RL78 ソフトウェア置き換えガイド

CA78K0R から CC-RL への移行編（CS+）

R01AN3100JJ0110
Rev.1.10
2017.09.29

要旨

本アプリケーションノートでは、開発統合環境 CS+ 用 C コンパイラ CA78K0R で作成されたソースコードを開発統合環境 CS+ 用 C コンパイラ CC-RL に対応したソースコードに置き換える方法について説明します。

C コンパイラの対象バージョンは以下の通りです。

・CA78K0R V1.20 以降

・CC-RL V1.01.00

対象デバイス

RL78/G13、RL78/G14

本アプリケーションノートを他のマイコンへ適用する場合、そのマイコンの仕様にあわせて変更し、十分評価してください。

目次

1. CA78K0R から CC-RL へのプロジェクトの移行方法

開発統合環境 CS+ 用 C コンパイラ CA78K0R で作成されたソースコードを、開発統合環境 CS+ 用 C コンパイラ CC-RL に対応したソースコードに置き換える方法は、二つあります。

一つは、開発統合環境 CS+で新規にプロジェクトを作成し開発統合環境 CS+ 用 C コンパイラ CA78K0R で作成されたソースコードを手動で移植して、開発統合環境 CS+ 用 C コンパイラ CC-RL に対応したプロジェクトへ移行する方法。もう一つは、開発統合環境 CS+の移行支援機能を使用し開発統合環境 CS+ 用 C コンパイラ CA78K0R で作成されたソースコードを流用する事によって、開発統合環境 CS+ 用 C コンパイラ CC-RL に対応したプロジェクトへ移行する方法です。

第 2 章では手動での移行方法について、第 3 章では移行支援機能を用いた移行方法についての説明を行います。

2.　手動での移行方法

2.1　　ソースコードの自動生成

開発統合環境 CS+ 用 C コンパイラ CC-RL のコード生成ツールでソースコードの自動生成を行います。置き換え元の開発統合環境 CS+ 用 C コンパイラ CA78K0R で作成されたソースコードを参照して、コード生成ツールを設定します。

(1)　プロジェクト・ツリーの中からコード生成（設計ツール）の「クロック発生回路」をクリックします。（図 2.1 の A）

(2)　「端子割り当て設定」を行い、[確定する]ボタンをクリックします。（図 2.1 の B）

注意.　　他の機能を設定するためには、端子割り当て設定を行う必要があります。なお、端子割り当て設定を一度確定させると、端子割り当て設定は変更できません。

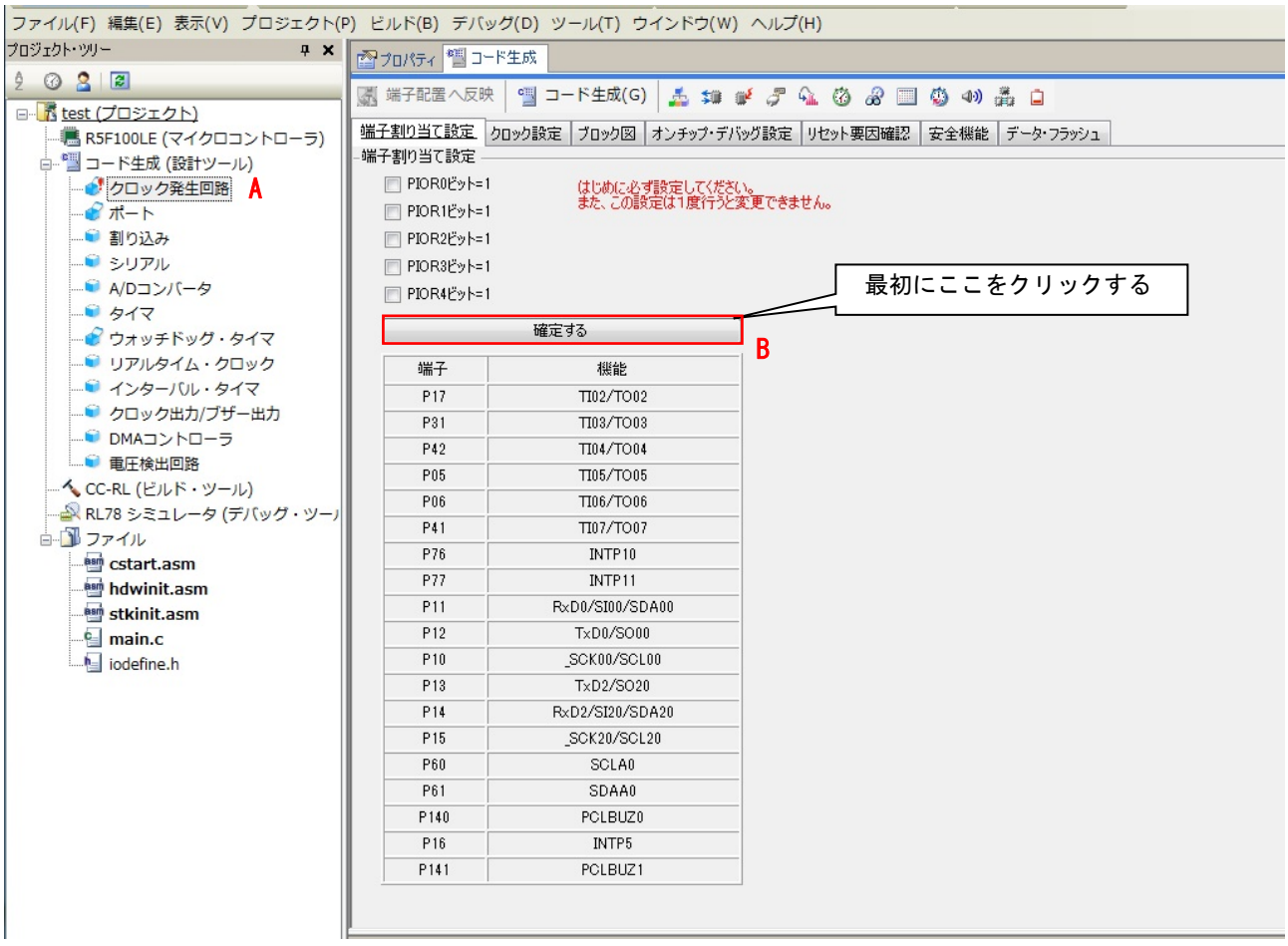

図 2.1　コード生成ツールの設定画面 (1)

(3)　置き換え元の開発統合環境 CS+ 用 C コンパイラ CA78K0R で作成されたソースコードを参照して、各機能の設定をします。

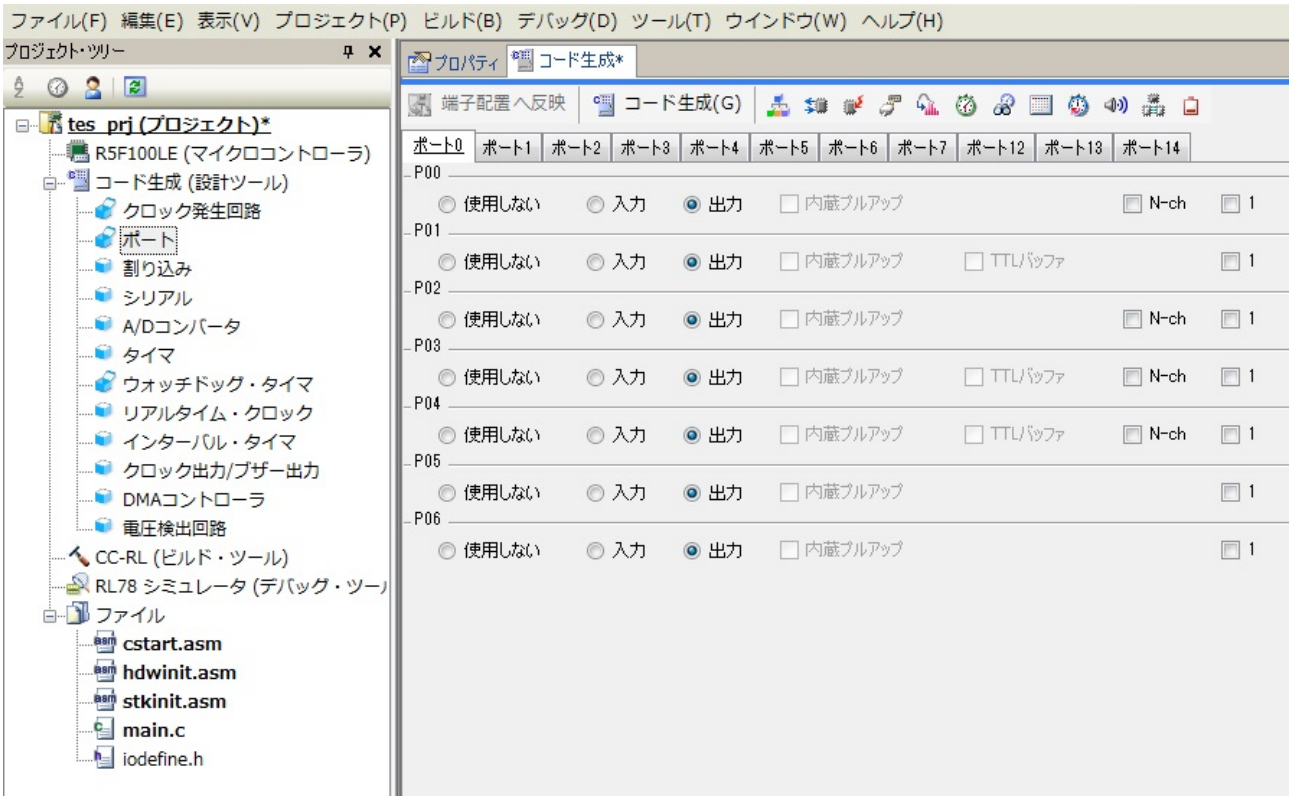

図 2.2　コード生成ツールの設定画面 (2)

(4)　全ての機能の設定を完了したら、画面上部にある[コード生成 (G)]ボタンをクリックして、コード生成（ソースコードの自動生成）を行います。（図 2.3 の C）

図 2.3　コード生成ツールの設定画面 (3)

2.2　自動生成されたソースコード以外のソースコード追加

2.2.1　ソースコード追加

コード生成ツールで自動生成されたソースコードに必要となるソースコードを追加します。

まず、CA78K0R で作成されたソースコードと自動生成されたソースコードの差分を確認します。差分確認は、複数のテキストファイルを比較できるソフト等を利用してください。

次に、差分を自動生成されたソースコードに追加します。”/* Start user code for include. Do not edit comment generated here */”と”/* End user code. Do not edit comment generated here */”の間にソースコードを追加します。

上記範囲外の箇所にソースコードを追加した場合は、自動生成ツールの「コード生成(G)」ボタンを押して再度ソースコードの自動生成を行うと、上記範囲外に追加したソースコードが消去されます。この消去を回避するためには、下記コード生成ツールの設定変更を実施してください。

図 2.1 の赤枠内の様に、「ファイル生成モード」の「モード」を「ファイルをマージする」から「すでにファイルがあれば何もしない」に変更します。

図 2.4 コード生成のプロパティ

2.2.2　ユーザ初期化関数の追加

CA78K0R のコード生成ツールで自動生成されていたユーザ初期化処理用の関数（R_×××_Create_UserInit（×××は機能名））は、CC-RL のコード生成ツールでは自動生成されません。また、機能の初期化関数内でのユーザ初期化処理用関数の呼び出し処理についても自動生成されません。

CA78K0R 用 r_cg_dmac.c

```
44
45      /***********************************************************************************************
46       * Function Name: R_DMAC0_Create
47       * Description  : This function initializes the DMA0 transfer.
48       * Arguments    : none
49       * Return Value : none
50       ***********************************************************************************************/
51      void R_DMAC0_Create(void)
52      {
53          DRC0 = _80_DMA_OPERATION_ENABLE;
54          NOP();
55          NOP();
56          DMAMK0 = 1U;    /* disable INTDMA0 interrupt */
57          DMAIF0 = 0U;    /* clear INTDMA0 interrupt flag */
58          /* Set INTDMA0 low priority */
59          DMAPR10 = 1U;
60          DMAPR00 = 1U;
61          DMC0 = _00_DMA_TRANSFER_DIR_SFR2RAM | _20_DMA_DATA_SIZE_16 | _01_DMA_TRIGGER_AD;
62          DSA0 = _1E_DMA0_SFR_ADDRESS;
63          DRA0 = _FE20_DMA0_RAM_ADDRESS;
64          DBC0 = _0000_DMA0_BYTE_COUNT;
65          DEN0 = 0U;  /* disable DMA0 operation */
66      ①   R_DMAC0_Create_UserInit();
67          }
68      /***********************************************************************************************
69       * End of function R_DMAC0_Create
70       ***********************************************************************************************/
71
```

CC-RL 用 r_cg_dmac.c

```
49
50      /***********************************************************************************************
51       * Function Name: R_DMAC0_Create
52       * Description  : This function initializes the DMA0 transfer.
53       * Arguments    : None
54       * Return Value : None
55       ***********************************************************************************************/
56      void R_DMAC0_Create(void)
57      {
58          DRC0 = _80_DMA_OPERATION_ENABLE;
59          NOP();
60          NOP();
61          DMAMK0 = 1U;    /* disable INTDMA0 interrupt */
62          DMAIF0 = 0U;    /* clear INTDMA0 interrupt flag */
63          /* Set INTDMA0 low priority */
64          DMAPR10 = 1U;
65          DMAPR00 = 1U;
66          DMC0 = _00_DMA_TRANSFER_DIR_SFR2RAM | _20_DMA_DATA_SIZE_16 | _01_DMA_TRIGGER_AD;
67          DSA0 = _1E_DMA0_SFR_ADDRESS;
68          DRA0 = _FE20_DMA0_RAM_ADDRESS;
69          DBC0 = _0000_DMA0_BYTE_COUNT;
70          DEN0 = 0U;  /* disable DMA0 operation */
71      ②
72
73      /***********************************************************************************************
74       * Function Name: R_DMAC0_Start
```

ユーザ初期化処理用関数
"R_DMAC0_Cereate_UserInit()" の
呼び出し処理が生成されない

図 2.5 CA78K0R と CC-RL のコード生成ツールの違い

DMA0 のユーザ初期化処理用関数（R_DMA0_Create_UserInit）を使用していた場合を例として、ユーザ初期化処理関数の追加方法を説明します。

(1) CA78K0R 用ソースコード r_cg_dmac_user.c にある” R_DMA0_Create_UsetInit ”を CC-RL 用ソースコード r_cg_dmac_user.c にコピーします。

CA78K0R 用 r_cg_dmac_user.c

```
52      /*·End·user·code.·Do·not·edit·comment·generated·here·*/
53      #include·"r_cg_userdefine.h"
54
55      /***********************************************************************************
56      *·Function·Name:·R_DMAC0_Create_UserInit
57      *·Description··:·This·function·adds·user·code·after·initializing·DMA0.
58      *·Arguments··:·none
59      *·Return·Value··:·none
60      ***********************************************************************************/
61      void·R_DMAC0_Create_UserInit(void)
62      {
63      ⟶/*·Start·user·code.·Do·not·edit·comment·generated·here·*/
64      ⟶DEN0⟶=·1U;⟶⟶⟶⟶⟶⟶⟶/*·Enable·DMA0·operation·*/
65      ⟶DRA0⟶=·(uint16_t)&g_AdResult;⟶/*·Set·destination·RAM·address·*/
66      ⟶DBC0⟶=·ADC_USED_CH_NUM·*·ADC_EXEC_TIMES;
67      ⟶DEN0⟶=·0U;⟶⟶⟶⟶⟶⟶⟶/*·Disable·DMA0·operation·*/
68      ⟶/*·End·user·code.·Do·not·edit·comment·generated·here·*/
69      }
70      /***********************************************************************************
71      End·of·function·R_DMAC0_Create_UserInit
72      ***********************************************************************************/
73
```

CC-RL 用 r_cg_dmac_user.c

“R_DMAC0_Create_UserInit”を全てコピー

```
66      /*·Start·user·code·for·adding.·Do·not·edit·comment·generated·here·*/
67      /***********************************************************************************
68      *·Function·Name:·R_DMAC0_Create_UserInit
69      *·Description··:·This·function·adds·user·code·after·initializing·DMA0.
70      *·Arguments··:·none
71      *·Return·Value··:·none
72      ***********************************************************************************/
73      void·R_DMAC0_Create_UserInit(void)
74      {
75      ····DEN0····=·1U;·····················/*·Enable·DMA0·operation·*/
76      ····DRA0····=·(uint16_t)&g_AdResult;····/*·Set·destination·RAM·address·*/
77      ····DBC0····=·ADC_USED_CH_NUM·*·ADC_EXEC_TIMES;
78      ····DEN0····=·0U;·····················/*·Disable·DMA0·operation·*/
79      }
80      /***********************************************************************************
81      End·of·function·R_DMAC0_Create_UserInit
82      ***********************************************************************************/
83      /*·End·user·code.·Do·not·edit·comment·generated·here·*/
84
85
```

図 2.6 ユーザ初期化処理用関数の追加

(2)　追加した関数についてグローバル宣言をします。

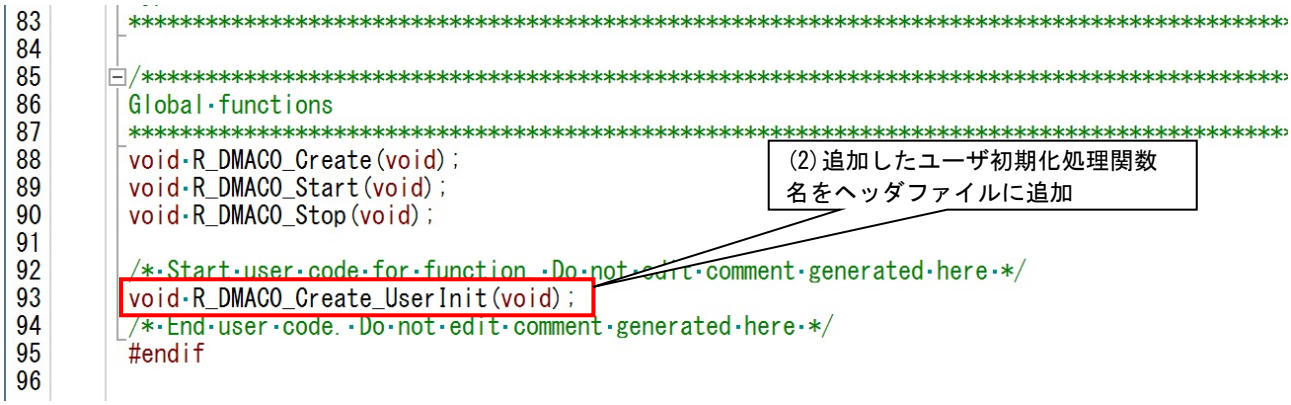

図 2.7 CC-RL 用ヘッダファイル”r_cg_dmac.”h の記述例

(3)　追加したユーザ初期化処理関数の呼び出し処理を r_main.c ファイル内の関数 R_MAIN_UserInit()に追加します。

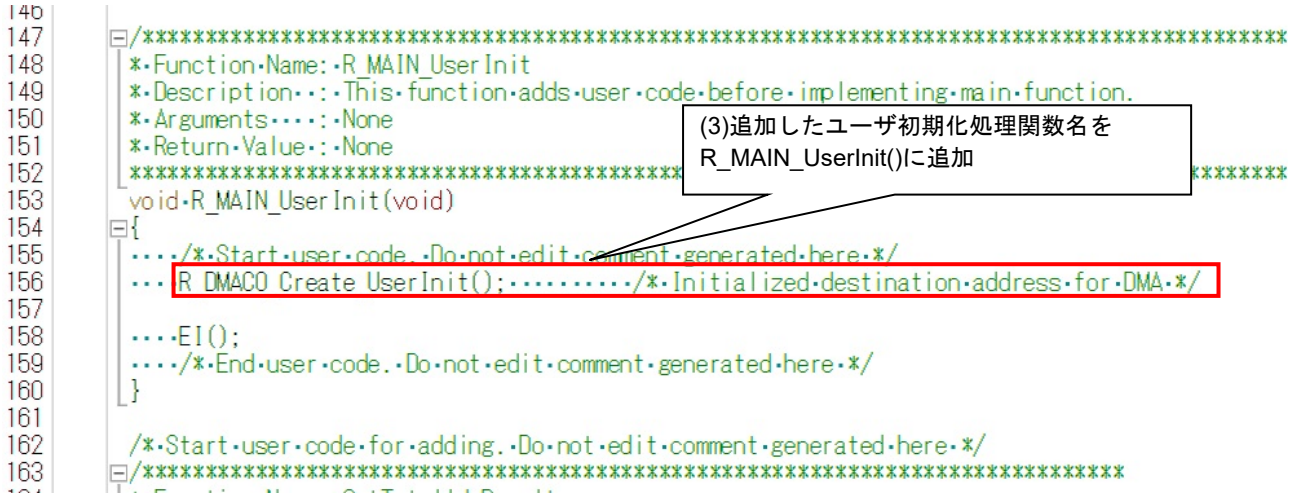

図 2.8　CC-RL 用ユーザ初期化処理用関数の呼び出し処理の追加

以上で、ユーザ初期化処理関数の追加作業は完了です。

2.3　追加部分の修正

第 2 章の作業で追加したソースコードの状態だと、ワーニングまたはエラーが出る可能性があります。その場合は、CC-RL 仕様に合わせて記述を修正する必要があります。

CA78K0R と CC-RL の主な記述仕様差について説明します。

2.3.1　特殊機能レジスタ（SFR）へのアクセス

(1)　特殊機能レジスタ（SFR）へのアクセス方法を変更します。

CA78K0R：#pragma sfr

CC-RL：#include ”iodefine.h”

#pragma sfr は CC-RL ではサポートされていませんので、コード生成ツールにより自動生成される sfr アクセス用デファインヘッダファイルの”iodefine.h”をインクルードしてください。

(2)　ポート・レジスタの記述を修正します。

CA78K0R を利用する場合はレジスタ名称の後に「.ビット番号」を付加しますが、CC-RL を利用する場合はレジスタ名称の後に「_bit.no ビット番号」を付加します。

CA78K0R 用 r_main.c

```
88
89          /* AD conversion stop */
90          R_ADC_Stop();
91
92          /* Check result of AD conversion data */
93          if (result == 0x00)
94          {
95              if (testVoltageIndex == 2)   /* AD test all OK */
96              {
97                  /* LED1 turn on */
98                  P6.2 = 0;
99                  while (1U)
100                 {
101                     /* Do Nothing */
102                 }
103             }
104             else                    /* Next AD test */
105             {
106                 ++testVoltageIndex;
107             }
108         }
109         else    /* AD test NG */
110         {
111             /* LED blinks */
112             R_Main_Blink_Led();
113         }
114     }
115     /* End user code. Do not edit comment generated here */
116 }
```

CC-RL 用 r_main.c

```
83
84          /* Gets the check result of AD conversion data */
85          result = R_Main_Check_AD_Data(testVoltageIndex);
86
87          /* AD conversion stop */
88          R_ADC_Stop();
89
90          /* Check result of AD conversion data */
91          if (result == 0x00U)
92          {
93              if (testVoltageIndex == 2U) /* AD test all OK */
94              {
95                  /* LED1 turn on */
96                  P6_bit.no2 = 0U;
97                  while (1U)
98                  {
99                      /* Do Nothing */
100                 }
101             }
102             else                    /* Next AD test */
```

図 2.9 ポート・レジスタの記述

2.3.2　割り込み機能の有効化

#pragma 指令を関数に置き換えます。

1. di の場合

   #pragma di → __DI();

   (r_cg_macrodriver.h を使用している場合は、DI();でも可)

2. ei の場合

   #pragma ei → __EI();

   (r_cg_macrodriver.h を使用している場合は、EI();でも可)

2.3.3　CPU 制御命令の有効化

#pragma 指令を関数に置き換えます。

1. halt の場合

   #pragma halt → __halt();

   (r_cg_macrodriver.h を使用している場合は、HALT();でも可)

2. stop の場合

   #pragma stop → __stop();

   (r_cg_macrodriver.h を使用している場合は、STOP();でも可)

3. brk の場合

   #pragma brk → __brk();

   (r_cg_macrodriver.h を使用している場合は、BRK();でも可)

4. nop の場合

   #pragma nop → __nop();

   (r_cg_macrodriver.h を使用している場合は、NOP();でも可)

2.3.4　絶対番地指定（__directmap）の置き換え

絶対番地を指定するために、CA78K0R を利用する場合は「 __directmap 」を使用しますが、CC-RL を利用する場合は「 #pragma address」を使用します。

①「__directmap 型指定 変数名 = 開始アドレス;」を②「#pragma address 変数名 = 開始アドレス」と③「形指定 変数名;」に変更します。

例）

① __directmap uint8_t p130_high = {0xFE900};

② #pragma address p130_high = 0xFE900U

③ uint8_t __near p130_high;

CA78K0R 用 r_main.c

```
/************************************************************************************
Global variables and functions
************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
① __directmap uint8_t p130_high            = {0xFE900};
  __directmap uint8_t p130_low             = {0xFE901};
  __directmap uint8_t adc_snooze           = {0xFE902};
  __directmap uint16_t get_adcr[MAX_BUFFER] = {0xFEA00};

  uint8_t  buffer_count;                /* buffer counter */
  uint16_t result_buffer[MAX_BUFFER];   /* AD converter result buffer */
  /* End user code. Do not edit comment generated here */
```

CC-RL 用 r_main.c

```
/************************************************************************************
Pragma directive
************************************************************************************/
/* Start user code for pragma. Do not edit comment generated here */
② #pragma address p130_high = 0xFE900U
  #pragma address p130_low = 0xFE901U
  #pragma address adc_snooze = 0xFE902U
  #pragma address get_adcr = 0xFEA00U
  /* End user code. Do not edit comment generated here */

/************************************************************************************
Global variables and functions
************************************************************************************/
/* Start user code for global. Do not edit comment generated here */
③ uint8_t __near p130_high;
  uint8_t __near p130_low;
  uint8_t __near adc_snooze;
  uint16_t __near get_adcr[MAX_BUFFER];

  uint8_t  buffer_count;                /* buffer counter */
  uint16_t result_buffer[MAX_BUFFER];   /* AD converter result buffer */
  /* End user code. Do not edit comment generated here */
```

図 2.10 __directmap の記述

2.3.5　変数の saddr 領域配置（sreg, __sreg）の置き換え

変数の saddr 領域への配置を行うために、CA78K0R を利用する場合は「sreg(もしくは__sreg)」を使用しますが、CC-RL を利用する場合は「__saddr」を使用します。

①「sreg 形指定 変数名；（もしくは、__sreg 形指定 変数名；）」を②「__saddr 形指定 変数名；」に変更します。

例）

① sreg uint32_t g_PulseWidth[8];

② __saddr uinit32_t g_pulse_width[8];

CA78K0R 用 r_main.c

```
52      /************************************************************************
53      Exported global variables and functions (to be accessed by other files)
54      ************************************************************************/
55  ①  sreg uint32_t g_PulseWidth[8];        /* Store pulse width */
56      uint8_t g_Times;                     /* Measurement times counter */
57      /* End user code. Do not edit comment generated here */
58
59      /************************************************************************
```

CC-RL 用 r_main.c

```
46      /************************************************************************
47      Global variables and functions
48      ************************************************************************/
49      /* Start user code for global. Do not edit comment generated here */
50  ②  __saddr uint32_t g_pulse_width[8];    /* Store pulse width */
51      uint8_t g_times;                     /* Measurement times counter */
52      /* End user code. Do not edit comment generated here */
53      void R_MAIN_UserInit(void);
54
55      /************************************************************************
```

図 2.11 sreg 記述の変更例

2.3.6　near/far 属性

メモリ・モデルについて、CA78K0R はスモール・モデル、ミディアム・モデル、ラージ・モデルがありますが、CC-RL はスモール・モデル、ミディアム・モデルのみになります。

CA78K0R のスモール・モデル、ミディアム・モデルを使用していた場合は、同じメモリ・モデルである CC-RL のスモール・モデル、ミディアム・モデルを使用します。

CA78K0R のラージ・モデルを使用していた場合は、CC-RL のミディアム・モデルを使用します。さらに、near 領域または far 領域の指定がない関数および変数は near 領域に配置されますので、__far 型修飾子を利用して far 領域に配置します。

CRC 演算結果データの参照など 64KB を超える領域を参照するソースコードでは、ポインタを far 属性に変更する必要があります。

例）

① ”uint16_t *oc_calc_hs_crc;” を
”__far uint16_t *oc_calc_hs_crc;”に変更。

② ”oc_calc_hs_crc = (uint16_t *)HIGHSPEED_CALC_ADDR;”を
”oc_calc_hs_crc = (__far uint16_t *)HIGHSPEED_CALC_ADDR;”に変更。

CA78K0R 用 r_main.c

```
79      ①uint16_t  *oc_calc_hs_crc;  /*Pointer of OC result.(High-Speed)*/
80      uint16_t  count;
81
82      /*High-speed CRC*/
83      /*Get High-speed CRC calculated result that OC output.*/
84      ②oc_calc_hs_crc = (uint16_t*)HIGHSPEED_CALC_ADDR;
85
86      result_hs_crc = R_HighSpeedCRCProc();/*Process of high-speed CRC.*/
87
88      /*The results are compared and it outputs it to LED.*/
89      if (result_hs_crc == *oc_calc_hs_crc)/*High-speed CRC.*/
90      {
91          P6.2 = 0;                    /*OK = LED Lighting.*/
92      }
```

CC-RL 用 r_main.c

```
80      ........uint16_t........result_gp_crc = 0U;/*Program result.(General-Purpose)*/
81      ....①..__far uint16_t  *oc_calc_hs_crc;....../*Pointer of OC result.(High-Speed)*/
82      ........uint16_t........count = 0U;
83
84      ......../*High-speed CRC*/
85      ......../*Get High-speed CRC calculated result that OC output.*/
86      ....②..oc_calc_hs_crc = (__far uint16_t *)HIGHSPEED_CALC_ADDR;
87      ........result_hs_crc = R_HighSpeedCRCProc();.../*Process of high-speed CRC.*/
88
89      ......../*The results are compared and it outputs it to LED.*/
90      ........if (result_hs_crc == *oc_calc_hs_crc)....../*High-speed CRC.*/
91      ........{
92      ............P6_bit.no2 = 0U;....................../*OK = LED Lighting.*/
93      ........}
```

図 2.12 ポインタの属性変更例

3.　移行支援機能を用いた移行方法

開発統合環境 CS+ 用 C コンパイラ CA78K0R の既存プロジェクトを流用して開発統合環境 CS+ 用 C コンパイラ CC-RL のソースコードに移行する方法について説明します。

3.1　既存プロジェクトを流用したプロジェクトの作成

① 画面上部にある[スタート(S)]ボタンをクリックして、スタートメニューを表示します。

② スタートメニューの「新しいプロジェクトを作成する」の項目の[GO]ボタンをクリックします。

③ 使用するマイクロコントローラを選択します。

④ プロジェクトの種類(K)を「アプリケーション(CC-RL)」にします。

⑤ 「既存のプロジェクトのファイル構成を流用する(S)」のチェックボックスにチェックを入れ「流用元のプロジェクト(P)」に流用するプロジェクトファイル名を入力します。最後に「プロジェクト・フォルダ以下の構成ファイルをコピーして流用する(O)」のチェックボックスにチェックを入れます。

⑥ [作成(C)]ボタンをクリックしてプロジェクトを作成します。

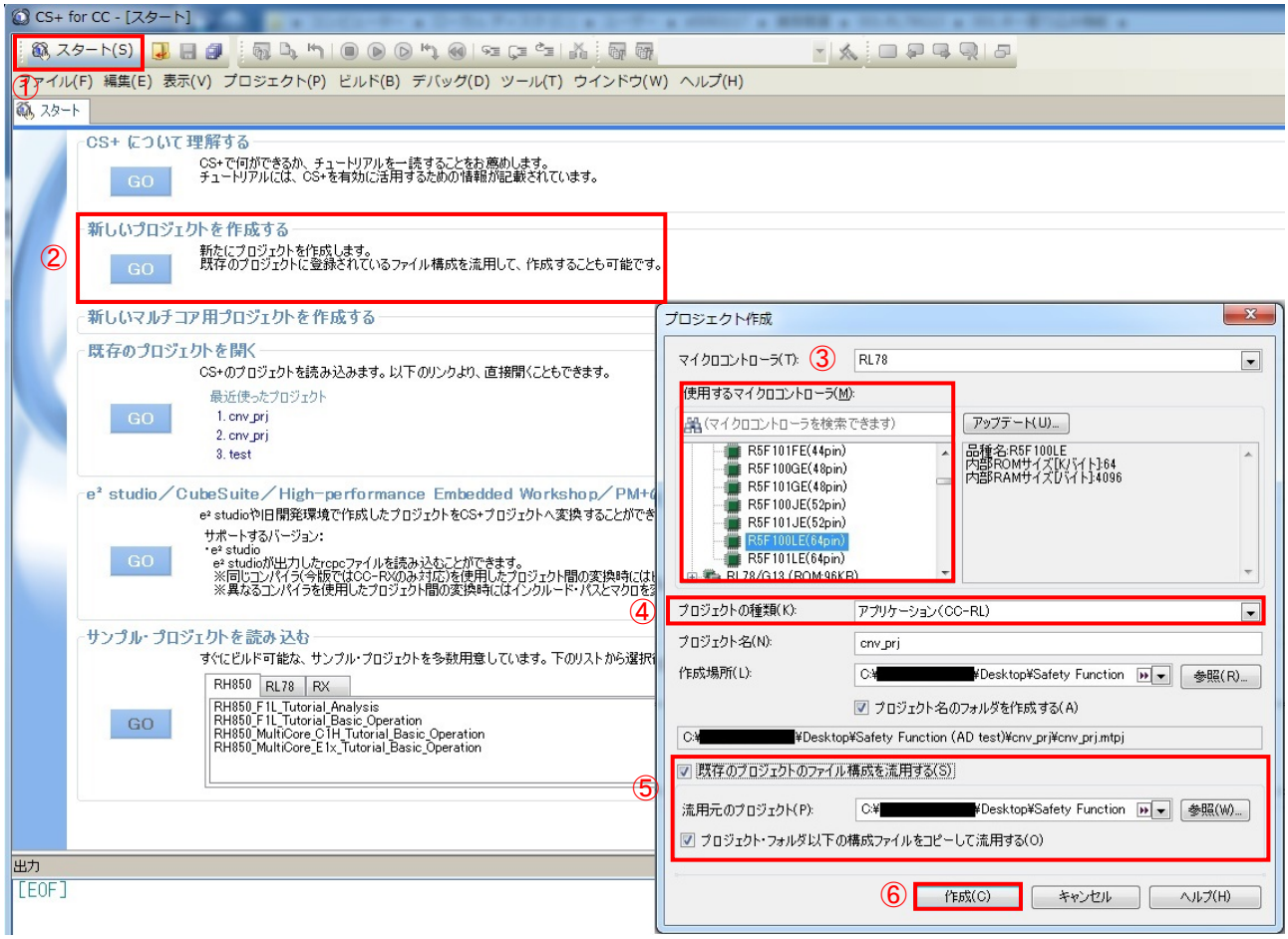

図 3.1 CS+のスタートメニュー画面

3.2　インクルードファイルの追加

プロジェクト・ツリーの「CC-RL(ビルド・ツール)」をクリックしコンパイル・オプションタブの「コンパイル単位の先頭にインクルードするファイル」項目を開き、”iodefine.h”を追加します。

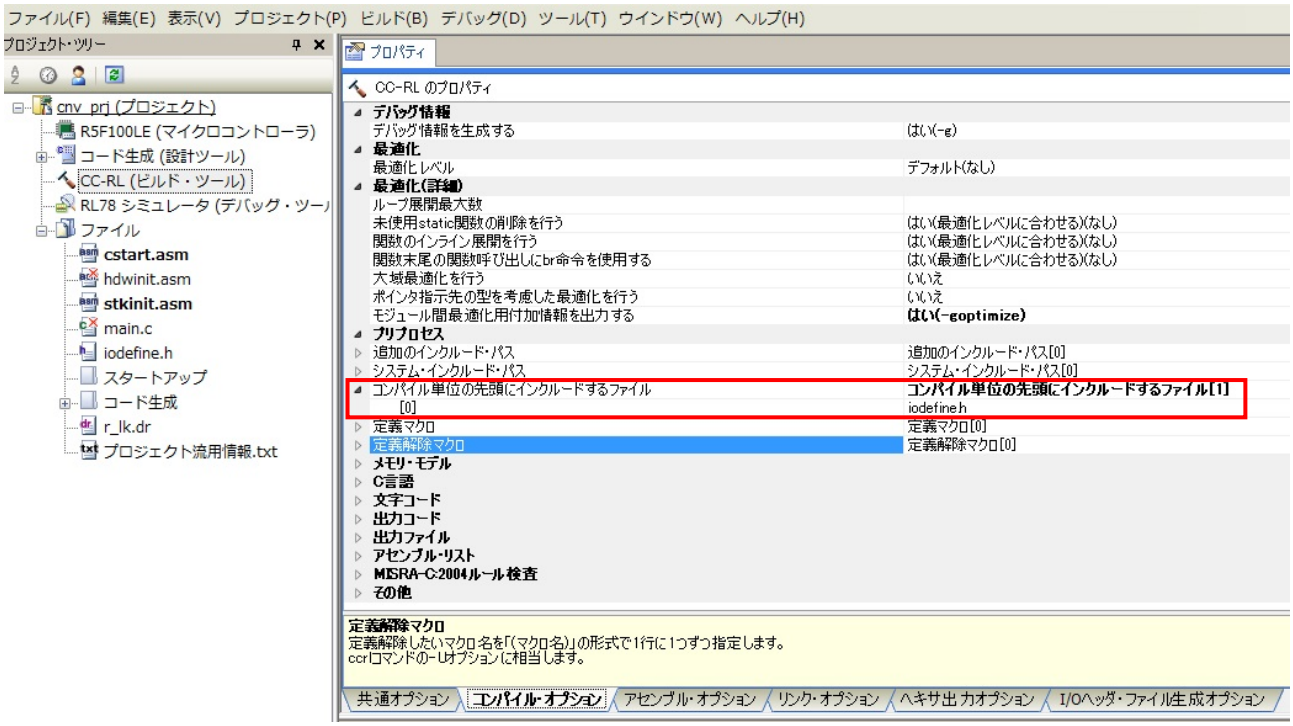

図 3.2 コンパイル・オプション

3.3　スタートアップファイルの変更

移行元のプロジェクトで、main 関数、hdwinit 関数が登録されている場合には、以下の手順で。プロジェクト作成時に自動生成されたファイル（main.c、hdwinit.asm）をビルド対象外にしてください。

① プロジェクト・ツリーの main.c を右クリックしてメニューを表示します。

② メニューから[プロパティ(P)]を選択します。

③ ファイルのプロパティにある[ビルドの対象とする]の項目を「はい」から「いいえ」に変更します。
（hdwinit.asm も同様の手順で変更します。）

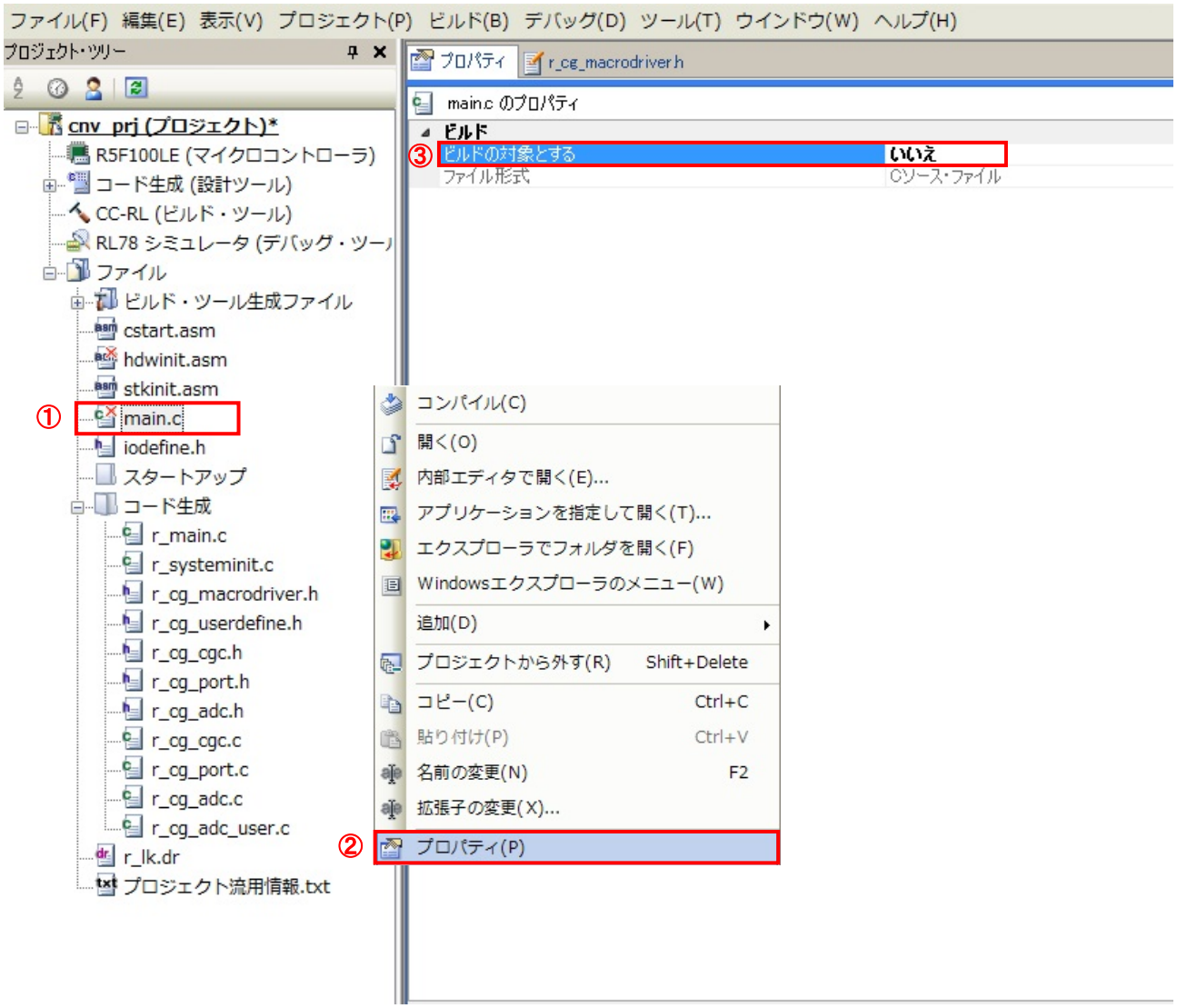

図 3.3 ファイルのプロパティ表示

3.4　特殊機能レジスタ（SFR）アクセス記述の削除

”r_cg_macrodriver.h”にある”#pragma sfr”の記述を削除します。

```
36      /***********************************************************************************
37      Includes
38      ***********************************************************************************
39      #pragma sfr
40      #pragma DI
41      #pragma EI
42      #pragma NOP
43      #pragma HALT
44      #pragma STOP
45
46      /***********************************************************************************
```

図 3.4 r_cg_macrodriver.h

絶対番地指定（__directmap）の置き換えは移行支援機能が対応していませんので、2.3.4 を参考にして手動で置き換えを行ってください。

4.　ROM 配置方法

セクション配置をするために、CA78K0R を利用する場合はリンク・ディレクディブ・ファイルを使用しますが、CC-RL を利用する場合はリンク・オプションのセクションで設定します。なお、セクション配置は-start オプションでも設定できます。

プロジェクト・ツリーの「CC-RL（ビルド・ツール）」をクリックします。

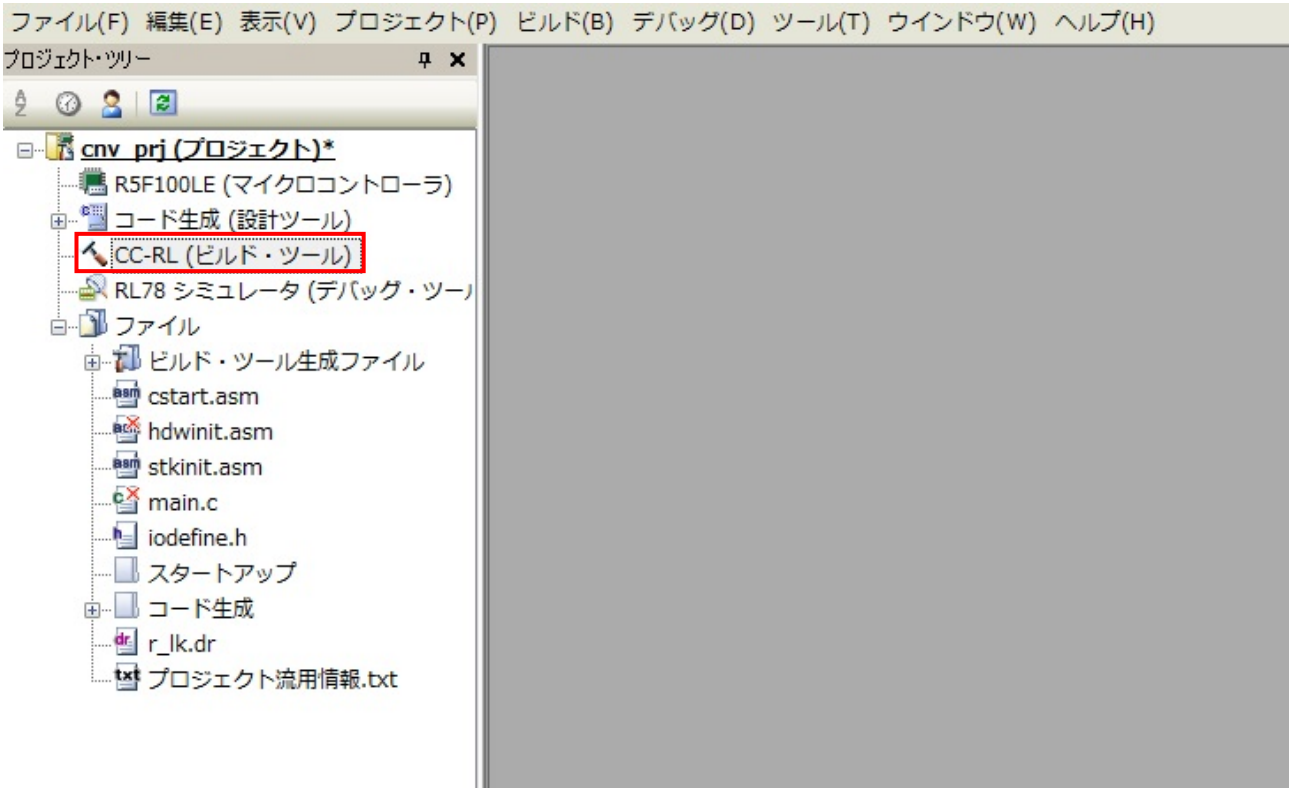

図 4.1 CC-RL（CS+）でのリンク・オプション設定例(1)

① [リンク・オプション]のタブをクリックします。

② セクションの項目を開いてください。

③ セクションを自動的に配置する：「はい」から「いいえ」に変更します。

④ セクションの開始アドレスの項目の[…]ボタンをクリックします。

⑤ セクション設定編集ウインドウが立ち上がるので、セクションの設定を行ってください。

⑥ [OK]ボタンをクリックでセクション設定の編集を完了してください。

必須事項

・各セクションの ROM 領域/RAM 領域の配置を変更することはできません。
・SFR 領域、割り込みベクタ領域（セクション.vect）、CALLT 関数テーブル領域（セクション.callt0）は配置場所が決まっているため指定しません。
・saddr 領域配置のセクション（セクション.sdataR、.sbss）は.saddr 領域の範囲内に配置します。

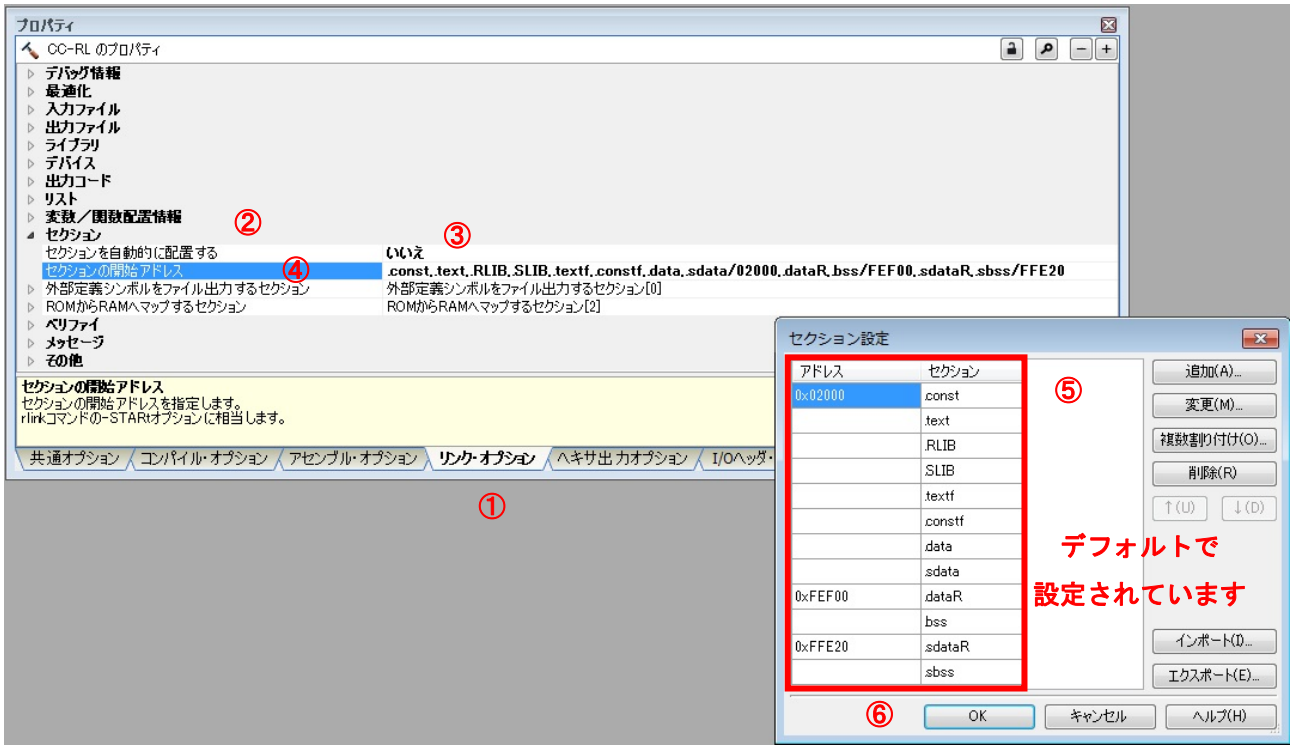

図 4.2 CC-RL（CS+）でのリンク・オプション設定例(2)

5. サンプルコード

サンプルコードは、ルネサス エレクトロニクスホームページから入手してください。

6. 参考ドキュメント

RL78 ファミリ ユーザーズマニュアル ソフトウェア編（R01US0015J）

RL78 コンパイラ CC-RL ユーザーズマニュアル(R20UT3123J)

RL78 ファミリ用 統合開発環境 CA78K0R から CC-RL への移行（プロジェクト操作編）(R20UT3415J)

RL78 ファミリ用 統合開発環境 CA78K0R から CC-RL への移行（コーディング編）(R20UT3416J)

RL78 ファミリ用 統合開発環境 CA78K0R から CC-RL への移行（リンカオプション編）(R20UT3417J)

RL78 ファミリ用 統合開発環境 CA78K0R から CC-RL への移行（コンパイラオプション・アセンブラオプション編）(R20UT3418J)

CS+ コード生成ツール 統合開発環境ユーザーズマニュアル RL78 API リファレンス編（R20UT3102J）

CS+ V3.01.00 統合開発環境ユーザーズマニュアル メッセージ編（R20UT3286J）

CS+ V3.01.00 統合開発環境ユーザーズマニュアル プロジェクト操作編（R20UT3287J）

（最新版をルネサス エレクトロニクスホームページから入手してください。）

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/inquiry

改訂記録	CS+用サンプルコードの置き換え方法

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2015.09.30	—	初版発行
1.10	2017.09.27	4,6	図番号修正

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生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、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またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 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、全部または一部を問わず、改造、改変、複製、その他の不適切に使用しないでください。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）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   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（宇宙、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に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損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合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を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他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。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. 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、(1)核兵器、化学兵器、生物兵器等の大量破壊兵器およびこれらを運搬することができるミサイル（無人航空機を含みます。）の開発、設計、製造、使用もしくは貯蔵等の目的、(2)通常兵器の開発、設計、製造または使用の目的、または(3)その他の国際的な平和および安全の維持の妨げとなる目的で、自ら使用せず、かつ、第三者に使用、販売、譲渡、輸出、賃貸もしくは使用許諾しないでください。
   当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様の転売、貸与等により、本書（本ご注意書きを含みます。）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、当社は一切その責任を負わず、お客様にかかる使用に基づく当社への請求につき当社を免責いただきます。
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。
12. 本資料に記載された情報または当社製品に関し、ご不明点がある場合には、当社営業にお問い合わせください。

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。

(Rev.3.0-1　2016.11)

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　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■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：https://www.renesas.com/contact/