User's Manual

CcnvCA78K0

Cソースコンバータ

ユーザーズマニュアル

対象デバイス

RL78 ファミリ

対象バージョン

V1.00.00 以降

本資料に記載の全ての情報は本資料発行時点のものであり、ルネサス エレクトロニクスは、
予告なしに、本資料に記載した製品または仕様を変更することがあります。
ルネサス エレクトロニクスのホームページなどにより公開される最新情報をご確認ください。

ルネサス エレクトロニクス
www.renesas.com

Rev.1.00　2016.04

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。

(2012. 4)

このマニュアルの使い方

このマニュアルは，RL78ファミリ用アプリケーション・システムを開発する際のCソースコンバータ（CcnvCA78K0）について説明します。

対象者　　このマニュアルは，RL78 ファミリ用コンパイラ CC-RL を使用してアプリケーション・システムを開発するユーザを対象としています。

目的　　このマニュアルは，78K0 マイクロコントローラ用 C コンパイラ CA78K0，CC78K0 の開発環境を，RL78 ファミリ用 C コンパイラ CC-RL 用に移行するための資料として役立つことを目的としています。

構成　　このマニュアルは，大きく分けて次の内容で構成しています。

1. 概要
2. コマンド・リファレンス
3. コンバータ変換仕様
4. メッセージ
5. 注意事項

読み方　　このマニュアルを読むにあたっては，電気，論理回路，マイクロコンピュータに関する一般知識が必要となります。

凡例

データ表記の重み	：	左が上位桁，右が下位桁
注	：	本文中についた注の説明
注意	：	気を付けて読んでいただきたい内容
備考	：	本文中の補足説明
数の表記	：	10 進数 ... XXXX 16 進数 ... 0xXXXX

CA78K0，CC78K0，CC-RL については下記のマニュアルを参照してください。最新版はルネサス エレクトロニクスのホームページに掲載されています。

コンパイラ	資料名	資料番号
CA78K0	CubeSuite+ V1.03.00 統合開発環境 ユーザーズマニュアル 78K0 コーディング編 CubeSuite+ V1.01.00 統合開発環境 ユーザーズマニュアル 78K0 ビルド編	R20UT2141JJ0100 R20UT0783JJ0100
CC78K0	ユーザーズ・マニュアル CC78K0 Ver.3.70 C コンパイラ言語編 ユーザーズ・マニュアル CC78K0 Ver.3.70 C コンパイラ操作編	U17200JJ1V0UM00 U17201JJ1V0UM00
CC-RL	RL78 コンパイラ CC-RL ユーザーズマニュアル	R20UT3123JJ0102

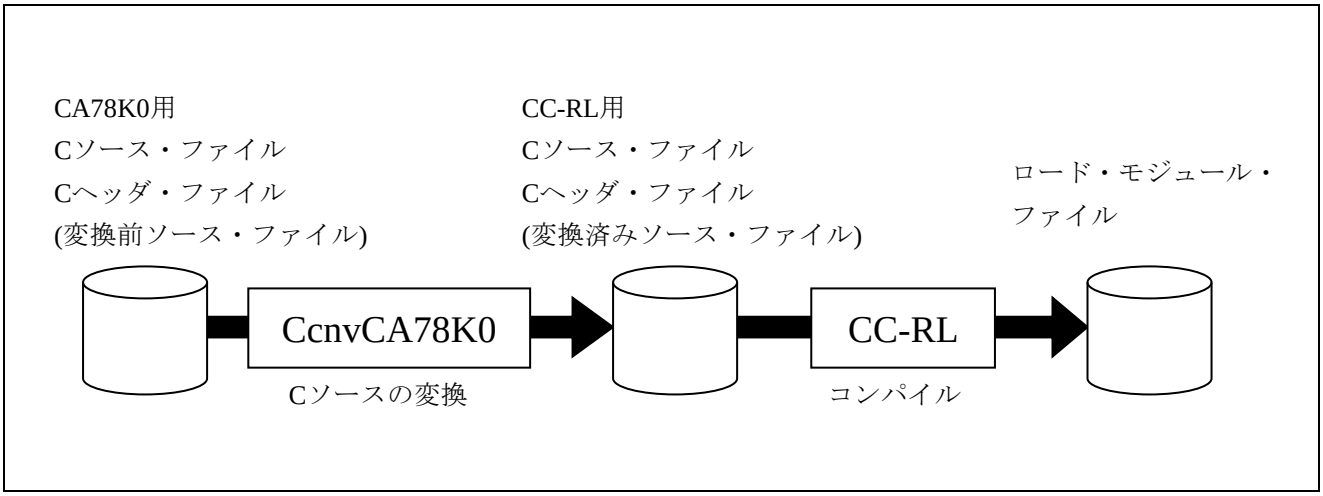
目次

1. 概要

CcnvCA78K0 は, 78K0 マイクロコントローラ用 C コンパイラである CA78K0, CC78K0(以降は, CA78K0 と CC78K0 をまとめて CA78K0 と記述します)の開発環境で作成した C ソース・プログラムを，RL78 ファミリ用 C コンパイラである CC-RL で動作する C ソース・プログラムに変換する，C ソースコンバータです。C ソース中に記述した CA78K0 用拡張機能を，CC-RL 用拡張機能に変換します。

CcnvCA78K0 は，CA78K0 用プログラムから CC-RL 用プログラムへの移行を支援するためのソフトウェアです。変換後のプログラムが完全に動作することは保証しません。必ず変換後の C ソースを用いてプログラムの動作確認をしてください。

図 1.1 CcnvCA78K0 の位置づけ



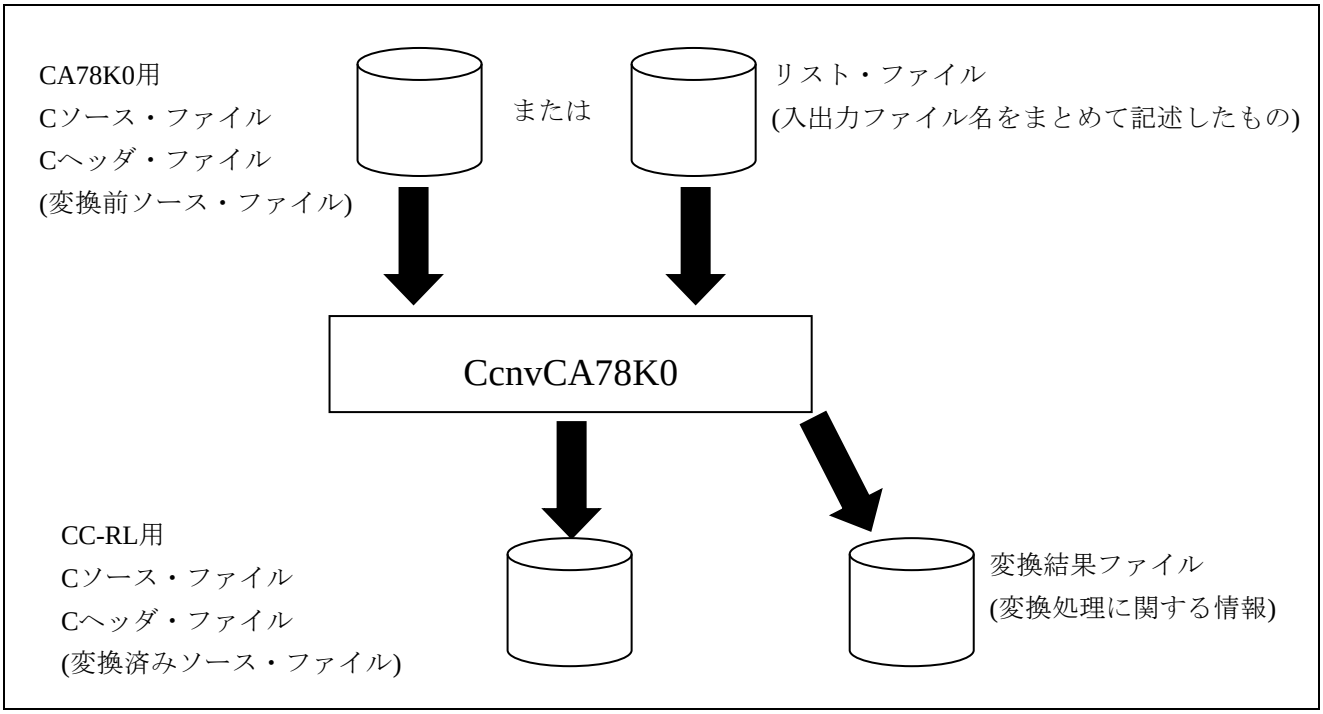
2. コマンド・リファレンス

この章では，CcnvCA78K0 における処理の流れについて説明します。

2.1 概要

CA78K0 用の C ソース・プログラムに対して，拡張言語仕様であるマクロ名，予約語，#pragma 指令，拡張機能の記述を，CC-RL の拡張言語仕様に変換し，CC-RL 用の C ソース・プログラムを生成します。

図 2.1 CcnvCA78K0 における処理の流れ



2.2 入出力ファイル

CcnvCA78K0 の入出力ファイルを以下に示します。

表 2.1 入出力ファイル

ファイル種別	入出力	拡張子	説明
C ソース・ファイル C ヘッダ・ファイル	入出力	(入力) .c .h (出力) 任意	CA78K0 用の C ソース・ファイルまたは C ヘッダ・ファイルを入力し，変換した CC-RL 用の C ソース・ファイルまたは C ヘッダ・ファイルを出力します。変換後のファイルは，先頭に CcnvCA78K0 のバージョン情報をコメントとして挿入し，変換箇所は元の記述をコメントとして残します。 入力ファイルの拡張子は固定です。他の拡張子を持つファイルを指定した場合は，入力ファイルの内容を変換せずそのまま出力します。 変換後のファイルは-o オプションまたは-l オプションで指定します。 変換後のファイルを再入力した場合は，変換せずそのまま出力し，既に変換済みであることを通知します。
リスト・ファイル	入力	任意	入力ファイル名と出力ファイル名を記述したテキスト･ファイルです。 -l オプションでリスト・ファイルを指定することで，複数のソース・ファイルをまとめて変換することができます。リスト・ファイルの書式については，-l オプションを参照してください。
変換結果ファイル	出力	任意	標準出力に出力する変換結果の中で，メッセージについては，-r オプションで指定したファイルに出力することができます。 メッセージの内容については，「メッセージ」を参照してください。

入力ファイルと出力ファイルの例を示します。変換仕様の詳細については「コンバータ変換仕様」を参照してください。

(入力ファイル：input.c)

```
#pragma sfr
char c;
void main(void)
{
    c = P0;
}
```

(出力ファイル：output.c)

```
/* CA78K0 C Source Converter Vx.xx.xx.xx [dd Mmm yyyy] */
/************************************************************************************
 DISCLAIMER
 This software is supplied by Renesas Electronics Corporation and is only
 intended for use with Renesas products. No other uses are authorized. This
 software is owned by Renesas Electronics Corporation and is protected under
 all applicable laws, including copyright laws.
 THIS SOFTWARE IS PROVIDED "AS IS" AND RENESAS MAKES NO WARRANTIES REGARDING
 THIS SOFTWARE, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT
 LIMITED TO WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE
 AND NON-INFRINGEMENT. ALL SUCH WARRANTIES ARE EXPRESSLY DISCLAIMED.
 TO THE MAXIMUM EXTENT PERMITTED NOT PROHIBITED BY LAW, NEITHER RENESAS
 ELECTRONICS CORPORATION NOR ANY OF ITS AFFILIATED COMPANIES SHALL BE LIABLE
 FOR ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR
 ANY REASON RELATED TO THIS SOFTWARE, EVEN IF RENESAS OR ITS AFFILIATES HAVE
 BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 Renesas reserves the right, without notice, to make changes to this software
 and to discontinue the availability of this software. By using this software,
 you agree to the additional terms and conditions found by accessing the
 following link:
 http://www.renesas.com/disclaimer
 Copyright (C) yyyy Renesas Electronics Corporation. All rights reserved.
************************************************************************************/
//[CcnvCA78K0]
#include "iodefine.h"

//[CcnvCA78K0] #pragma sfr
char c;
void main(void)
{
    c = P0;
}
```

2.3 変換結果

CcnvCA78K0 は変換結果を標準出力に出力します。出力形式は，以下の通りです。

```
メッセージ

入力ファイル名
    結果
    メッセージ個数
```

-l オプション指定時は，リスト・ファイルで指定したファイル数の分だけ，上記の出力を繰り返します。

「メッセージ」はエラーやワーニングなどがある場合に出力します。メッセージの出力形式については「メッセージ」を参照してください。-r オプション指定時は，メッセージを標準出力ではなく指定したファイルに出力します。

「入力ファイル名」はコマンドラインまたはリスト・ファイルで指定した入力ファイルです。

「結果」は以下のいずれかを表示します。

- 変更箇所がある場合

```
変換しました。
```

- 変更箇所がない場合

```
変更箇所はありませんでした。
```

- 変換したファイルを再度 CcnvCA78K0 に入力した場合

```
既に変換されています。
```

- エラーが発生した場合

```
変換に失敗しました。
```

「メッセージ個数」は出力したメッセージの個数をメッセージの種別ごとに表示します。

変換結果の例を示します。

(入力ファイル：input.c)

```
#pragma sfr
char c;
void main(void)
{
    c = P0;
}
```

(標準出力)

```
CA78K0 C Source Converter Vx.xx.xx.xx [dd Mmm yyyy]

input.c(1):M0592123:[追加]#include "iodefine.h" を生成しました。
input.c(1):M0592131:[削除]#pragma sfr を削除しました。
input.c(1):M0592146:[情報]78K0 に依存した言語仕様です。

input.c
                変換しました。
                1 deleted, 1 inserted, 0 changed, 1 information
                Total warning(s) : 0
```

2.4 操作方法

コマンドラインで以下のように入力します。

CcnvCA78K0[△*option*]…[△*file*][△*option*]…

file : 入力ファイル名
option : オプション名
[ ] : [ ]内は省略可能
… : 直前の[ ]内のパターンの繰り返しが可能
{ } : ｜で区切られた項目を選択
△ : 1 個以上の空白

- 入力ファイル名およびオプションで指定するファイル名は，Windows で認められるものであれば指定可能です。
- 入力ファイル名およびオプションで指定するファイル名は，絶対パスまたは相対パスでの指定が可能です。入力ファイル名およびオプションで指定するファイル名を，パスなしまたは相対パスで指定する場合は，カレント・フォルダを基準とします。
- 入力ファイル名およびオプションで指定するファイル名に空白を含める(パス名を含む)場合は，パス名を含むファイル名をダブルクォーテーションで囲んでください。
- 入力ファイル名およびオプションで指定するファイル名の長さは，Windows に依存します(259 文字まで)。
- 入力ファイル名を複数指定した場合はエラーとなります。複数の入力ファイル名を指定したい場合は，-l オプションを使用してください。
- 入力ファイルを指定した場合は，必ず出力ファイル名を指定する必要があります。コマンドラインで入力ファイルを指定した場合は，-o オプションで出力ファイルを指定してください。
- 同じオプションを複数回指定した場合はエラーとなります。

2.5 オプション

オプションについて説明します。

- オプションの大文字／小文字は区別します。
- パラメータとしてファイル名を指定する場合は，パス付き(絶対パス，または相対パス)での指定が可能です。パスなし，または相対パスで指定する場合は，カレント・フォルダを基準とします。
- パラメータ中に空白を含める場合(パス名など)は，そのパラメータ全体をダブルクォーテーションで囲んでください。

表 2.2 オプション

オプション	説明
-V	CcnvCA78K0 のバージョン情報を表示します。
-h	CcnvCA78K0 のオプションの説明を表示します。
-c	日本語の文字コードを指定します。
-l	リスト・ファイル名を指定します。
-o	出力ファイル名を指定します。
-r	メッセージの出力先を指定します。
-A	ANSI 規定に関する機能を有効にした状態で変換します。

-V

CcnvCA78K0 のバージョン情報を表示します。

［指定形式］

```
-V
```

・省略時解釈
CcnvCA78K0 のバージョン情報を表示しません。

［詳細説明］

- CcnvCA78K0 のバージョン情報を標準エラー出力に表示します。
- 本オプション指定時は変換しません。
- 他のオプションを同時に指定した場合は，他のオプションを無視します。

［使用例］

```
>CcnvCA78K0　-V
```

-h

CcnvCA78K0 のオプションの説明を表示します。

［指定形式］

```
-h
```

・省略時解釈

CcnvCA78K0 のオプションの説明を表示しません。

［詳細説明］

- CcnvCA78K0 のオプションの説明を標準エラー出力に表示します。
- 本オプション指定時は変換しません。
- 他のオプションを同時に指定した場合は，他のオプションを無視します。
- -V を同時に指定した場合は，-V を優先します。

［使用例］

```
>CcnvCA78K0    -h
```

-C

日本語の文字コードを指定します。

［指定形式］

```
-c={none | sjis | euc_jp}
```

・省略時解釈
本オプションのパラメータに sjis を指定したものと見なします。

［詳細説明］

- 入力ファイル中のコメントに対して，使用する文字コードを指定します。
- パラメータを省略した場合は，エラーとなります。
- パラメータに指定可能なものを以下に示します。これ以外のものを指定した場合は，ワーニングを出力し sjis として扱います。なお，入力ファイル中で使用している文字コードと異なるものを指定した場合，動作は保証されません。

none	日本語の文字コードを処理しません
sjis	SJIS
euc_jp	EUC(日本語)

［使用例］

```
>CcnvCA78K0　input.c　-c=euc_jp　-o=output.c
```

-l

リスト・ファイル名を指定します。

［指定形式］

-l=*file*

・省略時解釈
コマンドラインで指定したファイルを変換します。

［詳細説明］

- 複数のファイルを同時に変換する場合に指定します。
- 指定したリスト・ファイルが存在しない場合はエラーとなります。
- 本オプションを指定した場合，コマンドラインで指定したファイル名はワーニングを出力し無視します。
- -o オプションと同時に指定した場合，ワーニングを出力し，-o オプションを無視します。
- パラメータを省略した場合は，エラーとなります。
- リスト・ファイルの書式は以下の通りです。

```
[-c={none｜sjis｜euc_jp}] [-A]  入力ファイル名  出力ファイル名
[-c={none｜sjis｜euc_jp}] [-A]  入力ファイル名  出力ファイル名
(以下略)
```

[ ]　　：[ ]内は省略可能
{ }　　：｜で区切られた項目を選択

- 一行に-c オプション，-A オプション，入力ファイル名，出力ファイル名の順に指定します。
- -c オプション，-A オプションは省略可能です。入出力ファイル名は省略できません。
- 記述可能な入出力ファイル名はコマンドラインでの指定時と同じです。
- ファイル名に空白文字を含む場合は，ファイル名をダブルクォーテーションで囲んでください。
- -c オプションの指定がコマンドラインとリスト・ファイル内とで異なる場合，ワーニングを出力しリスト・ファイルの指定を優先します。
- 出力ファイルが既に存在する場合は，ワーニングを出さずに上書きします。
- 出力ファイル名が，入力ファイル名または-r オプションで指定したファイル名と一致する場合は，エラーとなります。
- リスト・ファイルの日本語文字コードは UTF-8N(BOM なし)のみ，改行コードは CR+LF のみ受容します。

［使用例］

```
>CcnvCA78K0    -l=listfile.txt
```

・リスト・ファイル(listfile.txt)の中身

```
-c=sjis  input¥file1.c  output¥file1.c

-c=sjis  input¥file2.c  output¥file2.c

-c=sjis  input¥file.h  output¥file.h
```

-o

出力ファイル名を指定します。

［指定形式］

-o=*file*

・省略時解釈
-V, -h, -l オプション指定時を除き，省略できません。省略した場合は，エラーとなります。

［詳細説明］

- 変換後の出力ファイル名を指定します。
- 指定したファイルが既に存在する場合は，ワーニングを出さずに上書きします。
- 出力ファイル名が入力ファイル名または-r オプションで指定したファイル名と一致する場合は，エラーとなります。
- -l オプションと同時に指定した場合，ワーニングを出力し，本オプションを無視します。
- パラメータを省略した場合は，エラーとなります。

［使用例］

```
>CcnvCA78K0    input.c    -o=output.c
```

-r

メッセージを指定したファイルに出力します。

［指定形式］

-r=*file*

・省略時解釈
メッセージを標準出力に出力します。

［詳細説明］

- メッセージを指定したファイルに出力します。
- 指定したファイルが既に存在する場合は，ワーニングを出さずに上書きします。
- 指定したファイル名が C ソース・ファイルや C ヘッダ・ファイルの入出力ファイル名と一致する場合は，エラーとなります。
- パラメータを省略した場合は，エラーとなります。

［使用例］

```
>CcnvCA78K0    input.c    -o=output.c    -r=input.txt
```

-A

CA78K0 の ANSI 規格準拠オプション-za を有効とした状態で変換します。

［指定形式］

```
-A
```

・省略時解釈
CA78K0 の ANSI 規格準拠オプション-za を無効とした状態で変換します。

［詳細説明］

- 本オプション指定時は，以下をキーワードと見なさず，変換しません。

```
callt,  sreg,  boolean,  bit
```

- 本オプションは，変換前の CA78K0 開発環境におけるオプション-za の有無に合わせてください。

［使用例］

```
>CcnvCA78K0  input.c  -o=output.c  -A
```

3. コンバータ変換仕様

この章では，CcnvCA78K0 の変換仕様を示します。

- CA78K0 の構文的に誤った C ソースを入力した場合は，動作を保証しません。
- コメントおよび文字列に含まれている内容については変換しません。
- コメントのネストに対応していません。ネスト構造のコメント文は正常に認識せず，コメントの範囲が不正となります。コメントのネストがないか，変換前に確認してください。
- 変換対象のキーワードが##演算子等で生成される等，キーワードとして見つけることができない場合は，変換できません。そのまま CC-RL でコンパイルすると，コンパイル・エラーとなります。変換キーワードに対する#define，typedef，##演算子がないかどうか，変換前に確認してください。
- CC-RL ではメモリ配置領域を__near/__far キーワードによって指定します。CC-RL でのメモリ・モデル指定により__near/__far キーワード省略時のデフォルト動作が変わるため，変換後にポインタの型が合わなくなる可能性があります。変換後の C ソースに対して，CC-RL ではスモール・モデルを指定してください。78K0 のメモリバンク機能を持つマイコンの場合，バンク 2 以降の関数には__far キーワードを指定して，Far 属性の関数にしてください。
- C ソース中でインクルードしているファイルは変換しません。別途，変換してください。

下記の拡張言語仕様に対して変換します。

- マクロ名
- 予約語
- ビット・アクセス
- #pragma section
- ASM 文
- 割り込み関数
- RTOS 割り込みハンドラ
- RTOS 用タスク関数
- 絶対番地配置指定
- 組み込み関数
- その他の#pragma 指令
- 標準ライブラリ関数

3.1 マクロ名

CA78K0 でサポートしているマクロは，以下の通り変換します。CC-RL に対応するマクロがない場合は，メッセージを出力します。CPU マクロは変換せず，メッセージも出力しません。

表 3.1 マクロ名の変換

CA78K0 マクロ名	変換後	備考
__LINE__	変換しません	そのまま CC-RL で使用できます。
__FILE__	変換しません	そのまま CC-RL で使用できます。
__DATE__	変換しません	そのまま CC-RL で使用できます。
__TIME__	変換しません	そのまま CC-RL で使用できます。
__STDC__	変換しません	そのまま CC-RL で使用できます。
__K0__	変換しません	メッセージを出力します。 CC-RL ではユーザ定義マクロとして扱います。
__STATIC_MODEL__	変換しません	メッセージを出力します。 CC-RL ではユーザ定義マクロとして扱います。
__CHAR__UNSIGNED__	__UCHAR	
__CA78K0__	変換しません	メッセージを出力します。 CC-RL ではユーザ定義マクロとして扱います。
CPU マクロ	変換しません	メッセージを出力しません。 CC-RL ではユーザ定義マクロとして扱います。

3.2 予約語

予約語に対する変換仕様を示します。

表 3.2 予約語の変換

CA78K0 の予約語	変換後	備考
__callt	変換しません	そのまま CC-RL で使用できます。
callt	__callt	-A オプションが無効の場合のみ変換します。
__callf	削除	
callf	削除	-A オプションが無効の場合のみ削除します。
__sreg	__saddr	常に変換します。
sreg	__saddr	-A オプションが無効の場合のみ変換します。
noauto	削除	-A オプションが無効の場合のみ削除します。
__leaf	削除	
norec	削除	-A オプションが無効の場合のみ削除します。
__boolean	_Bool	CC-RL にて-ansi オプションを指定する場合は, _Bool 型を char 型に変更してください。
boolean	_Bool	-A オプションが無効の場合のみ変換します。
bit	_Bool	-A オプションが無効の場合のみ変換します。
__interrupt	#pragma interrupt	詳細は「割り込み関数」を参照してください。
__interrupt_brk	#pragma interrupt_brk	詳細は「割り込み関数」を参照してください。
__asm	#pragma inline_asm	詳細は「ASM 文」を参照してください。
__rtos_interrupt	#pragma rtos_interrupt	詳細は「RTOS 割り込みハンドラ」を参照してください。
__directmap	#pragma address	詳細は「絶対番地配置指定」を参照してください。
__pascal	削除	
__flash	削除	
__flashf	削除	
__temp	削除	
__mxcall	削除	
バンク関数 (__BANK0, …)	変換しません	メッセージを出力します。 CC-RL ではユーザ定義関数として扱います。

3.3 ビット・アクセス

CC-RL は，CA78K0 のビット・アクセス(SFR 及び saddr 変数に対して，ピリオドに続けてビット位置を指定する形式)に対応していません。CcnvCA78K0 では，SFR 及び saddr 変数に対するビット・アクセスを，ビットフィールドの型宣言とマクロで置き換えます。

- ファイル先頭に型宣言およびマクロを出力し，アクセス箇所では，マクロ呼び出しに変更します。
- ビット・アクセスでは，ビット位置に応じて 8,16 ビットのビットフィールドを作成します。ビット位置が 8～15 を含む場合は，b8～b15 を追加したビットフィールドを，16 ビット用として別途作成します。

［例］

- ビット位置が 0～7 のみの場合

変換前	void func(void)
	{
	  i = var.3;
	  var.5 = 0;
	}

変換後	#ifndef __BIT8
	typedef struct {
	  unsigned int b0:1;
	  unsigned int b1:1;
	  unsigned int b2:1;
	  unsigned int b3:1;
	  unsigned int b4:1;
	  unsigned int b5:1;
	  unsigned int b6:1;
	  unsigned int b7:1;
	} __Bits8;
	#define __BIT8(name,bit) (((volatile __near __Bits8*)&name)->b##bit)
	#endif
	
	void func(void)
	{
	  i = __BIT8(var,3);
	  __BIT8(var,5) = 0;
	}

・ ビット位置が 8～15 を含む場合

変換前	void func(void)
	{
	  i = var2.10;
	  var2.12 = 0;
	}

変換後	#ifndef __BIT16
	typedef struct {
	  unsigned int b0:1;
	  unsigned int b1:1;
	  unsigned int b2:1;
	  unsigned int b3:1;
	  unsigned int b4:1;
	  unsigned int b5:1;
	  unsigned int b6:1;
	  unsigned int b7:1;
	  unsigned int b8:1;
	  unsigned int b9:1;
	  unsigned int b10:1;
	  unsigned int b11:1;
	  unsigned int b12:1;
	  unsigned int b13:1;
	  unsigned int b14:1;
	  unsigned int b15:1;
	} __Bits16;
	#define __BIT16(name,bit) (((volatile __near __Bits16*)&name)->b##bit)
	#endif
	
	void func(void)
	{
	  i = __BIT16(var2,10);
	  __BIT16(var2,12) = 0;
	}

3.4 #pragma section

#pragma section は，CA78K0 と CC-RL とでセクション名が異なるため，セクション名の変換処理が必要になります。ただし，セクションによっては CC-RL 側に対応するセクションが存在しないケースもあるため，変換が不可能なセクションも存在します。また，変換は可能であっても機能が少し異なるものもあります。CcnvCA78K0 ではいくつかのセクションの変換時に標準エラー出力にメッセージを出力します。詳細はセクション名対応表を参照してください。

CA78K0 の書式は以下の通りです。

#pragma section **セクション名 変更後セクション名** [AT **開始アドレス**]

CC-RL の書式は以下の通りです。

#pragma section [{text | const | data | bss}] [**変更後セクション名**]

- 「AT 開始アドレス」に相当する機能が CC-RL には存在しないため，「AT 開始アドレス」がある場合は削除し，メッセージを出力します。CC-RL でのセクションの配置は，-start オプションにて指定してください。-start オプションの詳細については，CC-RL のユーザーズ・マニュアルを参照してください。
- 「変更後セクション名」は変換せずそのまま出力します。変更後セクション名に CC-RL では使用不可能な文字（’?’など）を使っている場合は，CC-RL ではコンパイル・エラーとなります。変換後に文字列を変更してください。
- CC-RL の#pragma section におけるセクション名は「変更後セクション名+”_n”」または「変更後セクション名+”_f”」，saddr 領域用のセクション名は「変更後セクション名+”_s”」となります。詳細は CC-RL のユーザーズ・マニュアルを参照してください。
- CC-RL 側に対応するセクションがないため変換できない場合は，メッセージを出力し，変換しません。CC-RL ではワーニングを出力し，#pragma 指令を無視します。後述のセクション名対応表に従い，C ソースを修正してください。

[例]

パターン 1 (正常に置換)	変換前	#pragma section @@CODE MY_CODE
	変換後	#pragma section text MY_CODE
パターン 2 (AT の削除)	変換前	#pragma section @@CODE MY_CODE AT 0x2000
	変換後	#pragma section text MY_CODE
パターン 3 (置換後コンパイル・エラー)	変換前	#pragma section @@CODE ??CODE AT 0x2000
	変換後	#pragma section text ??CODE
	対応	変換しますが，コンパイル時にエラーとなります。 セクション名を変更してください。
パターン 4 (置換不可能)	変換前	#pragma section @@CALF　MY_BASE
	変換後	#pragma section @@CALF　MY_BASE
	対応	CC-RL に対応するセクションがないため，変換せず出力します。 セクション名対応表に従い，修正してください。

表 3.3 セクション名対応表

CA78K0 セクション名	説明	CC-RL セクション 種別	CcnvCA78K0 の動作 変換後の対応
@@CODE @ECODE	コード部用セグメント	text	対応したセクション種別に変更します。 不要です。 CC-RL でのセクション名は「変更後セクション名+”_n”」 または「変更後セクション名+”_f”」となります。
@@LCODE @LECODE	ライブラリ・コード用 セグメント	text	変換しません。 #pragma を削除してください。 CC-RL でのライブラリの配置は，リンク・オプション-ROm で指定してください。
@@CNST @ECNST	ROM データ用 セグメント	const	対応したセクション種別に変更します。 不要です。 CC-RL でのセクション名は「変更後セクション名+”_n”」となります。
@@R_INIT @ER_INIT	初期化データ用 セグメント	data	対応したセクション種別に変更します。 不要です。 CC-RL でのセクション名は「変更後セクション名+”_n”」となります。
@@R_INIS @ER_INIS	初期化データ用 セグメント (sreg 変数)	data	対応したセクション種別に変更します。 不要です。 CC-RL でのセクション名は「変更後セクション名+”_s”」となります。
@@CALF	callf 関数用 セグメント	なし	メッセージを出力し，変換しません。 #pragma を削除してください。 CC-RL には対応する機能がありません。処理の見直しが必要です。
@@CALT	callt 関数テーブル用 セグメント	なし	メッセージを出力し，変換しません。 #pragma を削除してください。 CC-RL ではセクション名を変更できません。
@@VECTnn @EVECTnn	ベクタ・テーブル用 セグメント	なし	変換しません。 #pragma を削除してください。 CC-RL ではセクション名を変更できません。
@EXTxx	フラッシュ領域 分岐テーブル用 セグメント	なし	変換しません。 #pragma を削除してください。 CC-RL には対応する機能がありません。処理の見直しが必要です。

CA78K0 セクション名	説明	CC-RL セクション 種別	CcnvCA78K0 の動作 変換後の対応
@@INIT @EINIT	データ領域用セグメント (初期値あり)	なし	メッセージを出力し，変換しません。 #pragma を削除してください。 ROM から RAM へマップするセクションは，リンク・オプション-ROm で指定してください。
@@INIS @EINIS	データ領域用セグメント (sreg 変数，初期値あり)	なし	メッセージを出力し，変換しません。 #pragma を削除してください。 ROM から RAM へマップするセクションは，リンク・オプション-ROm で指定してください。
@@DATA @EDATA	データ領域用セグメント (初期値なし)	bss	対応したセクション種別に変更します。 不要です。 CC-RL でのセクション名は「変更後セクション名+”_n”」となります。
@@DATS @EDATS	データ領域用セグメント (sreg 変数，初期値なし)	bss	対応したセクション種別に変更します。 不要です。 CC-RL でのセクション名は「変更後セクション名+”_s”」となります。
@@BITS @EBITS	boolean,bit 型変数用 セグメント	なし	メッセージを出力し，変換しません。 #pragma を削除してください。 CC-RL では_Bool 型として，他のデータと同じセクションに配置されます。
@@BANK0, …, @@BANK15	バンク関数用 セグメント	なし	メッセージを出力し，変換しません。 #pragma を削除してください。 CC-RL には対応する機能がありません。処理の見直しが必要です。

3.5 ASM 文

CA78K0 では__asm()関数や#asm～#endasm を用いて関数中にアセンブリ記述をしますが，CC-RL では #pragma inline_asm で宣言したアセンブリ記述関数をインライン展開します。CcnvCA78K0 では，__asm() や#asm～#endasm 内のアセンブラ命令を実行する inline_asm 関数をファイル先頭に作成し，アセンブラ命令の記述箇所ではこの関数を呼び出すように変換します。

CA78K0 の書式は以下の通りです。

```
#asm
  :     /* アセンブリ記述 */
#endasm
```

```
__asm("アセンブリ記述");
```

CC-RL の書式は以下の通りです。

```
#pragma inline_asm [(] 関数名 [, … ] [)]
関数宣言 {
  :     /* アセンブリ記述 */
}
```

- 78K0 と RL78 は命令セットや命令の仕様が異なるため，手動によるアセンブラ記述の修正が必要です。変換時にメッセージを出力します。
- inline_asm 関数内のアセンブリ記述に，インデントとしてタブ文字を追加します。
- 作成する関数名は__inline_asm_func_00000～__inline_asm_func_99999 とし，関数の数が 100000 件を超える場合は，エラーとなります。
- #asm～#endasm または__asm の中にラベルがある場合には，メッセージを出力します。CC-RL にて, #pragma inline_asm を指定した関数の中にラベルを書くと, コンパイル時にエラーとなります。このため CcnvCA78K0 では，#asm～#endasm または__asm の中にラベルがある場合には，メッセージを出力します。コンパイル・エラーを回避するには，アセンブリ記述のラベルをローカル・ラベルに変更する必要があります。詳細は，CC-RL のユーザーズ・マニュアルを参照してください。
- 以下のようにダブルクォートを含めて#define マクロの変換対象とする場合は，__asm( )関数から inline_asm 関数を生成できません。このような場合は，メッセージを出力し，入力ファイルの内容を変換せずそのまま出力します。予めマクロを展開した状態に修正してから変換してください。

  例） #define MAC “nop”
  __asm(MAC);
- __asm( )内の文字列に'¥n'や'¥t'といった制御文字を含む場合は，変換後にアセンブル・エラーとなります。予め制御文字を削除してから変換してください。
- #asm～#endasm 内のアセンブリ記述のコメント(“;”)に C 記述のコメント(“/*”)を含む場合は，変換後にコメントの範囲が不正となります。予めコメントを削除してから変換してください。

［例］

パターン 1	変換前	void func(void)
		{
		  __asm("nop");
		}

	変換後	#pragma inline_asm __inline_asm_func_00000
		static void __inline_asm_func_00000(void)
		{
		  nop
		}
		
		void func(void)
		{
		  __inline_asm_func_00000();
		}

パターン 2	変換前	void func(void)
		{
		  #asm
		  nop
		  #endasm
		}

	変換後	#pragma inline_asm __inline_asm_func_00001
		static void __inline_asm_func_00001(void)
		{
		  nop
		}
		
		void func(void)
		{
		  __inline_asm_func_00001();
		}

パターン 3	変換前	#define ASM_NOP __asm("nop");

	変換後	#pragma inline_asm __inline_asm_func_00002
		static void __inline_asm_func_00002(void)
		{
		  nop
		}
		#define ASM_NOP __inline_asm_func_00002();

パターン 4
(変換後にエラー)
	変換前	void func()
		{
		  __asm("¥tnop");
		}

	変換後	#pragma inline_asm __inline_asm_func_00003
		static void __inline_asm_func_00003(void)
		{
		  ¥tnop
		}
		
		void func()
		{
		  __inline_asm_func_00003();
		}

3.6 割り込み関数

CA78K0 の#pragma interrupt/vect およびキーワード__interrupt, __interrupt_brk を，CC-RL の#pragma interrupt/interrupt_brk に変換します。

CA78K0 の割り込み関数の書式は以下の通りです。

```
<ノーマル・モデル>
#pragma interrupt(または vect) 割り込み要求名 関数名
                [スタック切り替え指定] [{スタック使用指定 | 無変更指定 | レジスタバンク指定}]

<スタティック・モデル>
#pragma interrupt(または vect) 割り込み要求名 関数名 [{共有領域退避/復帰指定 | 退避/復帰対象}]
                [{スタック使用指定 | 無変更指定 | レジスタバンク指定}]
```

または

```
__interrupt 関数宣言
__interrupt_brk 関数宣言
```

CC-RL の割り込み関数の書式は以下の通りです。

```
#pragma interrupt [(]関数名[([vect=アドレス][,bank=レジスタバンク][,enable={true|false}])][)]
関数宣言
#pragma interrupt_brk [(]関数名[([bank=レジスタバンク][,enable={true|false}])][)]
関数宣言
```

- 割り込み要求名が存在する場合は，#include “iodefine.h”を出力します。デバイス変更により，割り込み要求名は適切でない場合があるため，メッセージを出力します。
- __interrupt は#pragma interrupt に，__interrupt_brk は#pragma interrupt_brk に変換します。
- 割り込み要求名が BRK_I の場合は，#pragma interrupt_brk に変換します。
- 「割り込み要求名」は，アドレスを示すマクロとして「vect=アドレス」に変換します。マクロの値は iodefine.h で定義されます。
- 「レジスタバンク指定」は，「bank=レジスタバンク」に変換します。
- 「スタック切り替え指定」，「スタック使用指定」，「無変更指定」，「共有領域退避/復帰指定」，「退避/復帰対象」は CC-RL に存在しないため，メッセージを出力し削除します。
- __interrupt, __interrupt_brk キーワードを用いた割り込み関数の宣言や定義にマクロや typedef を使用した場合，関数名の判断を誤る可能性があります。予めマクロや typedef を展開してから変換してください。
- 同一関数に対する pragma 指令とキーワードによる割り込み関数の記述がある場合，どちらも #pragma 指令に変換することで，変換後に#pragma 指令が重複してコンパイル・エラーが発生することがあります。この場合は，重複する記述を削除してください。
- __interrupt, __interrupt_brk キーワードを指定した関数宣言の引数を省略した場合, CC-RL ではコンパイル・エラーとなります。引数の型として void 型を記述してください。

［例］

パターン 1	変換前	#pragma vect INTP0 func sp=buff+10 rb1 void func(void) { }
	変換後	#pragma interrupt func(vect=INTP0, bank=RB1) void func(void) { }
パターン 2	変換前	#pragma interrupt INTP0 func leafwork1 rb1 void func(void) { }
	変換後	#pragma interrupt func(vect=INTP0, bank=RB1) void func(void) { }
パターン 3	変換前	__interrupt void func(void) { }
	変換後	#pragma interrupt func void func(void) { }
パターン 4	変換前	#pragma interrupt BRK_I func void func(void) { }
	変換後	#pragma interrupt_brk func void func(void) { }
パターン 5	変換前	__interrupt void func1(void), func2(void);

	変換後	#pragma interrupt func1
		void func1(void);
		#pragma interrupt func2
		void func2(void);

パターン 6	変換前	#pragma interrupt INTP0 func __interrupt func(void);
	変換後	#pragma interrupt func(vect=INTP0) void func(void); #pragma interrupt func void func(void);
	対応	CC-RL では#pragma 指令の重複でエラーとなります。キーワードから変換した#pragma 指令を削除してください。
パターン 7	変換前	typedef void func_t(void); __interrupt func_t f1;
	変換後	typedef void func_t(void); __interrupt func_t f1;
	対応	CC-RL ではコンパイル・エラーとなります。予め typedef やマクロを展開してください。

3.7 RTOS 割り込みハンドラ

CA78K0 の#pragma rtos_interrupt およびキーワード__rtos_interrupt を，CC-RL の#pragma rtos_interrupt に変換します。

CA78K0 の書式は以下の通りです。

#pragma rtos_interrupt [**割り込み要求名 関数名 [スタック切り替え指定**]]

または

__rtos_interrupt **関数宣言**

CC-RL の書式は以下の通りです。

#pragma rtos_interrupt [(]**関数名**[(vect=**アドレス**)][)] **関数宣言**

- 割り込み要求名が存在する場合は，#include “iodefine.h”を出力します。デバイス変更により，割り込み要求名は適切でない場合があるため，メッセージを出力します。
- __rtos_interrupt は#pragma rtos_interrupt に変換します。
- 「割り込み要求名」は，アドレスを示すマクロとして「vect=アドレス」に変換します。マクロの値は iodefine.h で定義されます。
- CA78K0 の書式では関数名を省略可能であり，RTOS 割り込みハンドラが使用する ret_int, ret_wup をユーザが定義できないようにする機能があります。同じ機能は CC-RL には存在しないため，割り込み要求名と関数名が省略された場合はメッセージを出力し，#pragma 指令をコメントアウトします。
- 「スタック切り替え指定」は CC-RL に存在しないため，メッセージを出力し削除します。
- __rtos_interrupt キーワードを用いた割り込み関数の宣言や定義にマクロや typedef を使用した場合，関数名の判断を誤る可能性があります。予めマクロや typedef を展開してから変換してください。
- 同一関数に対する pragma 指令とキーワードによる割り込み関数の記述がある場合，どちらも#pragma 指令に変換することで，変換後に#pragma 指令が重複して CC-RL ではコンパイル・エラーとなります。この場合は，重複する記述を削除してください。
- __rtos_interrupt キーワードを指定した関数宣言の引数を省略した場合，CC-RL ではコンパイル・エラーとなります。引数の型として void 型を記述してください。

［例］

パターン 1	変換前	#pragma rtos_interrupt INTP0 func void func(void) { }
	変換後	#pragma rtos_interrupt func (vect=INTP0) void func(void) { }
パターン 2	変換前	#pragma rtos_interrupt INTP0 func sp=buff+10 void func(void) { }
	変換後	#pragma rtos_interrupt func (vect=INTP0) void func(void) { }
パターン 3	変換前	__rtos_interrupt void func(void) { }
	変換後	#pragma rtos_interrupt func void func(void) { }
パターン 4	変換前	#pragma rtos_interrupt
	変換後	// #pragma rtos_interrupt
パターン 5	変換前	__rtos_interrupt void func1(void), func2(void);

	変換後	#pragma rtos_interrupt func1
		void func1(void);
		#pragma rtos_interrupt func2
		void func2(void);

パターン 6	変換前	#pragma rtos_interrupt INTP0 func __rtos_interrupt func(void);
	変換後	#pragma rtos_interrupt func(vect=INTP0) void func(void); #pragma rtos_interrupt func void func(void);
	対応	CC-RL では#pragma 指令の重複でエラーとなります。キーワードから変換した#pragma 指令を削除してください。
パターン 7	変換前	typedef void func_t(void); __rtos_interrupt func_t f1;
	変換後	typedef void func_t(void); __rtos_interrupt func_t f1;
	対応	CC-RL ではコンパイル・エラーとなります。予め typedef やマクロを展開してください。

3.8 RTOS 用タスク関数

RTOS 用タスク関数の書式は CA78K0 と CC-RL とでほぼ同じです。

CA78K0 の書式は以下の通りです。

```
#pragma rtos_task [タスク関数名]
```

CC-RL の書式は以下の通りです。

```
#pragma rtos_task [(]タスク関数名[, … ][)]
関数宣言
```

- CA78K0 の書式ではタスク関数名を省略可能であり，RTOS 用タスク関数が使用する ext_tsk をユーザが定義できないようにする機能があります。同じ機能は CC-RL には存在しないため，タスク関数名が省略された場合はメッセージを出力し，#pragma 指令をコメントアウトします。

［例］

パターン 1	変換前	#pragma rtos_task task1
	変換後	#pragma rtos_task task1
パターン 2	変換前	#pragma rtos_task
	変換後	// #pragma rtos_task

3.9 絶対番地配置指定

CA78K0 では__directmap キーワードを用いて配置先を指定しますが，CC-RL では#pragma address を変数宣言の直前に記述します。

CA78K0 の書式は以下の通りです。

```
__directmap [__sreg] [static] 型名 変数名 = 配置アドレス;
```

CC-RL の書式は以下の通りです。

```
#pragma address 変数名 = 配置アドレス
変数宣言
```

- 78K0 と RL78 ではメモリマップが異なるため，メッセージを出力します。
- __directmap キーワードを削除し，直前に#pragma address を追加します。変数宣言から配置アドレス指定を削除し，#pragma address の配置アドレス指定に移動します。
- __directmap キーワードを用いた記述にマクロや関数ポインタを使用した場合，関数名の判断を誤る可能性があります。予めマクロを展開してから変換してください。関数ポインタの配置指定は，手動で修正してください。
- __directmap で異なる変数を同一アドレスに割り当てている場合，変換後 CC-RL にてコンパイル・エラーとなります。CcnvCA78K0 は異なる変数を同一アドレスに割り当てているかどうかをチェックしませんのでご注意ください。

［例］

パターン 1	変換前	__directmap int i = 0xfe00;
	変換後	#pragma address i=0xfe00 int i;
パターン 2	変換前	__directmap int* i = 0xfe00;
	変換後	#pragma address i=0xfe00 int* i;
パターン 3	変換前	__directmap int i = 0xfe00, j=0xfe10;
	変換後	#pragma address i=0xfe00 #pragma address j=0xfe10 int i,j;
パターン 4	変換前	__directmap struct x { char a ; char b ; } xx = { 0xfe30 } ;
	変換後	#pragma address xx=0xfe30 struct x { char a ; char b ; } xx;
パターン 5	変換前	#define MY_MACRO1 (int i = 0xfe00) __directmap MY_MACRO1;
	変換後	#define MY_MACRO1 (int i = 0xfe00) __directmap MY_MACRO1;
	対応	マクロを展開した状態に修正してから変換してください。
パターン 6	変換前	__directmap void (*fp[])(void) = 0x1234;
	変換後	#pragma address void=0x1234 void (*fp[])(void);
	対応	手動で CC-RL の#pragma address を記述してください。

3.10 組み込み関数

CA78K0 では#pragma 指令により組み込み関数が有効となっていましたが，CC-RL では組み込み関数が常に使用できます。CA78K0 の組み込み関数に対応する CC-RL の組み込み関数が存在する場合，CcnvCA78K0 は C ソース中の#pragma 指令を削除し，組み込み関数の呼び出し箇所を変更します。

- #pragma 指令がない場合は，組み込み関数が有効でないと判断し変換しません。
- CC-RL ではサポートしていない組み込み関数は，#pragma 指令を削除し，メッセージを出力します。組み込み関数の呼び出し箇所は変換しません。

表 3.4 組み込み関数の変換

CA78K0 組み込み関数	変換後	備考
#pragma DI DI	削除します __DI	
#pragma EI EI	削除します __EI	
#pragma HALT HALT	削除します __halt	
#pragma STOP STOP	削除します __stop	
#pragma BRK BRK	削除します __brk	
#pragma NOP NOP	削除します __nop	
#pragma rot rolb rorb rolw rorw	削除します __rolb __rorb __rolw __rorw	
#pragma mul mulu	削除します __mulu	
#pragma div divuw moduw	削除します __divui __remui	
#pragma bcd adbcdb, sbbcdb, adbcdbe, sbbcdbe, adbcdw, sbbcdw, adbcdwe, sbbcdwe, bcdtob, btobcde, bcdtow, wtobcd, btobcd	削除します いずれも変換しません	CC-RL ではサポートしていません。

CA78K0 組み込み関数	変換後	備考
#pragma opc __OPC	削除します 変換しません	CC-RL ではサポートしていません。
#pragma realregister __geta, __seta, __getax, __setax, __getcy, __setcy, __set1cy, __clr1cy, __not1cy, __inca, __deca, __rora, __rorca, __rola, __rolca, __shla, __shra, __ashra, __nega, __coma, __absa	削除します いずれも変換しません	CC-RL ではサポートしていません。
#pragma hromcall __hromcall __hromcalla __setsp	削除します いずれも変換しません	CC-RL ではサポートしていません。
#pragma access peekb, peekw, pokeb, pokew	削除します いずれも変換しません	CC-RL ではサポートしていません。

3.11 その他の#pragma 指令

その他の#pragma 指令に対する変換仕様を示します。

表 3.5 その他の#pragma 指令の変換

CA78K0 #pragma 指令	変換後	備考
#pragma sfr	#include “iodefine.h”	iodefine.h は統合開発環境が提供します。 デバイス変更により SFR のアクセスは見直す必要があります。メッセージを出力します。
#pragma name	削除します	CC-RL ではサポートしていません。
#pragma ext_table	削除します	CC-RL ではサポートしていません。
#pragma ext_func	削除します	CC-RL ではサポートしていません。
#pragma inline	削除します	CC-RL ではサポートしていません。 CC-RL の#pragma inline は別機能となります。
#pragma PC	削除します	CC-RL ではサポートしていません。

3.12 標準ライブラリ関数

CA78K0 の標準ライブラリ関数のうち，va_starttop, va_start_banked, va_starttop_banked の関数呼び出しを，CC-RL の標準ライブラリ関数に変換します。通常の標準ライブラリ関数については，同一関数が使用できるため，変換しません。

- CA78K0 用標準ライブラリのヘッダ・ファイルを CcnvCA78K0 で変換して，CC-RL で用いないでください。CC-RL 用標準ライブラリのヘッダ・ファイルを使用してください。
- 関数名の変換は文字列置換であるため，同名のマクロ名，変数名，タグ名等が存在する場合は，それらも変換します。
- CC-RL の標準ライブラリは引数や戻り値の型に__near/__far が指定されているため，変換後に引数や戻り値の型が合わなくなる場合があります。CC-RL のユーザーズ・マニュアルをご確認の上，手動で修正してください。

表 3.6 標準ライブラリ関数の変換

CA78K0 関数名	変換後	備考
toup _toupper	変換しません	CC-RL ではユーザ関数として扱います。 toupper 関数をご使用ください。
tolow _tolower	変換しません	CC-RL ではユーザ関数として扱います。 tolower 関数をご使用ください。
va_starttop va_start_banked va_starttop_banked	va_start	
atexit	変換しません	CC-RL は atexit をサポートしていません。 CC-RL ではユーザ関数として扱います。
brk	変換しません	CC-RL ではユーザ関数として扱います。
sbrk	変換しません	CC-RL ではユーザ関数として扱います。
itoa	変換しません	CC-RL ではユーザ関数として扱います。
ltoa	変換しません	CC-RL ではユーザ関数として扱います。
ultoa	変換しません	CC-RL ではユーザ関数として扱います。
strbrk	変換しません	CC-RL ではユーザ関数として扱います。
strsbrk	変換しません	CC-RL ではユーザ関数として扱います。
stritoa	変換しません	CC-RL ではユーザ関数として扱います。
strltoa	変換しません	CC-RL ではユーザ関数として扱います。
strultoa	変換しません	CC-RL ではユーザ関数として扱います。
strcoll	変換しません	CC-RL は strcoll をサポートしていません。 CC-RL ではユーザ関数として扱います。
strxfrm	変換しません	CC-RL は strxfrm をサポートしていません。 CC-RL ではユーザ関数として扱います。
matherr	変換しません	CC-RL ではユーザ関数として扱います。
__assertfail	変換しません	CC-RL ではユーザ関数として扱います。 assert マクロはそのまま CC-RL で使用できます。

3.13　CC-RL オプション-convert_cc との変換仕様差分

CC-RL には，コンパイル時に CA78K0 のソースコードをコンパイルするオプションは存在しません。CC-RL にて CA78K0R のソースコードをコンパイルためのオプション「-convert_cc=ca78k0r」指定時と，CcnvCA78K0 による変換とで動作が異なる拡張機能について，差分を示します。

表 3.7 CC-RL オプション-convert_cc=ca78k0r と異なる動作

CA78K0 の拡張機能	オプション-convert_cc=ca78k0r 指定時の動作	CcnvCA78K0 の変換
__boolean	-ansi オプション指定なしの場合は _Bool，-ansi オプション指定ありの場合は char として扱います。	常に_Bool に変換します。 CC-RL の-ansi オプション指定時は_Bool が使用できませんので，手動で型を変更してください。
__callf __leaf __pascal __flash __flashf __temp __mxcall	コンパイル・エラーとなります。	削除します。
callf noauto norec	コンパイル・エラーとなります。	オプション-A を指定しない場合に削除します。
__interrupt __interrupt_brk __rtos_interrupt	同名関数に対する#pragma 指令が既に存在する場合は，キーワードを無視します。	関数宣言の前に#pragma 指令を追加します。同名関数に対する#pragma 指令が既に存在する場合は，変換後に#pragma 指令が重複して CC-RL ではコンパイル・エラーとなります。その場合は，キーワードを変換した#pragma 指令を削除してください。
__asm( )	通常の関数呼び出しと認識します。 手動で inline_asm 関数に修正する必要があります。 デバイス変更によりアセンブラ記述は見直してください。	各__asm( )に対して#pragma inline_asm と関数定義を出力します。 __asm( )呼び出しは，新しく生成した関数呼び出しに変換します。 デバイス変更によりアセンブラ記述は見直してください。
va_start_banked va_starttop_banked	通常の関数呼び出しと認識します。	va_start に変換します。
#pragma sfr	CC-RL のオプション-preinclude で iodefine.h をインクルードしてください。 デバイス変更により SFR のアクセスは見直してください。	ファイルの先頭に#include “iodefine.h”を挿入します。 デバイス変更により SFR のアクセスは見直してください。

CA78K0 の拡張機能	オプション-convert_cc=ca78k0r 指定時の動作	CcnvCA78K0 の変換
#asm ～ #endasm	構文エラーとなります。 手動で inline_asm 関数に修正する必要があります。 デバイス変更によりアセンブラ記述は見直してください。	各#asm～#endasm に対して#pragma inline_asm と関数定義を出力します。 #asm～#endasm を，新しく生成した関数呼び出しに変換します。 デバイス変更によりアセンブラ記述は見直してください。
#pragma interrupt #pragma vect #pragma rtos_interrupt	割り込み要求名がある場合，CC-RL のオプション-preinclude で iodefine.h をインクルードしてください。 デバイス変更により割り込み要求名は見直してください。	割り込み要求名がある場合，ファイルの先頭に#include “iodefine.h”を挿入します。 デバイス変更により割り込み要求名は見直してください。

4. メッセージ

この章では，CC-RL が出力するメッセージについて説明します。

4.1 出力形式

メッセージの出力形式は，以下の通りです。

・ファイル名と行番号を含む場合

- メッセージ番号の種別がインフォメーションの場合

```
ファイル名(行番号):メッセージ番号:[情報種別]メッセージ
```

情報種別は，変更・追加・削除・情報のいずれかとなります

- メッセージ番号の種別がインフォメーション以外の場合

```
ファイル名(行番号):メッセージ番号:メッセージ
```

・ファイル名と行番号を含まない場合

```
メッセージ番号:メッセージ
```

メッセージ番号は，1 文字の英字(メッセージ種別)，0592，3 桁の数字が連続した文字列として出力されます。

4.2 メッセージ種別

メッセージ種別は，以下のように分類されています。

表 4.1 メッセージ種別

種別	先頭の文字	説明
内部エラー	C	処理を中止します。 変換後 C ソースを出力しません。
エラー	E	処理を中止します。 変換後 C ソースを出力しません。
ワーニング	W	処理を継続します。 変換後 C ソースを出力します。
インフォメーション	M	処理を継続します。 変換後 C ソースを出力します。

4.3 情報種別

メッセージ番号の種別がインフォメーションの場合，情報種別は以下のように分類されています。

表 4.2 情報種別

情報種別	説明
変更	CC-RL 用の C ソースにするため，内容を変更しました。
追加	CC-RL 用の C ソースにするため，内容を追加しました。
削除	CC-RL では記述しないため，内容を削除しました。
情報	CA78K0 と CC-RL との仕様の違いにより，十分な変換ではない場合があります。 個別に確認してください。

4.4 メッセージ一覧

CcnvCA78K0 が出力するメッセージは，以下の通りです。

4.4.1 内部エラー

表 4.3 内部エラー

番号	メッセージ	説明
C0592*nnn*	内部エラーが発生しました	特約店，または当社までご連絡ください。

nnn は 3 桁の 10 進数字です。

4.4.2 エラー

表 4.4 エラー

番号	メッセージ	説明
E0592001	複数の入力ファイルが指定されています。	入力ファイルは 1 つしか指定できません。 入力ファイルを複数指定したい場合は，リスト・ファイルを使用してください。
E0592002	*option* オプションに引数は指定できません。	引数を指定できないオプションに引数を指定しています。
E0592003	*option* オプションに引数を指定してください。	引数が必要なオプションに引数を指定していません。
E0592004	*option* オプションが複数回指定されています。	オプションは同時に 1 つしか指定できません。
E0592005	出力ファイルが指定されていません。	入力ファイルに対応する出力ファイルが指定されていません。
E0592006	入力ファイル *file* の読込みに失敗しました。	フォルダ名またはファイル名に誤りがある可能性があります。リスト・ファイルに次のファイルが指定されている倍は，次の変換に移ります。
E0592007	変換結果ファイル *file* の書込みに失敗しました。	フォルダ名に誤りがある可能性があります。
E0592008	出力ファイル *file* の書込みに失敗しました。	フォルダ名に誤りがある可能性があります。
E0592009	リスト・ファイル *file* の読込みに失敗しました。	フォルダ名に誤りがある可能性があります。
E0592010	リスト・ファイル *file* の構文が認識できません。	リスト・ファイルの記述が正しくありません。
E0592011	ファイル名が重複しています。	入力ファイル，出力ファイル，変換結果出力ファイルのファイル名が重複しています。
E0592012	ファイル名が不正です。	コマンドライン指定時の入力ファイル名，もしくは，リスト・ファイル指定時の入出力ファイル名のいずれかが 260 文字を超えています。
E0592013	*option* オプションに指定された引数が不正です。	引数の指定が不正です。 または，指定したファイル名が 260 文字を超えています。

番号	メッセージ	説明
E0592101	*string* の書式が不正です。	CA78K0 で許可されていない文法があるため，変換できません。入力ファイルを修正してください。
E0592102	アセンブラ埋め込みインライン関数をこれ以上追加できません。	アセンブラ埋め込みインライン関数の作成上限数を超えました。入力ファイルを修正してください。
E0592103	テンポラリ・ファイルの削除に失敗しました。	テンポラリ・ファイルの削除に失敗しました。テンポラリ・ファイルを削除してください。

4.4.3 ワーニング

表 4.5 ワーニング

番号	メッセージ	説明
W0592051	"-l"オプションが指定されたので，入力ファイルの指定は無視されます。	リスト・ファイル指定時は，コマンドラインで入力ファイルを同時に指定できません。-l オプションで指定されたリスト・ファイルの変換が実行され，入力ファイルを無視します。
W0592052	"-c"の指定が，リスト・ファイルとコマンドラインで異なります。コマンドラインの指定は無視されます(*file*)。	リスト・ファイル中で指定された入力ファイル *file* に対応する-c オプションの指定がリスト・ファイルとコマンドラインで異なります。リスト・ファイルの指定で変換します。
W0592053	不正な *option* オプションが指定されています。	不正なオプションが指定されています。オプションを無視します。
W0592054	*option* オプションに指定された引数(*argument*)が不正です。	オプションに指定された引数が不正です。 -c オプションの引数が不正な場合，デフォルト時の指定で処理します。
W0592055	入力ファイル名が指定されていませんでした。この変換を無視します。	-l オプションで指定されたリスト・ファイルに入力ファイルの指定がない箇所があります。
W0592151	*string* を CC-RL の形式に変更できませんでした。	*string* を CC-RL の形式に変更できませんでした。入力ファイルを修正してください。

4.4.4 インフォメーション

表 4.6 インフォメーション

番号	情報種別	メッセージ	説明
M0592111	変更	*string1* を *string2* に変換しました。	字句を変換しました。
M0592112	変更	I/O レジスタのビット指定アクセスをマクロ呼び出しに変換しました。	SFR のビット・アクセス方法が CA78K0 と CC-RL とで異なるため，マクロを用いてアクセスするよう変更しました。
M0592113	変更	*string* を CC-RL の形式に変更しました。	記述形式が CA78K0 と CC-RL とで異なるため, CC-RL の記述形式に変更しました。
M0592121	追加	I/O レジスタのビット・アクセス用マクロを生成しました。	SFR のビット・アクセス方法が CA78K0 と CC-RL とで異なるため，マクロを用いてアクセスするよう変更しました。
M0592122	追加	#pragma interrupt NO_VECT を生成しました。	ベクタ・テーブル指定がない#pragma interrupt を生成しました。
M0592123	追加	*string* を生成しました。	CC-RL の形式に合わせて, 記述を追加しました。
M0592124	追加	アセンブラ埋め込みインライン関数 *string* を生成しました。	アセンブラ埋め込みインライン関数を生成しました。
M0592125	追加	#pragma rtos_interrupt NO_VECT を生成しました。	ベクタ・テーブル指定がない#pragma rtos_interrupt を生成しました。
M0592131	削除	*string* を削除しました。	CC-RL にない記述形式です。記述を削除しました。
M0592142	情報	対応するセクションが無いため，変換を行いません(*section*)。	CC-RL に対応するセクションが存在しないため，変換できませんでした。
M0592143	情報	「AT *開始アドレス*」を削除します。	CC-RL では#pragma section によるアドレス指定ができないため，削除します。
M0592144	情報	対応するマクロが存在しないので，変換しません(*MACRO*)。	CC-RL に対応するマクロが存在しないため，変換できませんでした。
M0592145	情報	アセンブラ命令中にラベルの使用が検出されました。ラベルの変換はできませんので，適切な内容に修正してください。	CC-RL のアセンブリ記述関数内にはローカル・ラベルしか書けません。ラベルを適切な内容に修正してください。
M0592146	情報	78K0 に依存した言語仕様です。	78K0 から RL78 へのデバイス変更の際に見直しが必要です。手動で修正してください。

5. 注意事項

下記の項目に該当する場合, 変換後のC ソースはCC-RL で正しくコンパイルできないことがあります。

表 5.1 注意事項

項番	項目	CcnvCA78K0 の動作	変換結果に対するCC-RL の動作	参照先
1	ネスト構造のコメント文がある場合	正常に変換できない場合があります。	コメントの範囲が不正となります。	3 章
2	##演算子等の使用により，キーワードを検出できない場合	メッセージを出力せず，変換しません。	エラーE0520065 等になります。	3 章
3	#pragma section で，セクション名に「?」が含まれる場合	メッセージを出力せず，変換しません。	エラーE0520014 になります。	3.4 節
4	#pragma section のセクション名に, CC-RL には存在しないセクション名を指定している場合	メッセージを出力せず，変換しません。	ワーニングW0523037 を出力し，#pragma 指令を無視します。セクションの配置に失敗し，期待した動作とならない可能性があります。	3.4 節
5	__asm("文字列")中の文字列に¥n や¥t を使用している場合	制御文字をそのまま出力します。	エラーE0550249 になります。	3.5 節
6	#asm～#endasm 内のアセンブリ記述のコメント(“;”以降の記述)内に「/*」が含まれている場合	アセンブリ記述のコメントをそのまま出力します。	アセンブリ記述のコメント(“;”)よりもC 記述のコメント(“/*”)を優先し，コメントの範囲が不正となります。	3.5 節
7	__asm( )または#asm～#endasm のアセンブリ記述にラベルが含まれる場合	メッセージを出力します。	エラーE0550213 になります。	3.5 節 CC-RL ユーザーズ・マニュアル #pragma inline_asm の[制限]

項番	項目	CcnvCA78K0 の動作	変換結果に対する CC-RL の動作	参照先
8	__asm( )または#asm～#endasm のアセンブリ記述に 78K0 固有の命令または RL78 とは動作が異なる命令を使用している場合	メッセージを出力せず，#pragma inline_asm 関数内にそのまま出力します。	78K0 固有の命令を使用している場合は，アセンブル・エラーになります。 RL78 とは動作が異なる命令を使用している場合は，期待する動作にならない可能性があります。	3.5 節
9	ファイル内に，同一関数に対する#pragma interrupt とキーワード__interrupt の記述がある場合	それぞれ，#pragma interrupt に変換します。	#pragma 指令が重複し，エラー E0523006 になります。	3.6 節
10	__interrupt, __interrupt_brk, __rtos_interrupt の関数宣言の引数を省略した場合	#pragma 指令を出力し，関数宣言はそのまま出力します。	void 型の指定がないため，エラー E0523008 になります。	3.6 節 3.7 節
11	型名，関数名，変数名等を取得する箇所において，マクロや typedef などを使用していた場合	メッセージを出力し，変換せずに出力します。	エラーE0520020, E0520065 等になります。	3.6 節 3.7 節 3.9 節
12	ファイル内に，同一関数に対する#pragma rtos_interrupt とキーワード __rtos_interrupt の記述がある場合	それぞれ，#pragma rtos_interrupt に変換します。	#pragma 指令が重複し，エラー E0523006 になります。	3.7 節
13	__directmap で，異なる変数に対して同一アドレスを指定している場合	メッセージを出力しません。	エラーE0541854 になります。	3.9 節 CC-RL ユーザーズ・マニュアル #pragma address の[制限]

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2016.04.01	ー	初版発行

CcnvCA78K0 ユーザーズマニュアル

発行年月日　2016年4月1日　　Rev.1.00

発行　　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）

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：http://japan.renesas.com/contact/

Colophon 3.0

CcnvCA78K0

ルネサス エレクトロニクス株式会社

R20UT3684JJ0100