CS+ RL78 コンパイラ CC-RL V1.03.00 リリースノート

R20UT3810JJ0100
Rev.1.00
2016.06.30

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-RL 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：

http://japan.renesas.com/cs+

第2章　ユーザーズ･マニュアルについて

本製品に対応したユーザーズ･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-RL コンパイラ	R20UT3123JJ0103
CS+ CC-RL ビルド・ツール操作編	R20UT3284JJ0102

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なう場合，コントロールパネルの「プログラムと機能」から，「CS+ CC-RL V1.03.00」を選択してください。

第4章　変更点

本章では、CC-RL V1.03.00 の変更点について説明します。

4.1　CC-RL V1.03.00の変更点

CC-RL V1.02.00 から V1.03.00 への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。

4.1.1　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。

(1)　異なるローカルスコープ内 auto 配列の領域を統合する最適化（スタックサイズの削減）

生存期間が重ならないブロック（{}）に属する auto 配列のスタック領域を統合するようにしました。

<ソースコード例>

```
int *g;
void func01(void)
{
  {
    int array01[10];
    g = array01;
    foo();
  }
  {
    int array02[10];      // array01[10]と array02[10]
                          // は生存期間が重ならない
    g = array02+2;
  }
}
```

<V1.02.00 の生成コード>

```
_func01:
        .STACK _func01 = 44        ;確保するスタックサイズ=44byte
        subw sp, #0x28
        movw ax, sp
        movw !LOWW(_g), ax
        call !!_foo
        movw ax, sp
        addw ax, #0x0018
        movw !LOWW(_g), ax
        addw sp, #0x28
        ret
```

```
<V1.03.00 の生成コード>
_func01:
        .STACK _func01 = 24        ;確保するスタックサイズ=24byte
        subw sp, #0x14
        movw ax, sp
        movw !LOWW(_g), ax
        call !!_foo
        movw ax, sp
        addw ax, #0x0004
        movw !LOWW(_g), ax
        addw sp, #0x14
        ret
```

(2) 定数伝播最適化の強化

ループ内で定数だとわかる計算を省略しました。

```
<ソースコード例>
int func02(int xtra, int n4, int e1[]) {
  int i,ix,j,k,l;
  j = 1;
  k = 2;
  l = 3;

  for (ix=0; ix<xtra; ix++) {
    for (i=0; i<n4; i++) {
      j = j*(k-j)*(l-k);   // j は常に 1
      k = l*k-(l-j)*k;     // k は常に 2
      l = (l-k)*(k+j);     // l は常に 3
      e1[l-2] = j+k+l;     // l-2 は常に 1, j+k+l は常に 6
      e1[k-2] = j*k*l;     // k-2 は常に 0, j+k*l は常に 6
    }
  }
  return e1[0]+e1[1];
}
```

```
<V1.02.00 の生成コード (1/3)>
.BB@LABEL@1_3:          ; bb49
        movw ax, [sp+0x0E]
        xor a, #0x80
        movw [sp+0x0A], ax
        movw ax, hl
        movw hl, sp
        xor a, #0x80
        cmpw ax, [hl+0x0A]
        bnc $.BB@LABEL@1_5
.BB@LABEL@1_4:          ; bb1
                        ;j*(k-j)*(l-k),l*k-(l-j)*k,(l-k)*(k+j),...
                        ;を毎回計算
        movw ax, [hl]
        subw ax, bc
        movw bc, ax
        movw ax, [sp+0x04]
        movw hl, ax
        mulh
        movw [sp+0x04], ax
        movw ax, [sp+0x02]
```

<V1.02.00 の生成コード (2/3)>

```
        subw ax, hl
        movw bc, ax
        movw ax, [sp+0x04]
        mulh
        movw hl, ax
        movw [sp+0x04], ax
        pop bc
        push bc
        movw ax, [sp+0x02]
        mulh
        movw [sp+0x0A], ax
        movw ax, [sp+0x02]
        movw bc, ax
        movw ax, [sp+0x00]
        subw ax, hl
        mulh
        movw bc, ax
        movw ax, [sp+0x0A]
        subw ax, bc
        movw [sp+0x02], ax
        addw ax, hl
        movw [sp+0x0A], ax
        movw hl, sp
        movw bc, ax
        movw ax, [hl]
        subw ax, [hl+0x02]
        mulh
        movw bc, ax
        movw [hl], ax
        movw ax, bc
        addw ax, ax
        addw ax, de
        addw ax, #0xFFFC
        movw hl, ax
        movw ax, [sp+0x0A]
        addw ax, bc
        movw [hl], ax
        movw ax, [sp+0x04]
        movw bc, ax
        movw ax, [sp+0x02]
        movw hl, ax
        mulh
        movw [sp+0x0A], ax
        pop bc
        push bc
        movw ax, [sp+0x0A]
        mulh
        movw bc, ax
        movw ax, hl
        addw ax, ax
        addw ax, de
        addw ax, #0xFFFC
        movw hl, ax
        movw ax, bc
        movw [hl], ax
        movw ax, [sp+0x06]
        movw hl, ax
```

```
<V1.02.00 の生成コード (3/3)>
        incw hl
        movw ax, hl
        movw [sp+0x06], ax
        movw ax, [sp+0x02]
        movw bc, ax
        movw [sp+0x02], ax
        br $!.BB@LABEL@1_3
```

```
<V1.03.00 の生成コード>
.BB@LABEL@1_3:          ; bb48
        movw ax, [sp+0x04]
        xor a, #0x80
        movw hl, ax
        movw ax, bc
        xor a, #0x80
        cmpw ax, hl
        bnc $.BB@LABEL@1_5
.BB@LABEL@1_4:          ; bb1
                        ;e1[1]と e1[0]に常に 6 を代入
        movw ax, #0x0006
        movw [de+0x02], ax
        movw [de], ax
        incw bc
        br $.BB@LABEL@1_3
```

(3) 帰納変数最適化の改善

冗長なループ帰納変数の更新コードを生成しないようにしました。

```
<ソースコード例>
void callee(unsigned i);
void caller(void){
  unsigned i;
  for(i=128; i != 0; --i){
    callee(i);
  }
}
```

```
<V1.02.00 の生成コード (1/2)>
_caller:
        .STACK _caller = 8
        subw sp, #0x04
        movw ax, #0x0080
        movw [sp+0x02], ax
        movw [sp+0x00], ax      ;冗長なループ帰納変数の初期化
.BB@LABEL@1_1:                  ; bb
        call !!_callee
        movw ax, [sp+0x02]
        addw ax, #0xFFFF
        movw [sp+0x02], ax
        movw ax, [sp+0x00]      ;冗長なループ帰納変数の更新
        decw ax
```

```
<V1.02.00 の生成コード (2/2)>
        movw [sp+0x00], ax
        bnz $.BB@LABEL@1_1
.BB@LABEL@1_2:                  ; return
        addw sp, #0x04
        ret
```

```
<V1.03.00 の生成コード>
_caller:
        .STACK _caller = 6
        push hl
        movw ax, #0x0080
        movw [sp+0x00], ax
.BB@LABEL@1_1:          ; bb
        call !!_callee
        movw ax, [sp+0x00]
        addw ax, #0xFFFF
        movw [sp+0x00], ax
        bnz $.BB@LABEL@1_1
.BB@LABEL@1_2:          ; return
        pop hl
        ret
```

(4) 不要コード削除最適化の改善

使われることのないコードの削除処理を強化しました。

```
<ソースコード例>
unsigned long test(unsigned long long variable, int var) {
  if (var) {
    variable &= 0x012345678abcdefULL;
  }
  return (variable >> 32);
}
```

```
<V1.02.00 の生成コード (1/2)>
_test:
        .STACK _test = 4
        or a, x
        movw ax, [sp+0x0A]
        movw bc, ax
        movw ax, [sp+0x08]
        movw de, ax
        movw ax, [sp+0x06]
        movw hl, ax
        movw ax, [sp+0x04]
        bz $.BB@LABEL@1_2
.BB@LABEL@1_1:          ; if_then_bb
        and a, #0xCD      ;ax([sp+4])はこの後使用されない
        xch a, x          ;
        and a, #0xEF      ;
        xch a, x          ;
        movw ax, hl       ;hl([sp+6])はこの後使用されない
```

<V1.02.00 の生成コード (2/2)>

```
        and a, #0x78      ;
        xch a, x          ;
        and a, #0xAB      ;
        xch a, x          ;
        movw ax, de
        and a, #0x34
        xch a, x
        and a, #0x56
        xch a, x
        movw de, ax
        movw ax, bc
        clrb a
        xch a, x
        and a, #0x12
        xch a, x
        movw bc, ax
.BB@LABEL@1_2:        ; if_break_bb
        movw ax, de
        ret
```

<V1.03.00 の生成コード>

```
_test:
        .STACK _test = 4
        or a, x
        movw ax, [sp+0x0A]
        movw bc, ax
        movw ax, [sp+0x08]
        movw de, ax
        bz $.BB@LABEL@1_2
.BB@LABEL@1_1:        ; if_then_bb
        and a, #0x34
        xch a, x
        and a, #0x56
        xch a, x
        movw de, ax
        movw ax, bc
        clrb a
        xch a, x
        and a, #0x12
        xch a, x
        movw bc, ax
.BB@LABEL@1_2:        ; if_break_bb
        movw ax, de
        ret
```

4.1.2 MISRA-C:2012ルールによるソース・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-misra2012 オプションの引数に、以下の番号を指定できるようにしました。

2.6 2.7

9.2 9.3

12.1 12.3 12.4

14.4

15.1 15.2 15.3 15.4 15.5 15.6 15.7

16.1 16.2 16.3 16.4 16.5 16.6 16.7

17.1 17.7

18.4 18.5

19.2

20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9 20.10 20.11 20.12 20.13 20.14

V1.02.00 と V1.03.00 でチェック可能な MISRA-C:2012 ルール数は以下の通りです。

ルール分類（ルール数）	*V1.02.00*	*V1.03.00*
必須ルール（10）	3	3
必要ルール（101）	31	**58**
推奨ルール（32）	7	**21**
合計ルール（143）	41	**82**

4.1.3 動的メモリ管理関数のセキュリティ強化【professional】

ヒープ・メモリ領域の解放時に、不正操作を検出する機能を追加しました。

本機能は専用の標準ライブラリをリンクすることにより使用可能です。

professional 版のライセンスが登録されていない場合は E0562310 エラーになります。

専用の標準ライブラリは以下のように動作します。

(1) calloc, malloc, realloc 関数内で、ヒープ・メモリ領域内にユーザー用に割り当てる領域の他に、不正操作を検出するための領域としてその外側の前後各 2 バイトを余分に割り当てます。

(2) free, realloc 関数内で、(a)～(c)のいずれかに該当していないかを判定します。

(a) 引数が(1)で割り当てたポインタでない

(b) 引数が既に開放済みのポインタである

(c) (1)で余分に割り当てた不正操作を検出するための領域が書き換わっている

(3) (2)の判定に該当した場合は不正な操作があったものとして__heap_chk_fail 関数を呼び出します。

__heap_chk_fail 関数はユーザーが定義する必要があります。ヒープ・メモリ領域の不正操作検出時に実行する処理を記述してください。例えば、以下のようなプログラムで、6 行目で余分に割り当てたヒープ・メモリ領域に書き込みがあった場合、8 行目でヒープ・メモリ領域を開放する際に__heap_chk_fail 関数を呼び出します。

```
1: #include <string.h>
2: #include <stdlib.h>
3: void func(char *str) {
4:    char *buf;
5:    buf = malloc(4);
6:    strcpy(buf, str);     //str の長さ次第でヒープ・メモリ領域を破壊
7:    …
8:    free(buf);
9: }
```

本機能を使用することにより、容易にセキュリティ対策（メモリ二重解放の問題やバッファ・オーバーフローの防止等）が可能となります。

4.1.4 オプション-Opipelineの追加

パイプライン最適化オプション-Opipeline を追加しました。

4.1.5 ビット操作組み込み関数の追加

ビットを操作する組み込み関数__set1, __clr1, __not1 を追加しました。

4.1.6 注意事項の改修

以下の 5 件の注意事項を解除しました。

- 条件アセンブル制御命令後に外部ラベルを定義した場合の注意事項 (CCRL#006)
- パッキングした構造体、共用体のメンバを初期化子に指定した場合の注意事項 (CCRL#007)
- 命令オペレーション一覧にない命令オペランドを記述した場合の注意事項 (CCRL#008)
- 割り込みハンドラ内でレジスタを上書きするコードを出力する注意事項 (CCRL#009)
- 最適化の適用範囲に関する注意事項 (CCRL#010)

4.1.7 その他変更・改善

主に以下のような変更・改善を行いました。

(a) デバッグ情報の改善

デバッグ時に C ソース、アセンブリソースが正しく表示できない場合がありましたが、これを改善しました。

(b) 内部エラーの改善

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。

第5章　注意事項

CC-RL V1.03.00 の注意事項についてはマニュアルを参照してください。

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、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。

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/