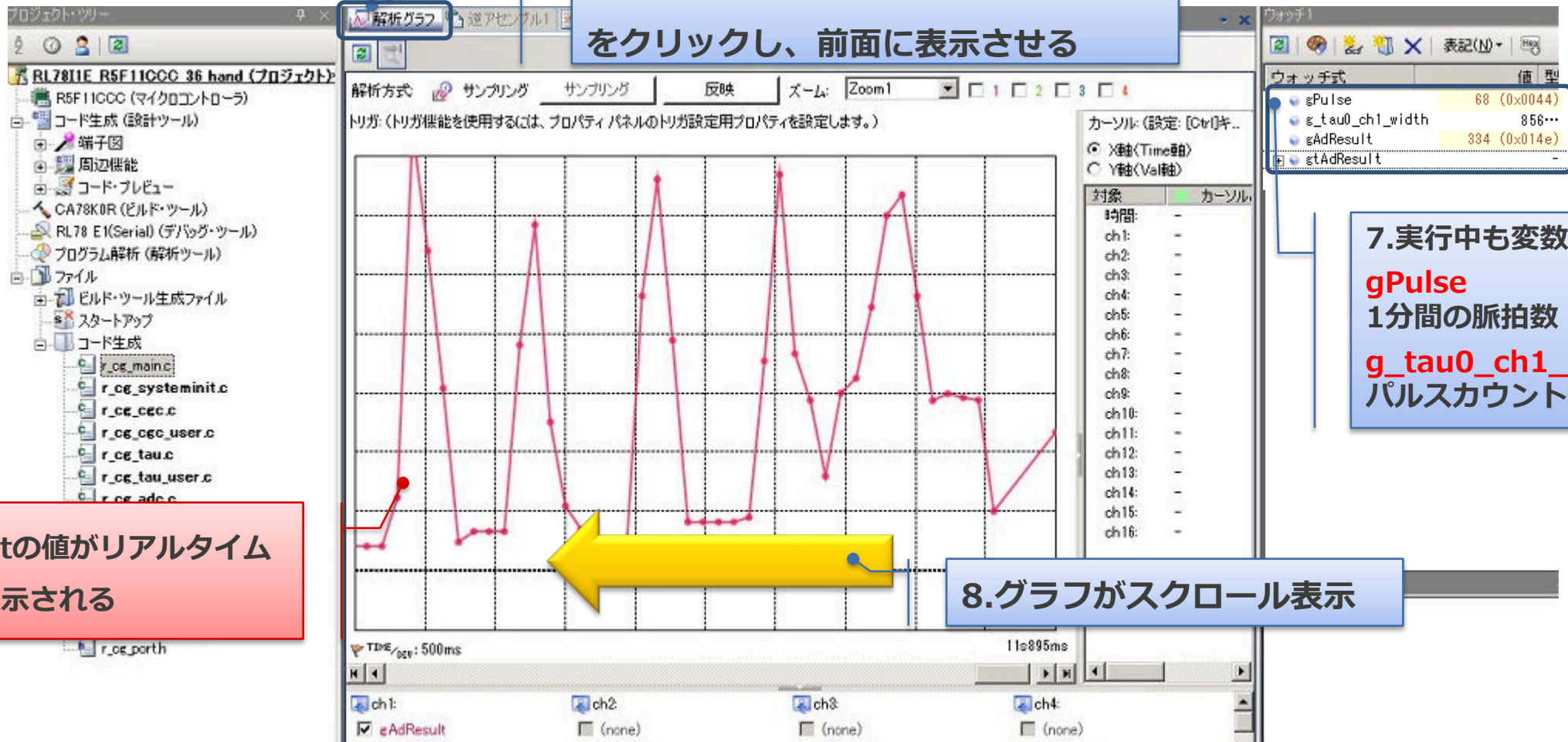
5. グラフが中央に表示されるようにオフセットを調整する

項目	設定値
解析方式	リアルタイム・サンプリング方式
サンプリング開始/停止	連動
自動調整	行わない
1グリッドあたりの時間[Time/Div]	500ms
グラフの種類	折れ線グラフ
背景色と前景色を指定する	い/い
カーソルAの色	PaleGreen
カーソルBの色	PaleTurquoise
ズーム枠1の色	64, 255, 10, 79
ズーム枠2の色	64, 91, 228, 22
ズーム枠3の色	64, 5, 109, 239
ズーム枠4の色	64, 255, 84, 28
トリガ	い/い
トリガ機能を使用する	い/い
チャンネル 1	
変数名/アドレス 1	εAdResult
型/サイズ 1	符号なし2バイト(16ビット)
1グリッドあたりの値[Val/Div] 1	100
オフセット 1	-300
色 1	192, 255, 10, 79
チャンネル 2	
変数名/アドレス 2	
型/サイズ 2	自動
1グリッドあたりの値[Val/Div] 2	25.5
オフセット 2	0
色 2	192, 91, 228, 22
チャンネル 3	
変数名/アドレス 3	
型/サイズ 3	自動

プログラムの実行

(プログラム実行中の表示)

6.プログラム実行中は解析グラフタブをクリックし、前面に表示させる



7.実行中も変数が更新される
gPulse
1分間の脈拍数
g_tau0_ch1_width
パルスカウント値

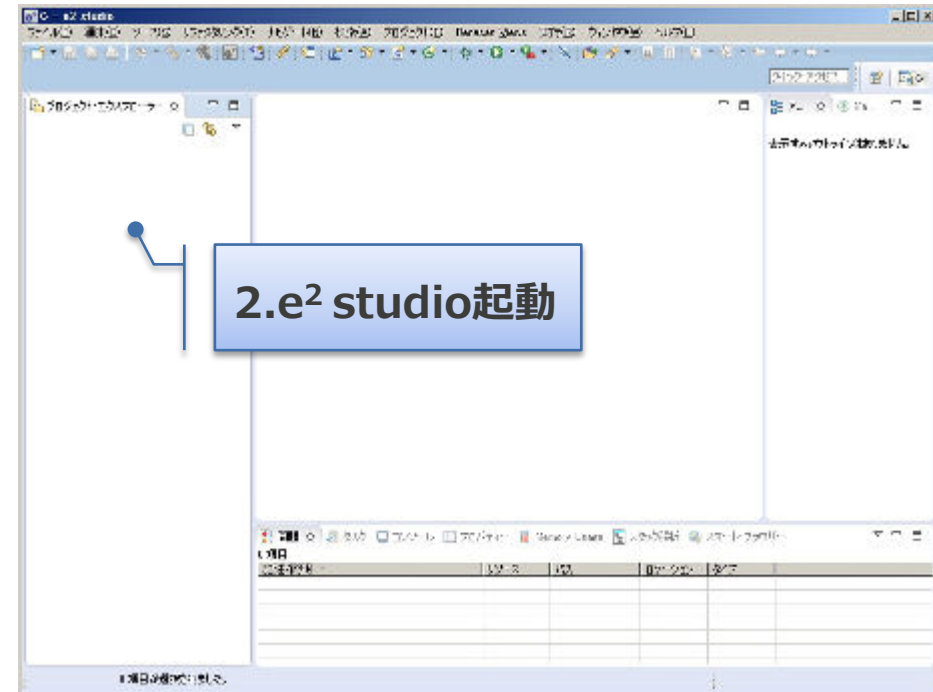
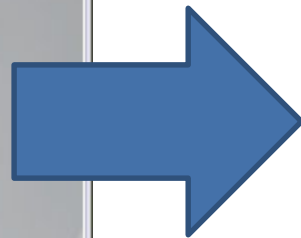
8.グラフがスクロール表示

gAdResultの値がリアルタイムでグラフ表示される

e² studioでプロジェクト作成



1. Workbenchを選択



2.e² studio起動

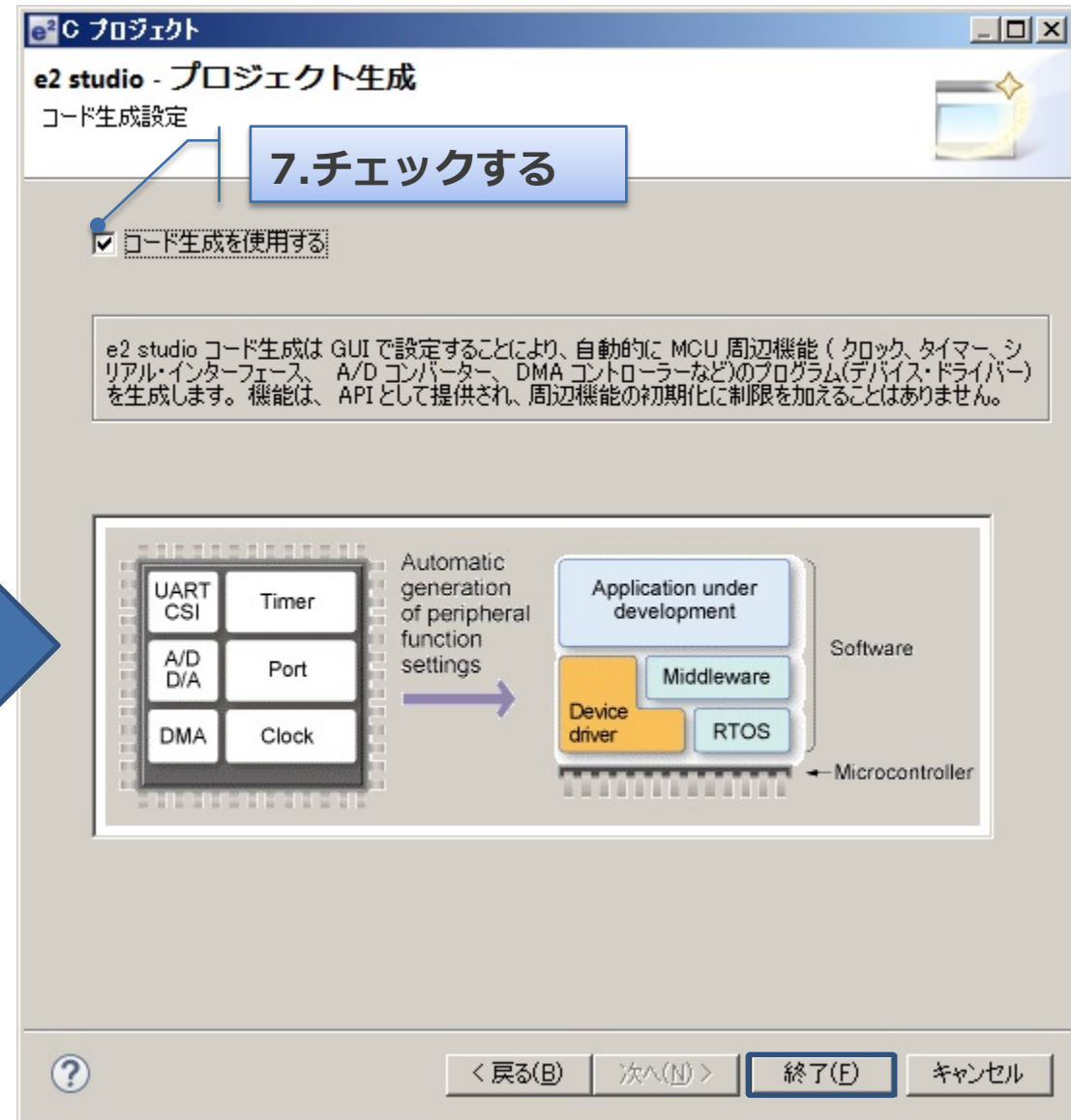
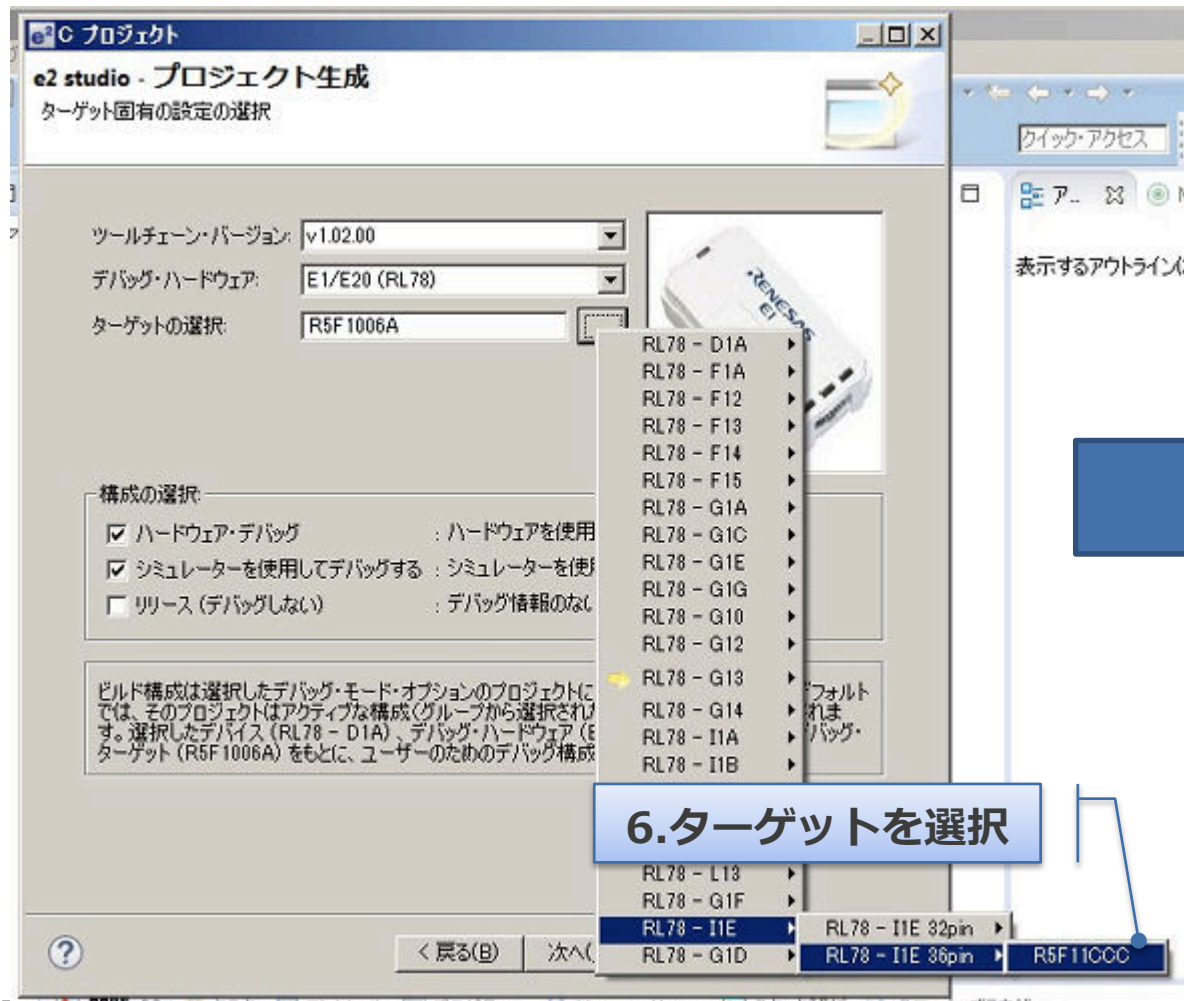
e² studioでプロジェクト作成

3. C Projectを新規作成

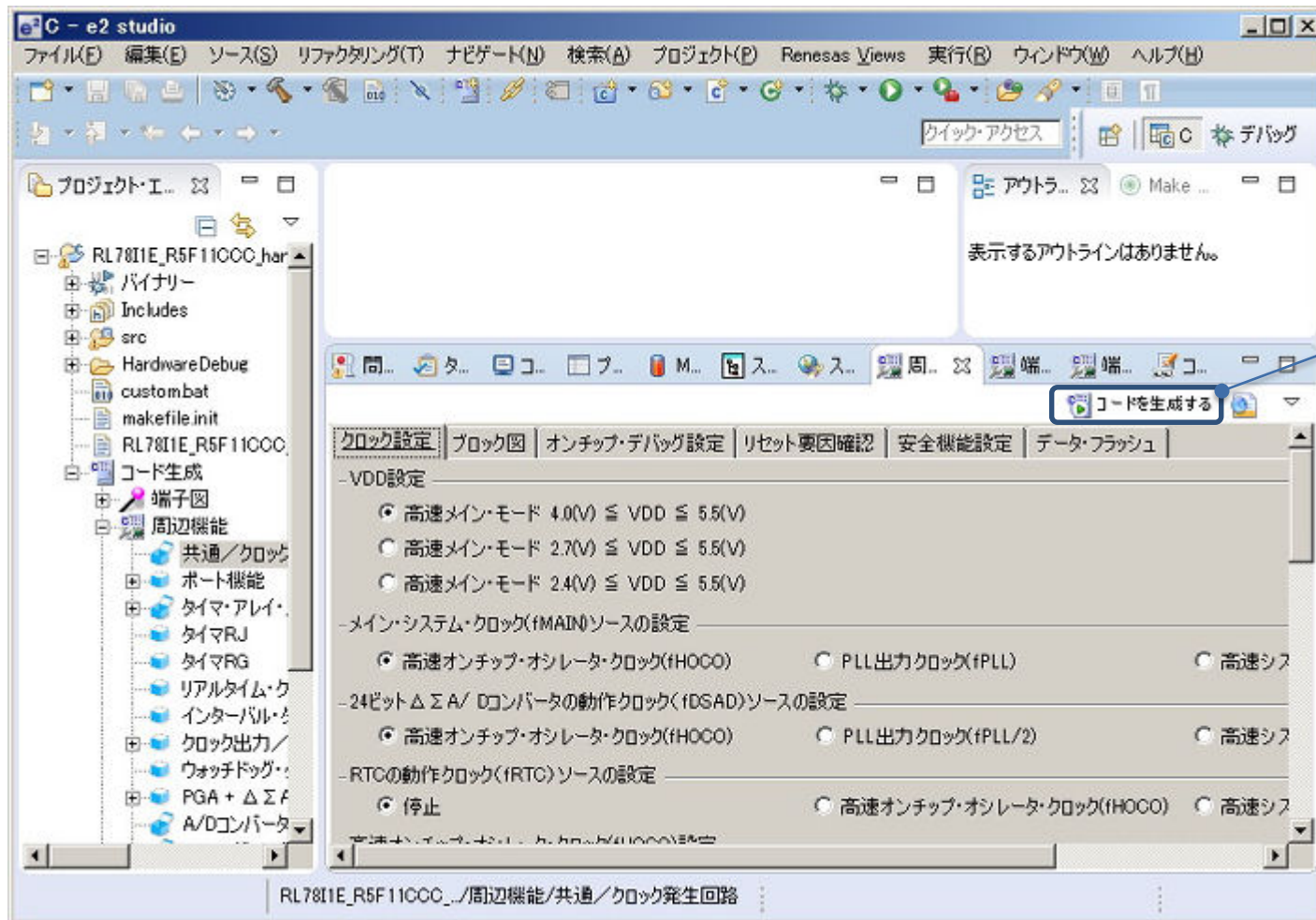
4. プロジェクト名、
作成場所を入力

5. ツールチェーンは
CC-RLを選択

e² studioでプロジェクト作成



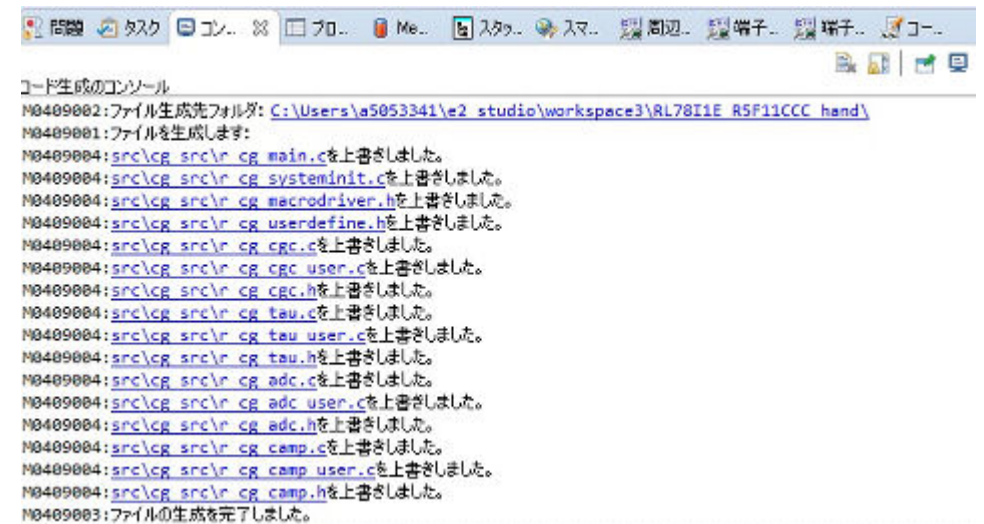
e² studioでプロジェクト作成



コード生成で設定する内容について

・ 9～19ページを参照

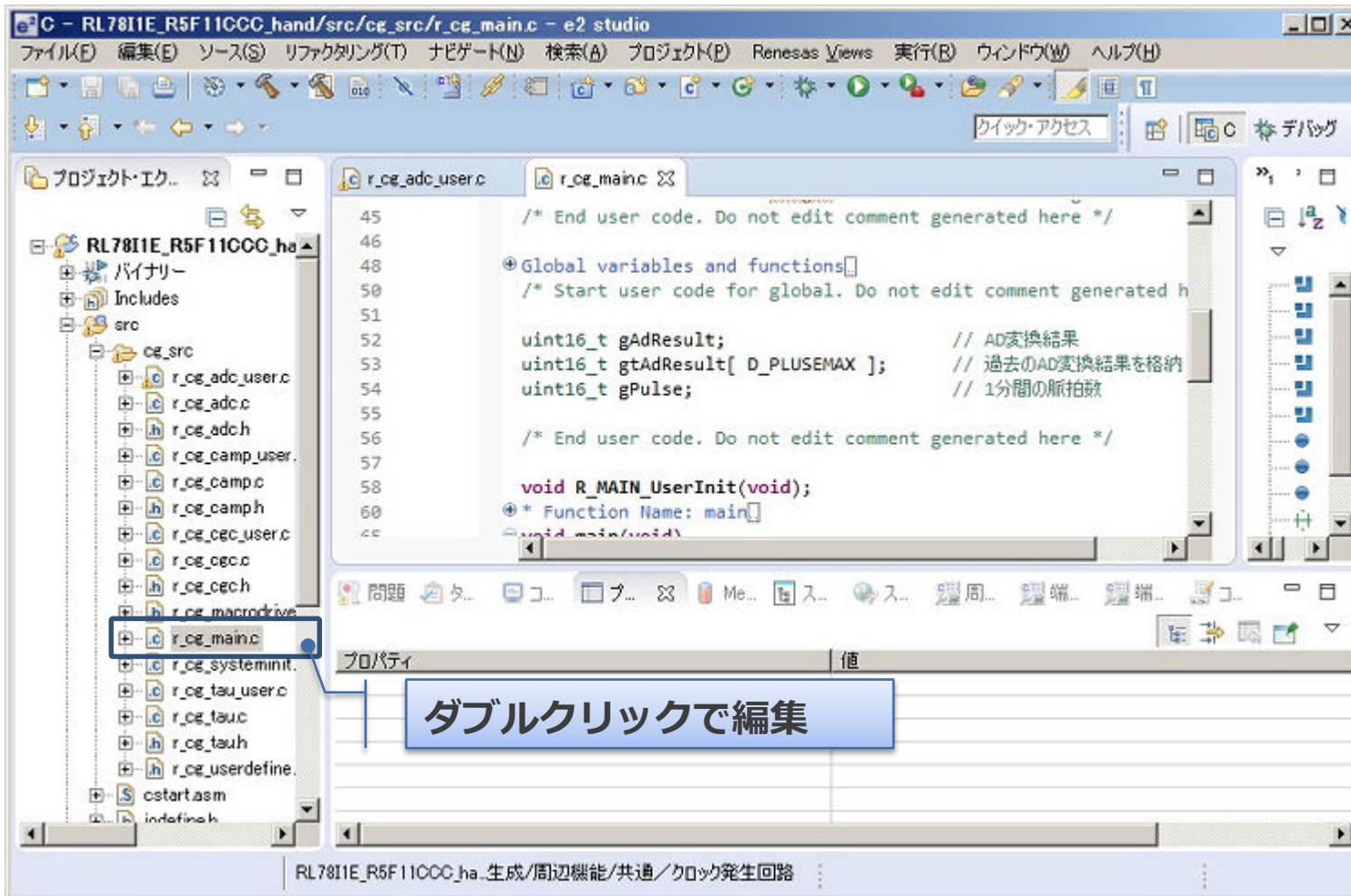
8. 「コードを生成する」を押下



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

1. ツール設定ボタンを押下

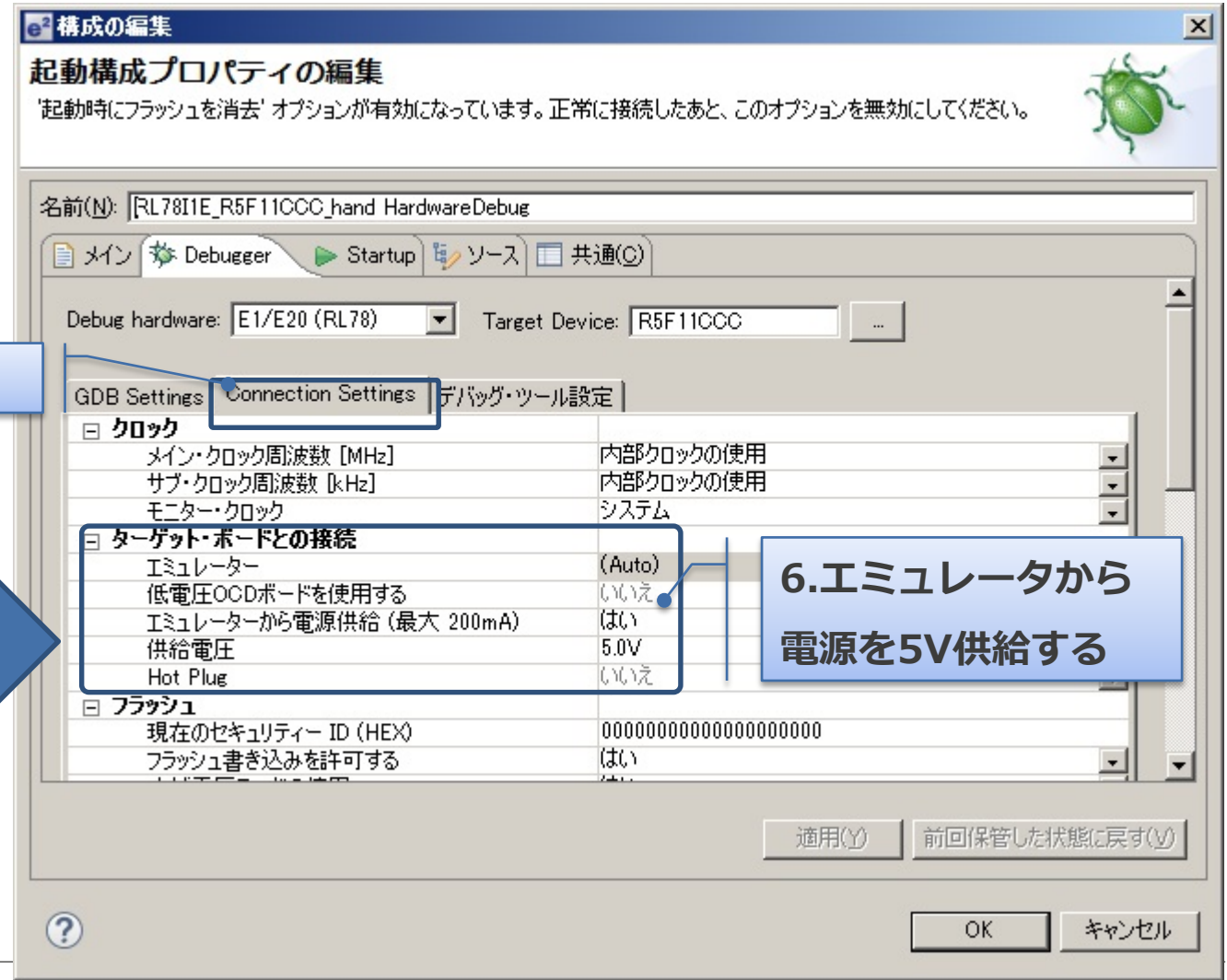
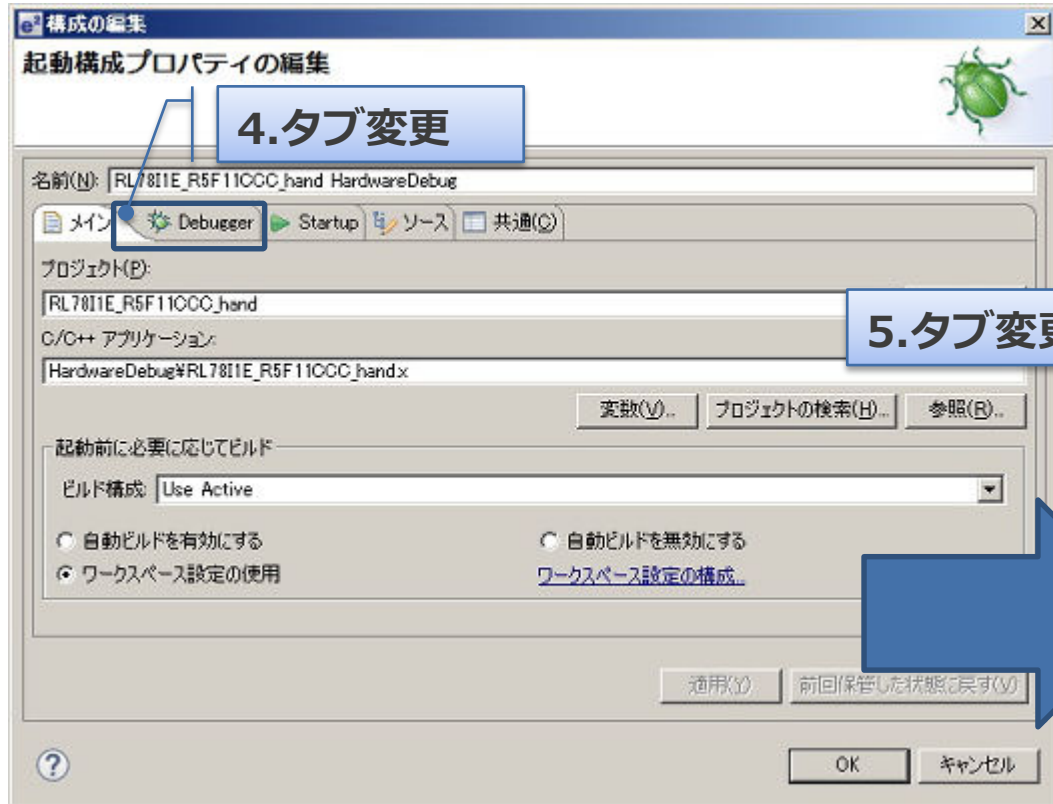
入力したプロジェクト名がアクティブになっていることを確認する

2. 実行/デバッグ設定

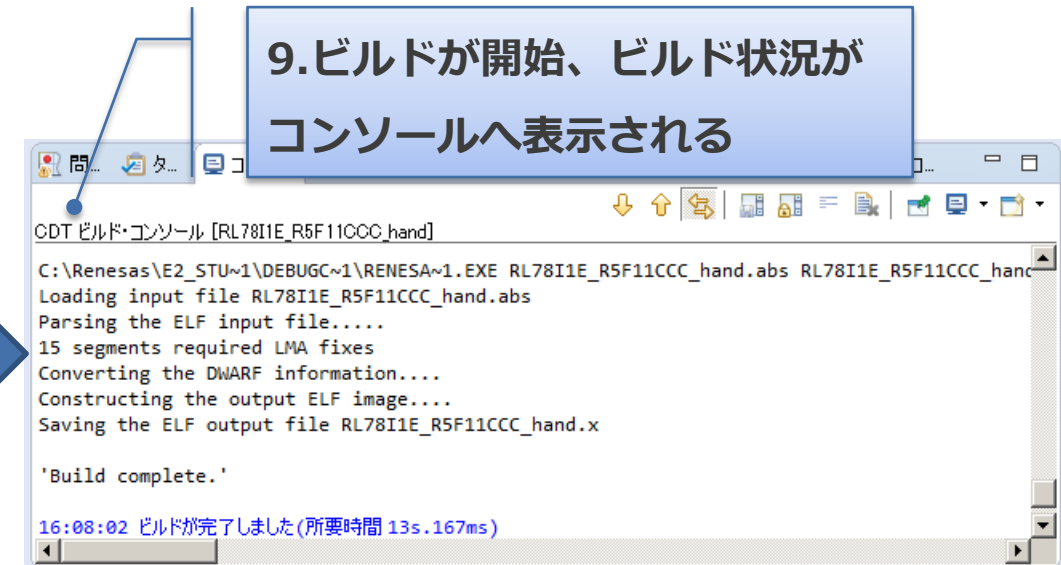
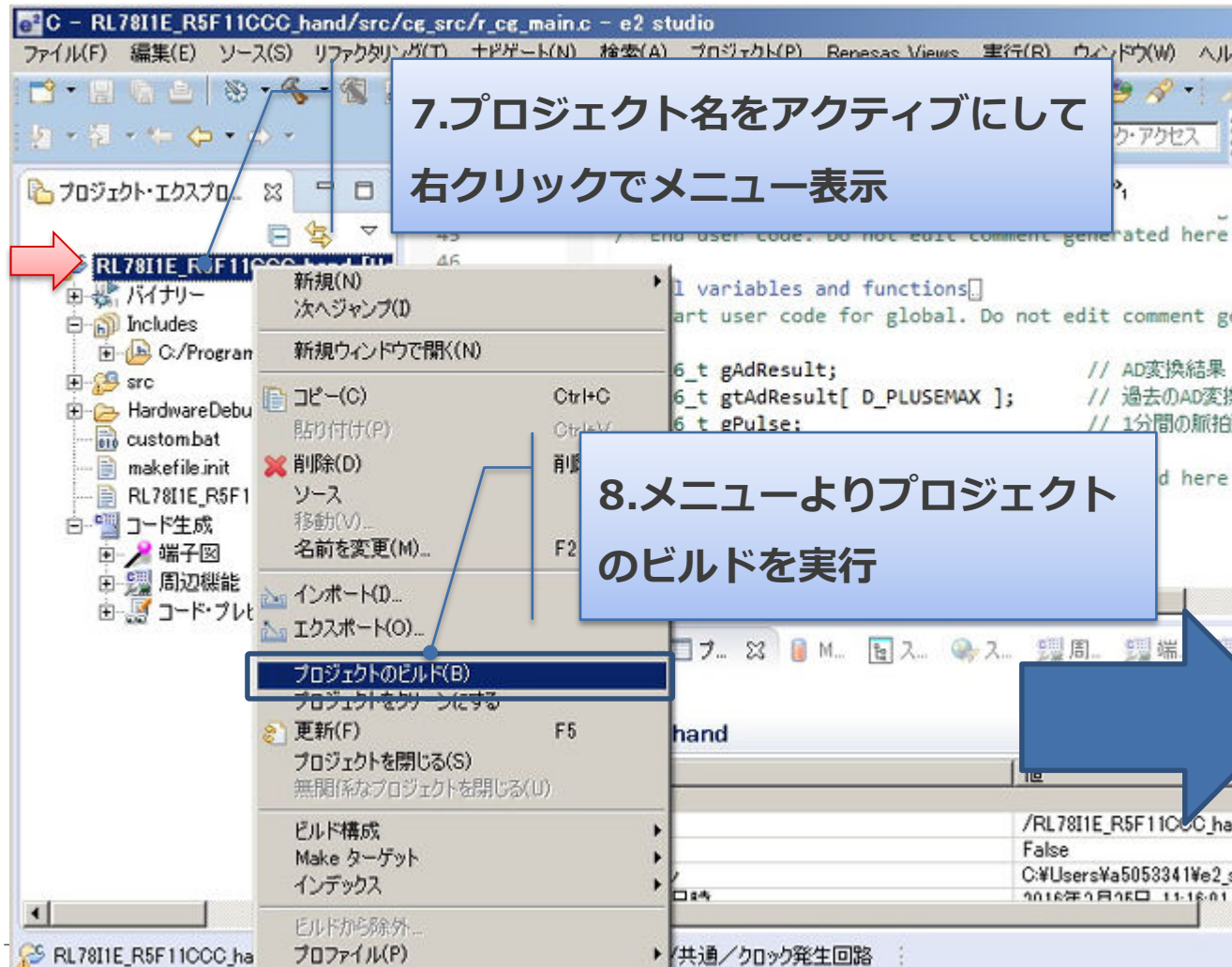
3. プロジェクトのデバッグ設定を編集

リソース	プロパティ	値
情報	パス	/RL78I1E_R5F11CCC_hand
	リンク	False
	ロケーション	C:\User
	生成/周辺機能/共通/クロック発生回路	0016CF

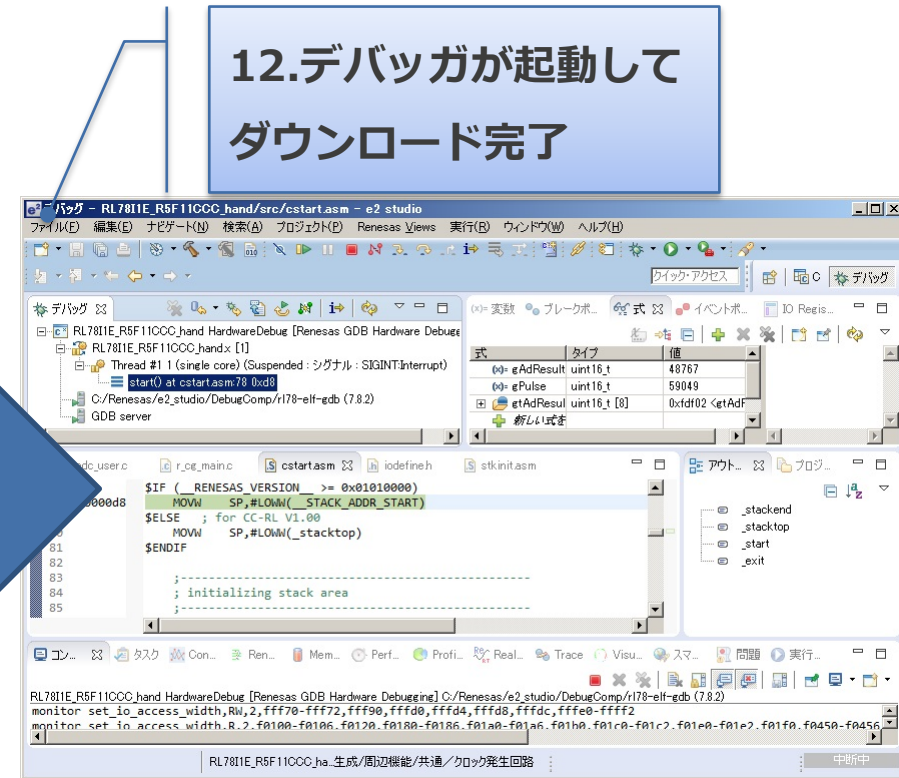
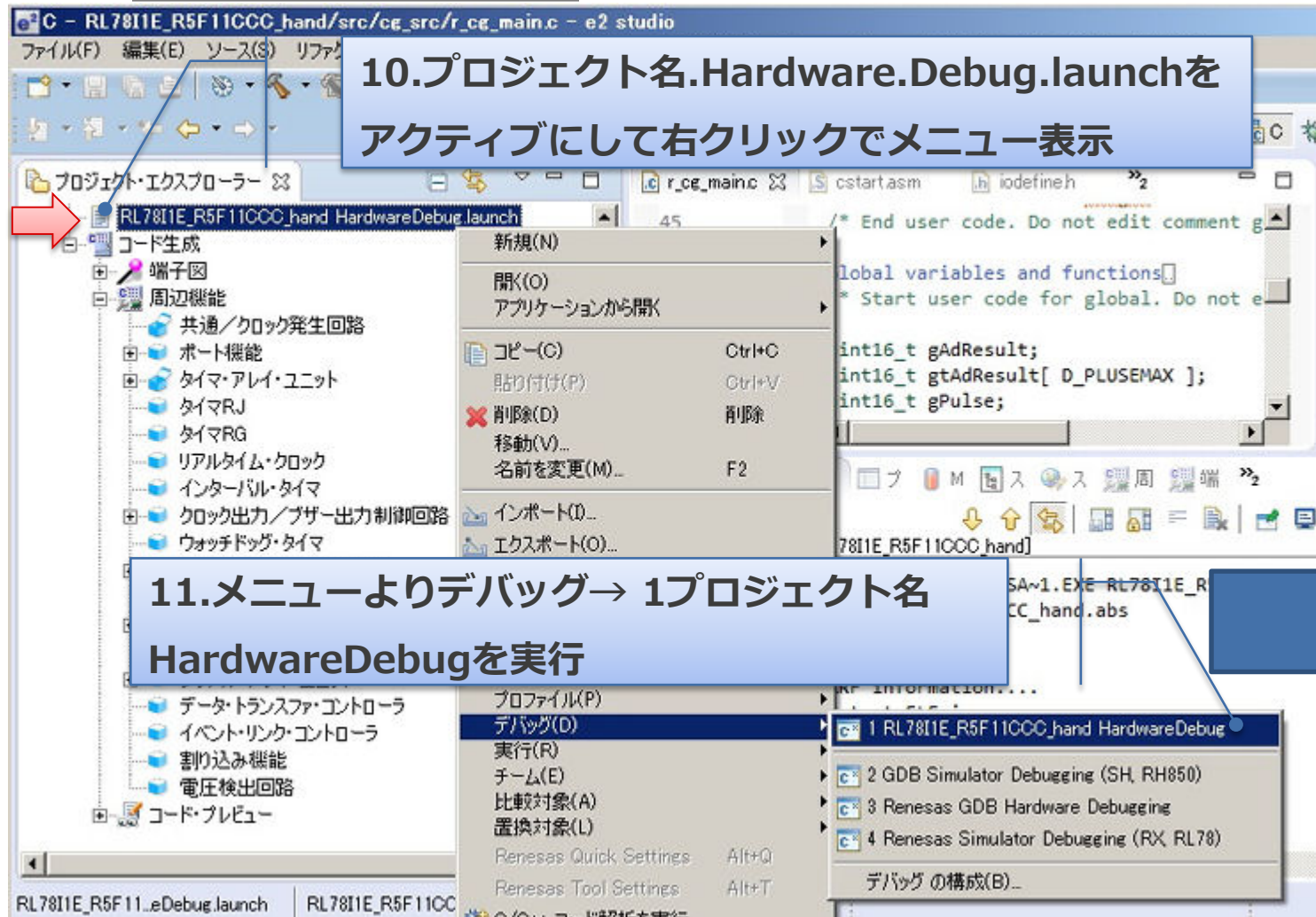
e² studioでデバッグツールの設定



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)

14. 選択した状態で右クリック
でメニューを表示

13. 変数をダブル
クリックして選択

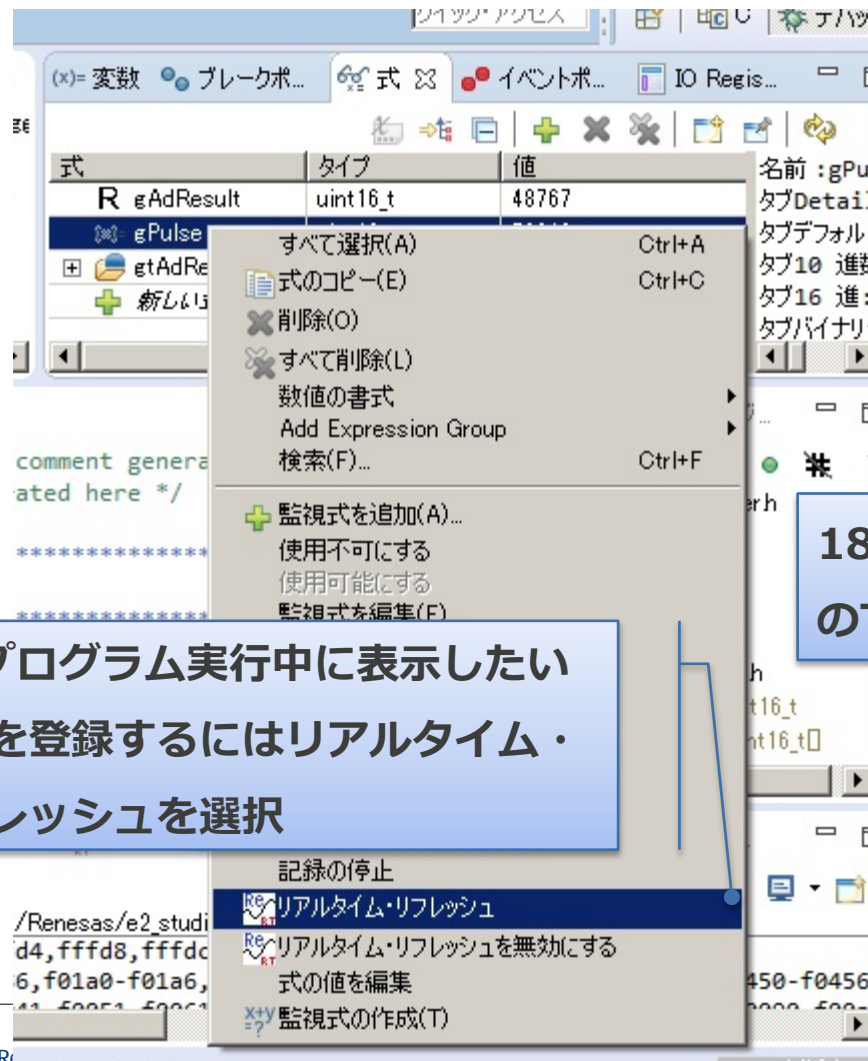
16.gAdResult, gPulse,
gtAdResultの各変数を
監視式として登録

15. メニューより監視式を追加
を選択

式	タイプ	値
gAdResult	uint16_t	48767
gPulse	uint16_t	59049
gtAdResult	uint16_t [8]	0xdf02 <gAdResult>
+ 新しい式を追加		

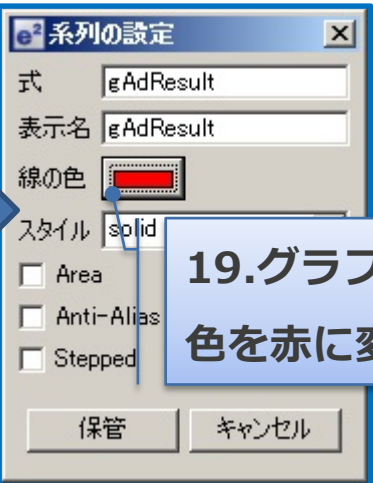
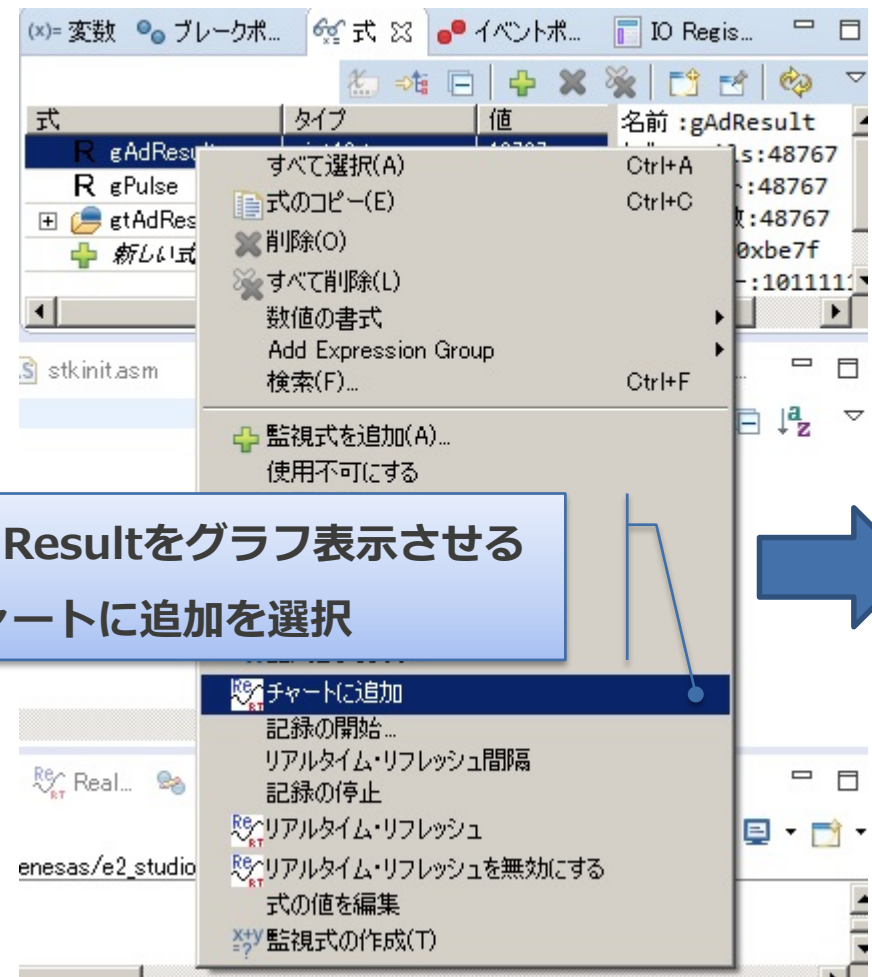
e² studioでデバッグツールの設定 (監視式の設定)

※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)

デバッグ - RL78I1E_R5F11CCC_hand/src/cstart.asm - e2 studio

再開(M) (F8 直接アクセス)

20.プログラムを実行

デバッグ - RL78I1E_R5F11CCC_hand/src/cg_src/r_cg_main.c - e2 studio

21.プログラム実行中でも変数表示が更新される

式	タイプ	値
R gAdResult	uint16_t	330
R gPulse	uint16_t	76
etAdResult	uint16_t [8]	0xdf02

```
r_cg_main.c  
{  
  R_MAIN_UserInit();  
  /* Start user code. Do not edit comment generated here */  
  while (1U)
```

Time (intervals)

gAdResult変数がリアルタイムでグラフ表示される

実行中

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

The screenshot shows the Renesas documentation search interface. At the top, there is a search bar with the text 'キーワード/品名/型名検索' and a '検索' button. Below the search bar is a navigation menu with items like '製品情報', 'アプリケーション', '開発環境', 'お問合せ/サポート', 'ご購入/サンプル', and '会社案内'. The main content area is titled 'ドキュメント検索' and features a search bar with the keyword 'r20ut3230jj'. The search results show one item: 'コード生成プラグイン学習ガイド' (Code Generation Plugin Learning Guide), with a release date of 'Oct.01.14 Rev.1.00' and a size of '4379 KB'. The title is highlighted with a red box. The page also includes a sidebar with filters for '製品をお選びください' and 'ドキュメントタイプをお選び下さい'.

タイトル	発行日 リビジョン	ソース etc.	ドキュメントNo. (旧ドキュメントNo.)	サイズ(KB)	対応 製品
コード生成プラグイン学習ガイド	Oct.01.14 Rev.1.00	-	R20UT3230JJ0100	4379	一覧

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	