RL78開発環境移行ガイド

RL78ファミリ間の移行
(統合開発環境編)
(CA78K0R→CC-RL)

2016/12/28
R20UT3415JJ0102

ソフトウェア事業部ソフトウエア技術部
ルネサス システムデザイン株式会社

はじめに

- 本資料は、 RL78ファミリ用Cコンパイラ CA78K0R用のプロジェクトを RL78ファミリ用Cコンパイラ CC-RLへ移行する際の、CS+のプロジェクトの操作方法について記述しています。

- 本資料では、統合開発環境CS+、RL78ファミリ用Cコンパイラ CA78K0RおよびRL78ファミリ用Cコンパイラ CC-RLを対象に説明しています。
  対象バージョンは以下の通りです。
  - CS+ V4.01.00
  - CA78K0R V1.20以降
  - CC-RL V1.03.00

アジェンダ

既存環境からCC-RL用CS+への移行

プロジェクト移行方法

作成済みのCS+/CubeSuite+のCA78K0R用プロジェクトをCC-RL用のCS+環境へ移行するには以下の2つの方法があります。

方法１：CS+で新規にプロジェクトを作成
作成済みのお客様のソースファイルをRL78用CS+のプロジェクトを作成し登録

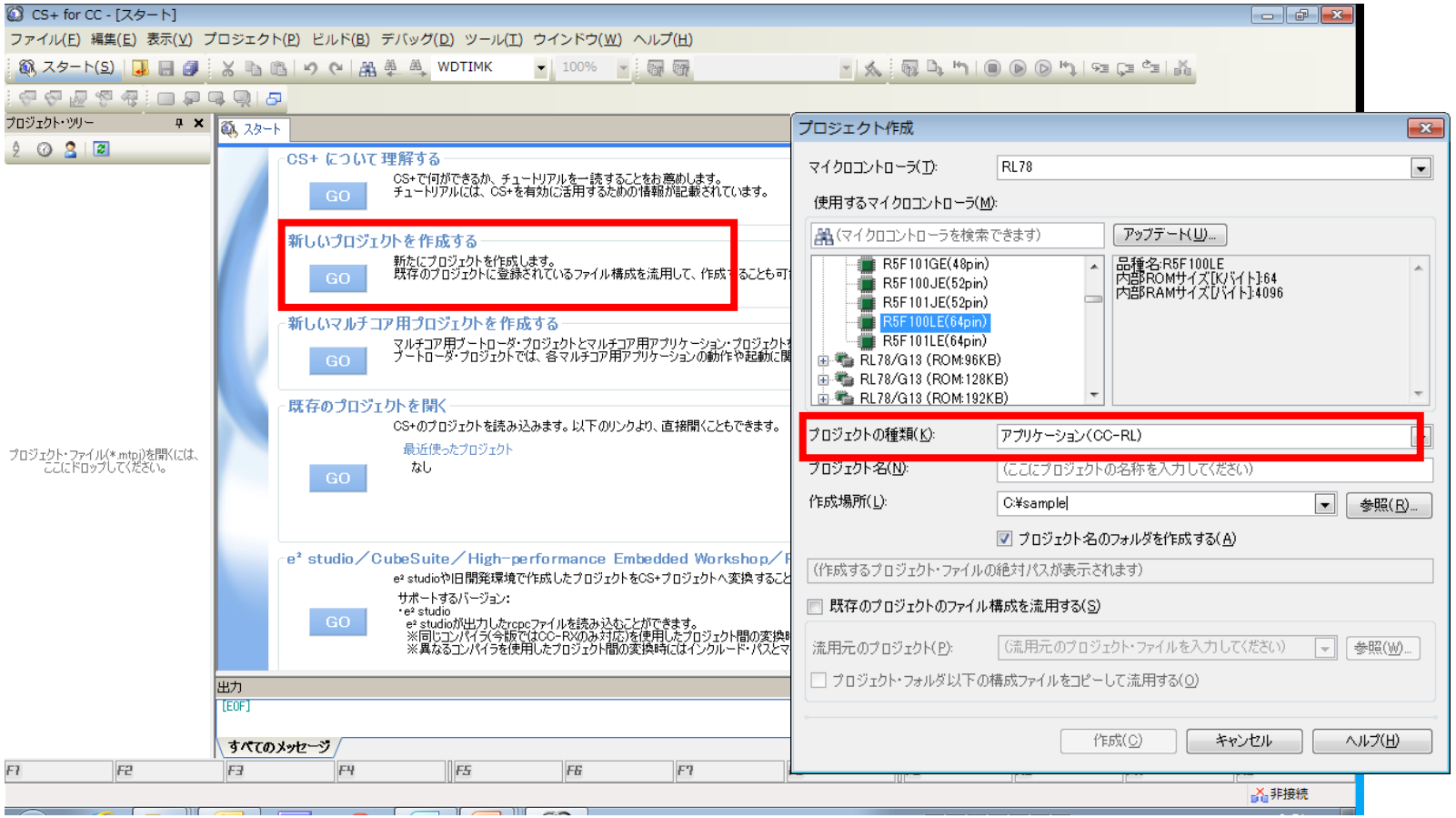
方法２：既存プロジェクトの流用
CA78K0R用CS+/CubeSuite+のプロジェクトを流用してCC-RL用CS+の新規プロジェクトを作成

内容	方法1	方法2
ソースファイル登録	手動	自動
オプション設定	手動	自動（一部のみ）
ソースファイルのフォルダ位置	考慮せず登録	流用元と同じ構成にする必要あり*
ソースファイルと自動生成ファイルとの競合	手動登録時に考慮	プロジェクト生成後に変更の必要あり

※：ソースファイルの変換を行わない場合、フォルダ構成が異なる場合パス解決に失敗しビルド・エラーとなる可能性があります。

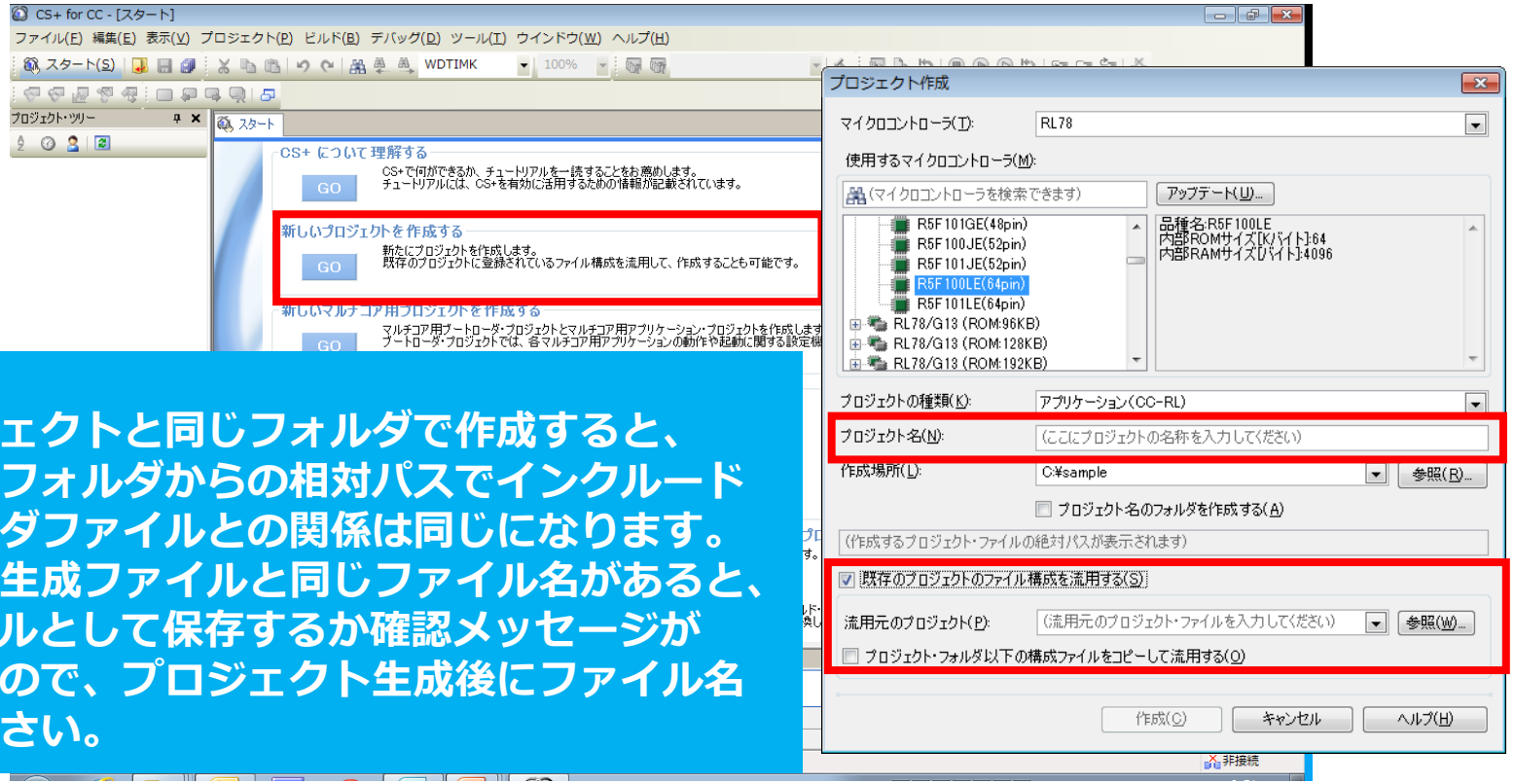
プロジェクトの新規作成によるプロジェクト移行

新規にCC-RL用のプロジェクト作成後、作成済みのCA78K0R用のソースファイルを登録して使用する



既存プロジェクトの流用によるプロジェクト移行 (1/2)

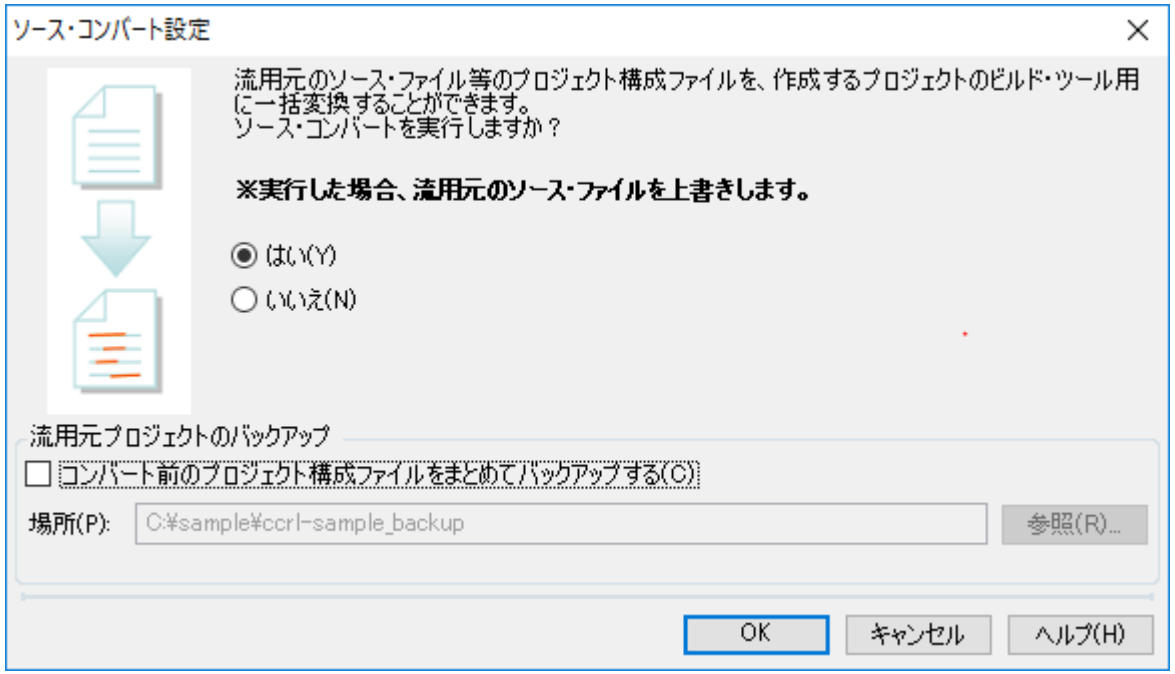
新規にCC-RL用のプロジェクトを作成時に[既存のプロジェクトファイル構成を流用する]を選択。
CA78K0Rで作成したプロジェクトを選択。



補足：
流用元プロジェクトと同じフォルダで作成すると、プロジェクトフォルダからの相対パスでインクルードしているヘッダファイルとの関係は同じになります。ただし、自動生成ファイルと同じファイル名があると、*.bakファイルとして保存するか確認メッセージが出力されますので、プロジェクト生成後にファイル名を戻してください。

既存プロジェクトの流用によるプロジェクト移行 (2/2)

プロジェクトを作成する際、CA78K0Rコンパイラ用のソース・ファイルをCC-RL用にコンバートすることもできます。

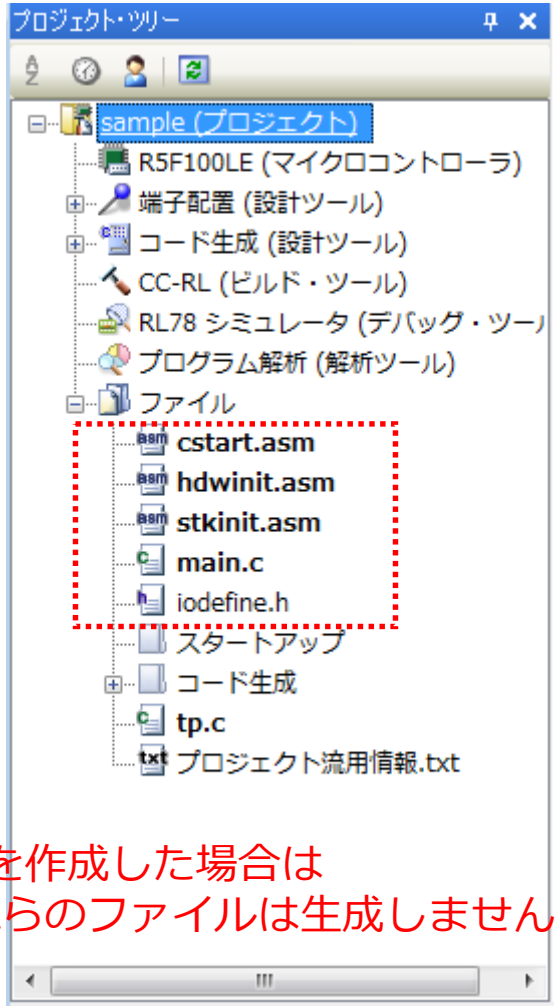


CA78K0R用プロジェクトとの違い

生成ファイル

CC-RL用のプロジェクトを新規に作成した場合、開発に必要な次のファイルを生成します。

- スタートアップファイル（cstart.asm）
- 初期化hdwinit関数ファイル（hdwinit.asm）
- スタック初期化stkinit関数ファイル（stkinit.asm）
  RL78-S1コアのマイコンの場合は出力されません）
- メイン関数ファイル（main.c）
- SFRファイル（iodefine.h）

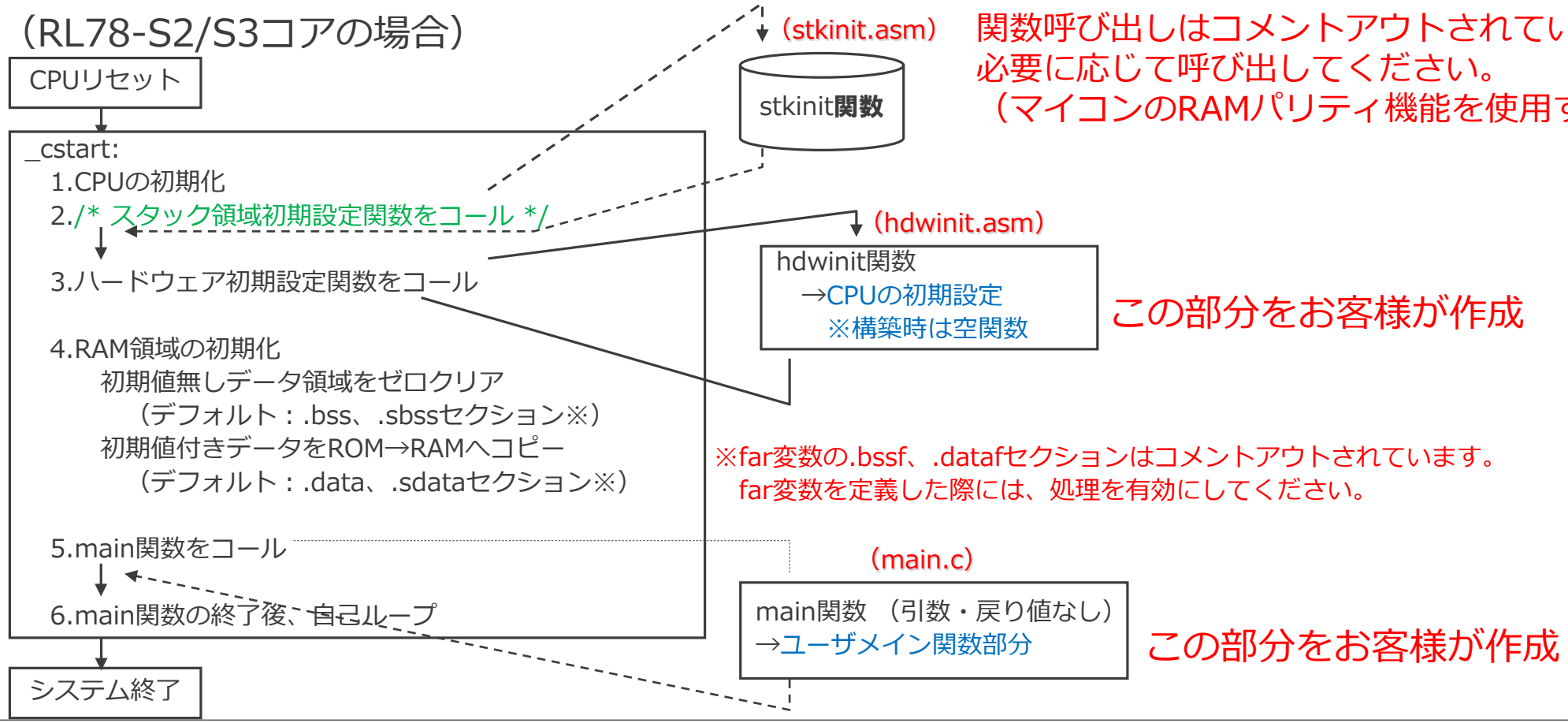


注
CA78K0R用プロジェクトを作成した場合は
CC-RLで生成しているこれらのファイルは生成しません。

スタートアップファイル (1/2)

プロジェクトツリーに登録されるスタートアップファイルの構成は、以下になります。

（RL78-S2/S3コアの場合）



関数呼び出しはコメントアウトされているので、
必要に応じて呼び出してください。
（マイコンのRAMパリティ機能を使用する場合など。）

この部分をお客様が作成

※far変数の.bssf、.datafセクションはコメントアウトされています。
far変数を定義した際には、処理を有効にしてください。

この部分をお客様が作成

スタートアップファイル (2/2)

移行元のプロジェクトで、main関数、hdwinit関数の登録がされている場合には、次のいずれかの方法で、プロジェクト作成時に自動で作成されたファイルをビルドの対象外にしてください。

- プロジェクトツリーからファイルの登録を削除
- プロジェクトツリーに登録された、main.c、hdwinit.asmファイルのプロパティで、ビルドの対象とするで「いいえ」を選択

iodefine.h (1/2)

RL78のSFRへアクセスするCソースの記述は、このファイル内の宣言を使用することで可能です。

```
<iodefine.h>
・・・
typedef struct
{
   unsigned char no0:1;
   unsigned char no1:1;
   unsigned char no2:1;
   unsigned char no3:1;
   unsigned char no4:1;
   unsigned char no5:1;
   unsigned char no6:1;
   unsigned char no7:1;
} __bitf_T;
・・・
#define ADM2      (*(volatile __near unsigned char  *)0x10)
#define ADM2_bit (*(volatile __near __bitf_T *)0x10)
#define P0          (*(volatile __near unsigned char  *)0xFF00)
#define P0_bit      (*(volatile __near __bitf_T *)0xFF00)
#define P1          (*(volatile __near unsigned char  *)0xFF01)
#define P1_bit      (*(volatile __near __bitf_T *)0xFF01)
 ・・・
#define INTP0                    0x0008
#define INTP1                    0x000A
#define INTP2                    0x000C
#define INTP3                    0x000E
 ・・・
```

```
<レジスタへアクセスするファイル>
#include “iodefine.h”
・・・
void main(void)
{
            ・・・
           ADM2 = 0x12;
           ADTYP = 1;
           P0_bit.no2 = 1;
            ・・・
}
・・・
#pragma interrupt inter
(vect=INTP0)
void __near inter ( void ) {
      /*割り込み処理*/
}
・・・
```

<記述方法>

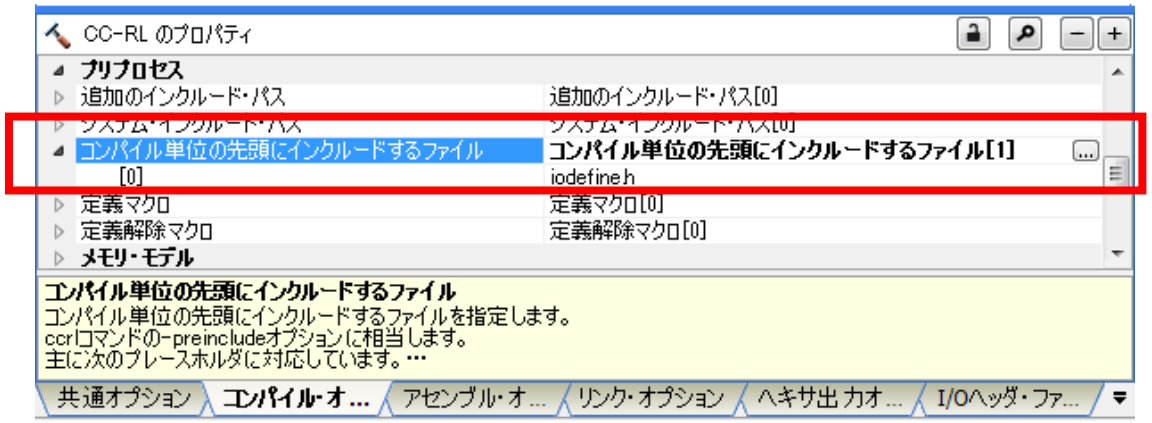
- iodefine.hファイル内の記述を使用して SFRレジスタ、ビットへアクセス可能
- 予約語になっていないビットについては、「_bit」のついた名前を使用してアクセス可能
- #pragma interruptにて、割り込み要求名を使用可能

iodefine.h (2/2)

iodefine.hをソースファイルにインクルードするには、次のいずれかの方法で可能です。

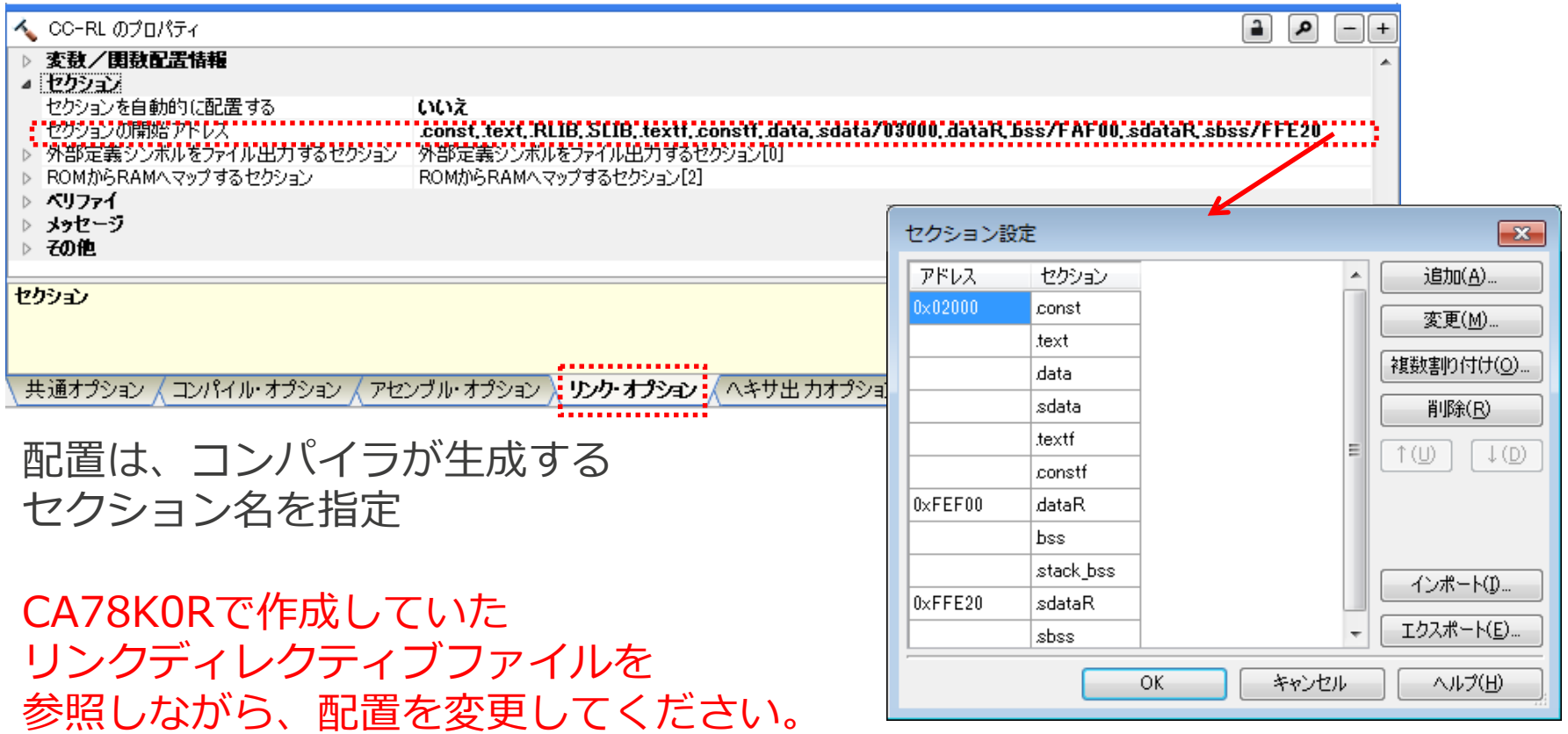
- 各ソースファイルに #include “iodefine.h” を記述する
  - -各ソースファイルごとに記述が必要です。
  - -#pragma interrupt のベクタテーブル指定の割り込み要求名、SFRアクセスの記述の前にインクルードする必要があります。
- コンパイルオプションで、-preinclude=iodefine.h を指定する
  - -全ソースファイルに適用されます。
  - -SFR名、割り込み要求名を、別の用途で使用していると、iodefine.hに定義に従って、#defineが展開されます。

```
<ソースファイルの例>
#include “iodefine.h”
・・・
void main(void)
{
        ADM2 = 0x12;
        P0_bit.no2 = 1;
}
#pragma interrupt inter
(vect=INTP0)
void __near inter ( void ) {
    /*割り込み処理*/
}
```



セクション配置 (1/3)

プログラムやデータのセクションの配置の指定は、プロパティのリンクオプションで設定してください。



配置は、コンパイラが生成する
セクション名を指定

CA78K0Rで作成していた
リンクディレクティブファイルを
参照しながら、配置を変更してください。

任意のアドレスに任意のセクションを指定可能

セクション配置 (2/3)

CC-RLでは、デフォルト・セクション名でセクションを生成します。

デフォルト・セクション名	再配置属性	内容
.callt0	CALLT0	callt関数呼び出しのテーブル用セクション
.text	TEXT	コード部用セクション（near領域配置）
.textf	TEXTF	コード部用セクション（far領域配置）
.textf_unit64kp	TEXTF_UNIT64KP	コード部用セクション（セクションを先頭が偶数番地になるように，64KB-1境界にまたがらないように配置）
.const	CONST	ROMデータ（near領域配置）（ミラー領域内）
.constf	CONSTF	ROMデータ（far領域配置）
.data	DATA	初期化データ用セクション（初期値あり、near領域配置）
.dataf	DATAF	初期化データ用セクション（初期値あり、far領域配置）
.sdata	SDATA	初期化データ用セクション（初期値あり、saddr配置変数）
.sbss_bit	SBSS_BIT	ビットデータ領域用セクション（初期値なし，saddr配置変数）

セクション配置 (3/3)

注　#pragma sectionでセクション名を変更することができません。

デフォルト・セクション名	再配置属性	内容
.bss_bit	BSS_BIT	ビットデータ領域用セクション（初期値なし，near領域配置）
.bss	BSS	データ領域用セクション（初期値なし，near領域配置）
.bssf	BSSF	データ領域用セクション（初期値なし，far領域配置）
.sbss	SBSS	データ領域用セクション（初期値なし，saddr配置変数）
.option_byte	OPT_BYTE	ユーザ・オプション・バイト，およびオンチップ・デバッグ指定専用セクション
.security_id	SECUR_ID	セキュリティID指定専用セクション
.vect注	AT	割り込みベクタテーブル
.dataR	DATA	初期化データRAM用セクション（初期値あり、near領域配置） スタートアップファイルで定義
.sdataR	DATA	初期化データRAM用セクション（初期値あり、saddr領域配置） スタートアップファイルで定義
.RLIB注	TEXTF	ランタイム・ライブラリ　コード用セクション
.SLIB注	TEXTF	標準ライブラリ　コード用セクション

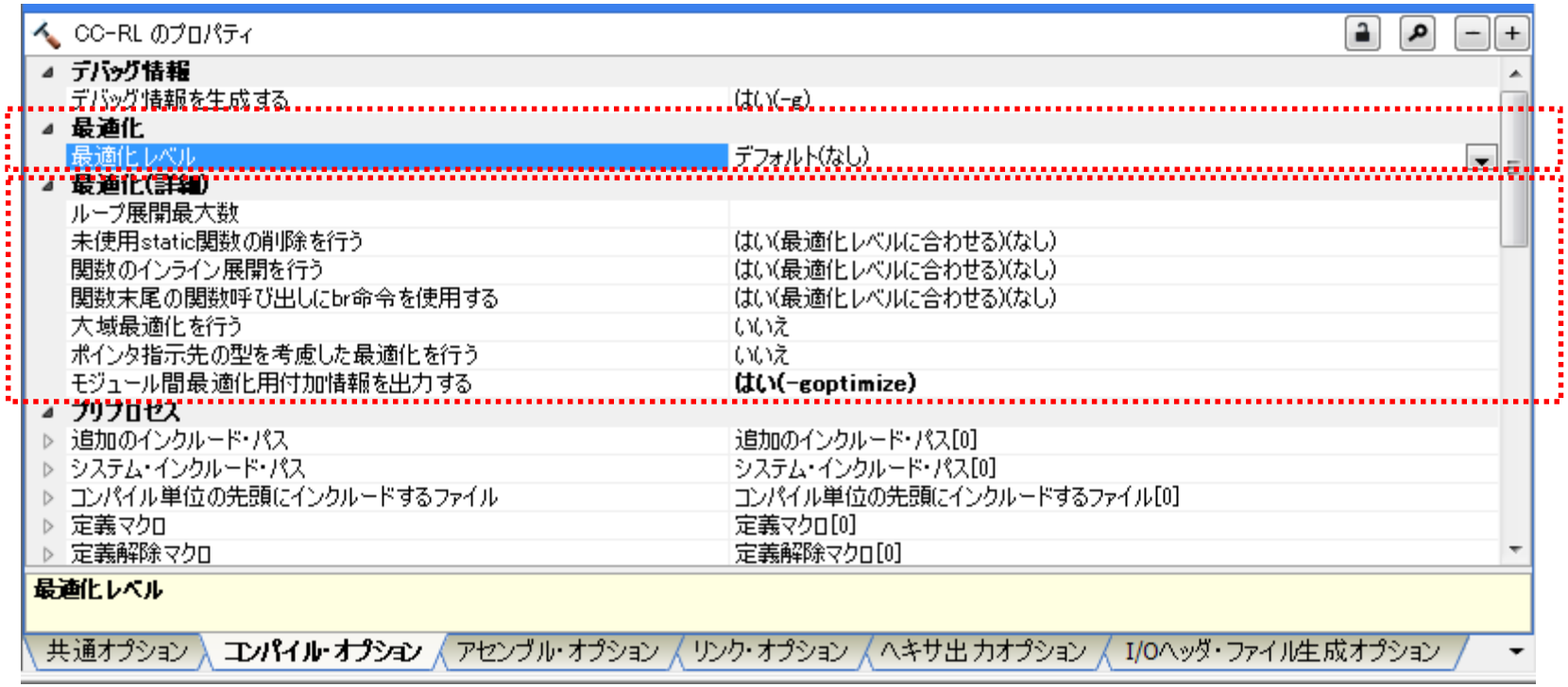
最適化オプション (1/4)

Renesas製 コンパイラ、リンカの最適化技術を適用し、RL78マイコンに適した更なる最適化へ強化（最適なレジスタ割り付け、最適命令選択、命令スケジューリング等）、コンパクトなコードを生成

- ●コンパイラの最適化
  - -最適化の簡単選択
    - – Size or Speed優先の選択
  - -コンパイル時の広範囲最適化
    - – ファイル間にまたがる関数インライン展開
  - -最適化の詳細設定
    - – ループ展開、関数のインライン展開、関数末尾の関数呼び出しのbr命令置き換え　etc
- ●最適化リンカの最適化
  - -モジュール間の最適化
    - – 分岐命令の最適化
    - – 最適化抑止の詳細設定

最適化オプション (2/4)

CC-RL（ビルド・ツール）プロパティのコンパイル・オプションタブで設定



最適化オプション (3/4)

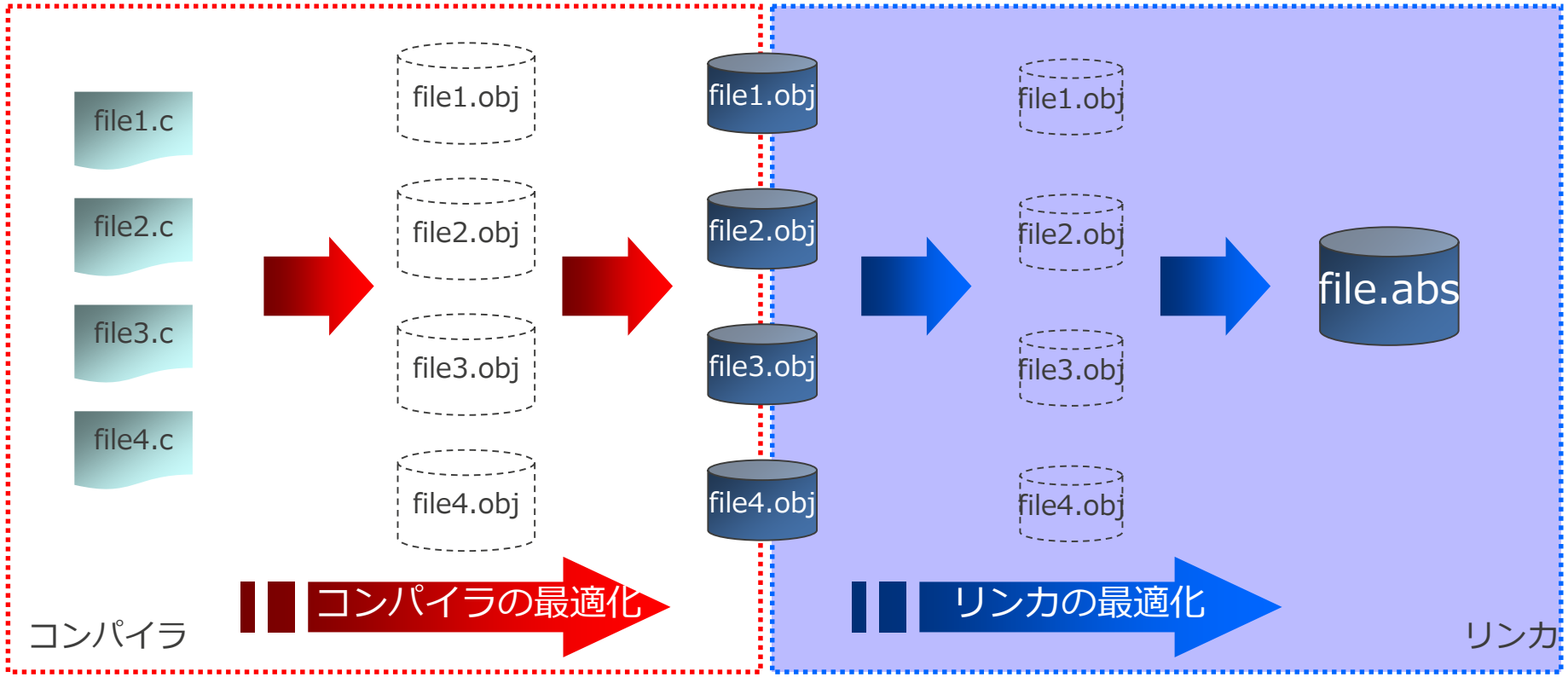
ROMサイズ優先か実行スピード優先の最適化を実施するかの選択で設定。さらに詳細設定でチューニングも可能。

最適化項目	説明	最適化レベル			
		-Osize	-Ospeed	-Odefault	-Onothing
unroll	ループ展開（最大何倍の展開を行うか）	1	2	1	1
delete_static_func	未使用static関数の削除	on	on	on	off
inline_level	関数のインライン展開（展開レベル）※1	3	2	3	-
inline_size	インライン展開サイズ　※2	0	100	0	-
tail_call	関数末尾の関数呼び出しのbr命令置き換え	on	on	on	off

※1　展開のレベル：
　　0：#pragma inline指定した関数を含めて，すべてのインライン展開を抑止します。
　　1：#pragma inline指定した関数のみ展開します。
　　2：自動的に展開対象の関数を判別して展開します。
　　3：コード・サイズがなるべく増加しない範囲で，自動的に展開対象の関数を判別して展開します。
　　ただし，1～3を指定した場合でも，関数の内容やコンパイル状況により，#pragma inline指定した関数が展開されない場合があります。
※2　インライン展開サイズ：コード・サイズが何%増加するまでインライン展開を行うかの指定です。

最適化オプション(4/4)

RL78のビルド環境では、コンパイラの最適化のほかにリンカにも最適化有り。リンク時の情報（配置アドレス等）を使用した最適化を実施し、より効率の良いコードを生成。



改版履歴

版数	内容	箇所
Rev. 1.00	初版	―
Rev. 1.01	CS+ バージョンの変更	P2
	iodefine.h のインクルード方法の説明追加	P12
Rev. 1.02	CS+とCC-RLのバージョンの変更	―

ルネサス システムデザイン株式会社