CS+ RH850 コンパイラ CC-RH V1.07.00 リリースノート

R20UT4208JJ0100
Rev.1.00
2017.12.20

この度は、統合開発環境 CS+をご使用いただきまして誠にありがとうございます。

この添付資料では、本製品をお使いいただく上での制限事項および注意事項等を記載しております。

ご使用の前に、必ずお読みくださいますようお願い申し上げます。

目次

第1章　対象デバイスについて

CC-RH がサポートする対象デバイスに関しては WEB サイトに掲載しています。

こちらをご覧ください。

CS+製品ページ：

https://www.renesas.com/cs+

第2章　ユーザーズ･マニュアルについて

本製品に関連したユーザーズ･マニュアルは次のようになります。本文書と合わせてお読みください。

マニュアル名	資料番号
CC-RH コンパイラ ユーザーズマニュアル	R20UT3516JJ0104
CS+ 統合開発環境 ユーザーズマニュアル CC-RH ビルド・ツール操作編	R20UT3283JJ0105

第3章　アンインストール時の選択キーワード

本製品をアンインストールする場合は、2 つの方法があります。

・統合アンインストーラを使用する(CS+自体をアンインストールする)

・個別にアンインストールする(本製品のみをアンインストールする)

個別にアンインストールを行う場合、コントロールパネルの

・「プログラムと機能」

から「CS+ CC-RH V1.07.00」を選択してください。

第4章　変更点

本章では、CC-RH V1.06.00 から V1.07.00 への主な変更点について説明します。
なお、professional 版のライセンス登録時のみ使用できる機能は【professional】と明記します。

4.1　C99規格

言語仕様として C99 規格に準拠し、コンパイル・オプション**-lang** と**-strict_std** を追加しました。

なお今版では、C99 規格のうち可変長配列型・複素数型・一部の標準ライブラリ関数は未サポートです。

-lang=*{c|c99}*

-lang=*c* オプション指定時、C90 規格に準拠します。
また、-lang=*c99* オプション指定時、C99 規格に準拠します。

-strict_std

-lang オプションで指定した言語規格（C90 あるいは C99）に厳密に合わせて処理し、規格に反する記述に対してエラーや警告を出力します。

V1.06.00 以前には C90 規格に厳密に合わせて処理する-Xansi オプションがありましたが、V1.07.00 からは-strict_std オプションに変更します。なお、V1.07.00 以降で-Xansi オプションを指定した場合、-strict_std オプションに自動的に変換して入力を受け付けます。

4.2 MISRA-C:2012ルールによるチェック機能の拡充【professional】

MISRA-C:2012 ルールによりソース・チェックを行う-Xmisra2012 オプションの引数に、下記の C99 規格用のルール番号を指定できるようにしました。

【必須ルール】 **17.6,**

【必要ルール】 **8.14 9.4, 9.5 13.1 18.7 21.11**

【推奨ルール】 **21.12**

各リビジョンでチェック可能な MISRA-C:2012 ルール数は下記の通りです。

ルール分類（ルール数）	*V1.03.00*	*V1.04.00*	*V1.05.00*	*V1.06.00*	***V1.07.00***
必須ルール（16）	3	3	4	6	**7**
必要ルール（108）	31	58	76	80	**86**
推奨ルール（32）	7	21	23	25	**26**
合計ルール（156）	41	82	103	111	**119**

4.3 不正な間接関数呼び出し検出機能【professional】

不正なアドレスへの間接関数呼び出しを検出する機能を追加しました。

下記の手順で間接関数呼び出しの分岐先アドレスをチェックし、問題を検出した場合にエラー関数を呼び出します。

1. コンパイル時に、間接関数呼び出しされる可能性のある関数をＣ ソース・プログラムから自動で抽出し、リンク時にその情報を統合して関数リストを実行形式ファイル内に生成します。

2. C ソース・プログラムを解析して間接関数呼び出しが行われる直前に、チェック関数”__control_flow_integrity”を呼び出す処理を挿入します。このチェック関数には、当該間接関数呼び出しによる分岐先アドレスが引数として渡されます。

3. 実行時にチェック関数内で、引数の分岐先アドレスが関数リストに含まれるかをチェックします。含まれていない場合は不正な間接関数呼び出しと判断してエラー関数”__control_flow_chk_fail”を呼び出します。

下記の C ソースを例に説明します。

```
1:  extern void func1(void);
2:  extern void func2(void);
3:
4:  void (*fp)(void) = &func1;
5:
6:  void main(void) {
7:      (*fp)();            // 関数func1の間接呼び出し
8:      func2();            // 関数func2の直接呼出し
9:  }
```

4 行目で関数 func1 のアドレスが取得されているため、間接呼び出しされる可能性のある関数と判断して関数リストに func1 を追加します。

7 行目で関数ポインタ fp を使用して間接呼び出ししているため、この呼び出しの直前で関数ポインタ fp の値を取得し、取得した値を引数としてチェック関数”__control_flow_integrity”を呼び出すコードを生成します。チェック関数内では、引数で指定した値（正常に実行している場合は func1 のアドレス）が関数リストに含まれるかをチェックし、次のように動作します。

- 含まれている場合 ⇒ C ソース・プログラムの処理を続行します。
- 含まれていない場合 ⇒ エラー関数”__control_flow_chk_fail”を呼び出します。

上記のように、不正な間接関数呼び出しを検出することが可能となります。

8 行目は関数 func2 の直接呼出しのため、検出機能の対象外です。

本機能を有効にするには、下記のオプションを指定してください。

【コンパイル・オプション】

-control_flow_integrity

不正な間接関数呼び出しを検出するコードを生成します。

【リンク・オプション】

-cfi

不正な間接関数呼び出し検出時に用いる関数リストを生成します。

また、本機能に関連して下記のリンク・オプションも追加しました。

- **-cfi_add_func**
  引数に指定した関数のシンボルまたはアドレスを関数リストに追加します。
- **-cfi_ignore_module**
  引数に指定したファイルに含まれる関数のアドレスを関数リストへ追加しません。
- **-show=*cfi***
  -list オプションを指定して出力するリスト・ファイルに関数リストの内容を出力します。

4.4 PIC/PID機能

関数・定数・変数をリンク時とは異なる任意のアドレスに配置できる PIC/PID 機能を追加しました。
CC-RH では、任意のアドレスに

- 関数を配置して実行できる機能を **PIC**（**P**osition **I**ndependent **C**ode）機能
- 定数を配置して参照できる機能を **PIROD**（**P**osition **I**ndependent **R**ead **O**nly **D**ata）機能
- 変数を配置して参照できる機能を **PID**（**P**osition **I**ndependent **D**ata）機能

と呼びます。

4.4.1 セクション

PIC/PID 機能をサポートするために、下記のセクションを追加しました。

対象	*セクション名*	*アクセス方法*
関数	**.pctext**	PC または__pc_data シンボルからの32 ビット長相対
定数	**.pcconst16**	__pc_data シンボルからの16ビット長相対
定数	**.pcconst23**	__pc_data シンボルからの23ビット長相対
定数	**.pcconst32**	__pc_data シンボルからの32ビット長相対
初期値あり変数	**.sdata32**	r4（GP）からの32 ビット長相対
初期値なし変数	**.sbss32**	
初期値あり変数	**.edata32**	r30（EP）からの32 ビット長相対
初期値なし変数	**.ebss32**	

上記のセクションの追加に関連して、下記を追加しました。

- #pragma section に指定可能な属性指定文字：
  pctext, pcconst16, pcconst23, pcconst32, gp_disp32, ep_disp32
- -Xsection オプションの引数に指定可能なセクション属性：
  pcconst16, pcconst23
- .cseg, .dseg, .section 疑似命令のオペランドに指定可能な再配置属性：
  PCTEXT, PCCONST16, PCCONST23, PCCONST32,
  SDATA32, SBSS32, EDATA32, EBSS32

4.4.2 コンパイル・オプション

PIC/PID 機能を有効にするために、下記のコンパイル・オプションを追加しました。

【PIC 機能を有効にする場合】

-pic

関数のデフォルト・セクションが.text ⇒ .pctext セクションになります。

.pctext セクションに配置した関数への呼び出しやアドレス参照を PC 相対で行うことにより任意の位置での実行を実現します。

【PIROD 機能を有効にする場合】

-pirod

定数のデフォルト・セクションが.const ⇒ .pcconst32 セクションになります。

.pcconst32 セクションに配置した定数への参照を PC 相対で行うことにより任意の位置への配置と参照を実現します。

【PID 機能を有効にする場合】

-pid

変数のデフォルト・セクションが.data/.bss ⇒ .sdata32/.sbss32 セクションになります。

.sdata32/.sbss32 セクションに配置した変数への参照を GP 相対で行うことにより任意の位置への配置と参照を実現します。

4.4.3 アセンブル・オプション

PIC/PID 機能を有効にするために、下記のアセンブル・オプションを追加しました。

【PIC 機能を有効にする場合】

-pic

.cseg, .section 疑似命令のオペランドに指定可能な再配置属性が次のように変更されます。指定可能でない再配置属性を指定するとエラーになります。

- オプション未指定時：TEXT
- オプション指定時：PCTEXT

【PIROD 機能を有効にする場合】

-pirod

.cseg, .section 疑似命令のオペランドに指定可能な再配置属性が次のように変更されます。指定可能でない再配置属性を指定するとエラーになります。

- オプション未指定時：CONST，ZCONST，ZCONST23
- オプション指定時：PCCONST16，PCCONST23，PCCONST32

【PID 機能を有効にする場合】

-pid

.cseg, .section 疑似命令のオペランドに指定可能な再配置属性が次のように変更されます。指定可能でない再配置属性を指定するとエラーになります。

- オプション未指定時：DATA，ZDATA，ZDATA23，BSS，ZBSS，ZBSS23
- オプション指定時：SDATA32，SBSS32，EDATA32，EBSS32

4.4.4 -r4オプション

コンパイル時に r4 レジスタ（GP）を使用しないコードを生成する**-r4** オプションを追加しました。

-r4={fix|none}

-r4=*fix* オプション指定時は、r4 レジスタの値をプロジェクト全体で固定します。PID 機能等で GP 相対セクションを使用する場合に指定してください。

-r4=*none* を指定時は、PID と PID でないプログラムの両方に使用できるオブジェクトを生成します。

4.5 最適化強化

主に以下のような最適化を実装することにより、生成コードの性能を改善しました。

4.5.1 ビット演算の改善

ビット幅の狭いデータに対するビット演算を改善しました。

<ソースコード例>

```
void func(unsigned char *t, unsigned char i, unsigned char j, unsigned char v) {
        unsigned char *p = &t[i & 0xff];
        unsigned char m = j >> (i & 0xf);
        *p = v ? m : ~m;
}
```

この例の場合、*p への代入値に対するビットマスク演算を除去します。またその結果、条件演算の一方にあるビット反転演算に対して、論理演算命令 not を使用可能になります。

<V1.06.00 の生成コード>

```
_func:
        .stack _func = 0
        andi 0x0000000F, r7, r2
        shr r2, r8
        andi 0x000000FF, r8, r2 ; ビットマスク演算
        add r7, r6
        cmp 0x00000000, r9
        bnz9 .BB.LABEL.1_2
.BB.LABEL.1_1:  ; bb20
        movea 0xFFFFFF00, r0, r2
        or r2, r8
        xori 0x000000FF, r8, r2
.BB.LABEL.1_2:  ; bb23
        st.b r2, 0x00000000[r6]
        jmp [r31]
```

<V1.07.00 の生成コード>

```
_func:
        .stack _func = 0
        andi 0x0000000F, r7, r2
        shr r2, r8
        add r7, r6
        cmp 0x00000000, r9
        bnz9 .BB.LABEL.1_2
.BB.LABEL.1_1:  ; bb20
        not r8, r8  ; 論理演算命令
.BB.LABEL.1_2:  ; bb23
        st.b r8, 0x00000000[r6]
        jmp [r31]
```

4.5.2 別名解析

別名解析の最適化を改善しました。別名解析は V1.06.00 で実装し、-Xalias=ansi オプション指定時に有効になる最適化です。

V1.06.00 では-Xmerge_files オプション指定時は無効でしたが、V1.07.00 では-Xmerge_files オプション指定時にも有効になるように改善しました

別名解析の最適化が有効になったときに現れる効果は V1.06.00 と同じです.

<ソースコード例>

```
struct tag1 {
    char member1;
    int   member2;
    long long member3;
} StructArray[2];

struct tag2 {
    short index0;
    short index1;
    short index2;
};

void func(struct tag2 *p) {
    StructArray[p->index1].member1 = 1;
    StructArray[p->index1].member2 = 2;
    StructArray[p->index1].member3 = 3;
}
```

この例の場合、別名解析の無効時には StructArray[p->index1]のアドレス計算を 3 回実施していますが、有効時には 1 回のみ実施します。

<無効時>

```
_func:
        .stack _func = 0
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        mov #_StructArray, r5
        add r5, r2
        mov 0x00000001, r7
        st.b r7, 0x00000000[r2]
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        add r5, r2
        mov 0x00000002, r7
        st.w r7, 0x00000004[r2]
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        add r2, r5
        mov 0x00000003, r2
        st.w r2, 0x00000008[r5]
        st.w r0, 0x0000000C[r5]
        jmp [r31]
```

<有効時>

```
_func:
        .stack _func = 0
        ld.h 0x00000002[r6], r2
        shl 0x00000004, r2
        mov #_StructArray, r5
        add r2, r5
        mov 0x00000001, r2
        st.b r2, 0x00000000[r5]
        mov 0x00000002, r2
        st.w r2, 0x00000004[r5]
        mov 0x00000003, r2
        st.w r2, 0x00000008[r5]
        st.w r0, 0x0000000C[r5]
        jmp [r31]
```

4.6 メモリ使用量の上限拡張

CC-RH が使用できるホスト PC 上のメモリ量を拡張しました。

- 32bit/64bit OS とも 2G byte 【V1.06.00 以前】
- 32bit OS の場合は **3G byte**、64bit OS の場合は **4G byte** 【V1.07.00 以降】

4.7 即値を用いた自動変数の初期化

構造体型や配列型の自動変数の初期化を即値で行うコンパイル・オプション**-Oinline_init** を追加しました。この機能により、プログラムの実行が高速になる場合があります。

<ソースコード例>

```
void main() {
  int array[4] = {1,2,3,4};
}
```

自動変数を初期化する際、初期化子を定義したテーブル（V1.06.00 の生成コード例では.STR.1）を参照して初期化していましたが、-Oinline_init オプションを指定すると初期化子を命令に埋め込んで初期化します。

<V1.06.00 の生成コード>

```
_main:
        .stack _main = 16
        add 0xFFFFFFF0, r3
        movea 0x00000010, r0, r2
        mov #.STR.1, r5
        mov r3, r6
        add r6, r2
        br9 .BB.LABEL.1_2
.BB.LABEL.1_1:   ; entry
        ld23.dw 0x00000000[r5], r8
        st23.dw r8, 0x00000000[r6]
        add 0x00000008, r5
        add 0x00000008, r6
.BB.LABEL.1_2:   ; entry
        cmp r6, r2
        bnz9 .BB.LABEL.1_1
.BB.LABEL.1_3:   ; entry
        dispose 0x00000010, 0x00000000, [r31]
        .section .const, const
        .align 4
.STR.1:
        .dw 0x0001,0x0002,0x0003,0x0004
```

<V1.07.00 の生成コード>[-Oinline_init 指定時]

```
_main:
        .stack _main = 16
        add 0xFFFFFFF0, r3
        mov 0x00000001, r2
        st.w r2, 0x00000000[r3]
        mov 0x00000002, r2
        st.w r2, 0x00000004[r3]
        mov 0x00000003, r2
        st.w r2, 0x00000008[r3]
        mov 0x00000004, r2
        st.w r2, 0x0000000C[r3]
        dispose 0x000010, 0x000000, [r31]
```

4.8 メッセージ制御機能

警告メッセージをエラーメッセージに変更するためのコンパイル・オプション**-change_message** を追加することで、警告メッセージの見落としを防止することができるようになりました。

また、警告メッセージの出力を抑止するコンパイル・オプション-Xno_warning に 0510000 番台を指定できるようにしました。

- W0520000 ～ W0559999 が抑止可能 【V1.06.00 以前】
- **W0510000 ～ W0559999** が抑止可能 【V1.07.00 以降】

4.9 ヘキサ・ファイルのレコード長固定機能

インテル拡張ヘキサ・ファイル(.hex)とモトローラ・S タイプ・ファイル(.mot)の出力アドレスを、指定したアライメントで整合し、固定レコード長で出力する**-fix_record_length_and_align** オプションを追加しました。常に一定のレコード長でヘキサ・ファイルを出力するため、ヘキサ・ファイルの比較等の作業効率が改善します。

また、-byte_count オプションを拡張し、-form=stype 指定時にも指定できるようにしました。

4.10 リンク時のメッセージ追加

V1.06.00 以前では、整列条件の異なる同名セクションをリンクした場合に警告メッセージ W0561322 を出力していましたが、V1.07.00 では整列条件の異なる同名セクションで、かつ、その整列条件の一方が他方の倍数ではないセクションをリンクした際に警告メッセージ **W0561331** を出力するようにしました。

W0561322：Section alignment mismatch : " *セクション*"

W0561331：Section alignment is not adjusted : " *セクション*"

いずれの警告も整列条件は最大の指定を有効にしてリンクします。

W0561322 は動作に問題はありませんので無視しても構いませんが、W0561331 は動作に問題がある可能性がありますので、整列条件を見直す必要があります。

4.11 注意事項の改修

以下 4 件の注意事項を改修しました。注意事項の詳細につきましてはツールニュースをご確認ください。

- pow 関数の戻り値が不正となる注意事項 (No.9)
- switch 文中のラベルへの goto 文を使用している場合の注意事項 (No.16)
- FPU 命令を含む数学ライブラリ関数に関する注意事項 (No.17)
- ループ制御変数の終了条件が定数のループ文に関する注意事項 (No.18)

4.12 その他の変更・改善

ビルド時に内部エラーが発生する場合がありましたが、これを改善しました。

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/