RENESAS ツールニュース

RENESAS TOOL NEWS 2014年6月1日：140601/tn2

CubeSuite+版 RXファミリ用 C/C++コンパイラパッケージ V1 および High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラパッケージ ご使用上のお願い

CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1 および High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラパッケージ の使用上の注意事項を連絡します。

- volatileまたはevenaccess修飾された構造体のメンバに関する注意事項 (RXC#026)
- リアルタイムOSのシステムコールの発行に関する注意事項 (RXC#027)
- 呼び出し後の処理が同一である関数呼び出しに関する注意事項 (RXC#028)
- &演算子の使用に関する注意事項 (RXC#030)

注: 各注意事項の後ろの番号は、注意事項の識別番号です。

---

1. volatileまたはevenaccess修飾された構造体のメンバに関する注意事項 (RXC#026)

1.1 該当製品およびバージョン

- CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1
  CC-RXコンパイラ V1.02.00 ～ V1.02.01
- High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラパッケージ V.1.00 Release 00 ～ V.1.02 Release 01

注: CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1 の受注型名は、R0C5RX00QSW01D および R0C5RX00QSW01Nです。
High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラパッケージの受注型名は、R0C5RX00XSW01Rです。

1.2 内容

関数の戻り値がvolatileまたは__evenaccess修飾された構造体のメンバのポインタ、または参照渡しの場合にvolatileまたは__evenaccess修飾が無効になる場合があります。

1.3 発生条件

以下の条件をすべて満たす場合に発生します。

(1) 関数戻り値が、構造体ポインタ (注1)、または構造体の参照渡し (注2) である。

(2) (1)の構造体に次のいずれかの修飾子が付加されている。
   (a) volatile
   (b) __evenaccess

(3) (1)の関数呼出しに対して "."演算子 または "->"演算子を使用し、構造体のメンバを参照している。

(4) (1)の関数呼び出しにインライン展開の最適化が適用される。

注1: CおよびC++の場合に該当します。
注2: C++の場合に該当します。

問題が発生すると、発生条件(3)の構造体のメンバ参照時に、発生条件(2)の修飾子の効果が無効になります。その結果、無効になった修飾子を無視した最適化により誤ったコードが生成されることがあります。

発生例:

```
------------------------------------------------------------------------
struct ST { unsigned char PRT; };
volatile struct ST *st2f()     /* 発生条件(1)および(2) */
{
   return (volatile struct ST*)0x2000;
}
unsigned char func_st2(void)
{
   st2f()->PRT = 0x10;     /* 発生条件(3)および(4) */
   st2f()->PRT = 0x11;     /* 発生条件(3)および(4) */
   st2f()->PRT |= 0x22;    /* 発生条件(3)および(4) */ /* (NG-1) */
   return st2f()->PRT;     /* 発生条件(3)および(4) */ /* (NG-2) */
}
------------------------------------------------------------------------
```

発生例に対する出力コード:

inline=100有効時

```
------------------------------------------------------------------------
MOV.L       #00002000H,R5
MOV.B       #10H,[R5]
MOV.B       #11H,[R5]
MOV.B       #33H,[R5]      ; (NG-1) 読出しがない
```

```
MOV.L       #00000033H,R1  ; (NG-2) 読出しがない
-----------------------------------------------------------------------
```

1.4 回避策

以下のいずれかの方法で回避してください。

(1) 関数の戻り値が構造体のアドレスの場合は、構造体へのポインタを経由して
メンバを参照する。

(2) 構造体定義の際、メンバにvolatileを修飾する。

(3) 次のいずれかの方法で、当該関数がインライン展開されないようにする。

(a) 当該関数に、#pragma noinlineを指定する。
なお、当該関数に#pragma inline指定している場合は、同時に
この#pragma inlineの指定をはずす。

(b) コンパイル時に -inline=0 を指定する。

2. リアルタイムOSのシステムコールの発行に関する注意事項 (RXC#027)

2.1 該当製品およびバージョン

- CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1
CC-RXコンパイラ V1.02.00 ～ V1.02.01
- High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラ
パッケージ V.1.00 Release 00 ～ V.1.02 Release 01

注: CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1 の受注型名は、
R0C5RX00QSW01D および R0C5RX00QSW01Nです。
High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラ
パッケージの受注型名は、R0C5RX00XSW01Rです。

2.2 内容

関数の最後でリアルタイムOS (注) のシステムコールを発行すると、
システムコール復帰後にシステムコールを発行した関数の最後にRTS命令が
生成されていない場合があります。

注: 該当製品は以下の通りです。

- RI600V4 V1.03.00以前
- RI600PX V1.02.00以前
- RI600/4 V.1.01 Release 01以前
- RI600/PX V1.00 Release 01以前

2.3 発生条件

以下の条件をすべて満たす場合に発生します。

(1) optimize=2またはoptimize=maxオプションを使用している。

(2) 関数の出口が以下のいずれかの制御文によって複数存在する (注)。

- if文
- switch文
- for文

- while文

(3) 関数の全ての出口の直前に関数呼び出しがある。
(4) (3)の関数呼び出しの少なくとも1つが、#pragma oscallを指定した関数呼び出しである。
(5) (2)の関数に以下の#pragmaを指定していない。
- #pragma interrupt
- #pragma task
- #pragma taskexception
- #pragma almhandler
- #pragma cychandler

(6) (2)の関数の引数および自動変数にvolatile修飾子を付加したものがない。
(7) (2)の関数の引数および自動変数のアドレスを参照していない。
(8) (2)の関数内にsetjmp()の呼び出しがない。

注: 制御文が、関数の出口の直前に、次のような形式で記述された場合、
関数の出口へのたどり着き方が2つに分かれるため、コンパイラでは
関数の出口は2つと判断します。

条件(2)で出口が複数とみなされる例:

```
-----------------------------------
void func(void)
{
 . . . . . .
      /* 関数の出口 */
   if (条件式) {
 . . . . . .
      /* 関数の出口 */
   }
   return;
}
-----------------------------------
```

発生例1:
誤ったコードになるケース

```
-------------------------------------------------------
#pragma oscall /s=0 5 tsk()
void tsk(void);
void sub(void);
int a;
void func()
{
   int b[1] = {0};
   if (a) {
      sub();    /* 発生条件(2)および(3) */
   }else{
```

```
    tsk();    /* 発生条件(2),(3)および(4) */
  }
}
void dummy(void){a=10;}
```

発生例1のコンパイル結果:
-optimize=2, -size および -output=src 指定時の例

```
_func:
    SUB     #04H,R0
L10:
    MOV.L   #L14,R5
    MOV.L   #_a,R4
    MOV.L   [R5],[R0]
    MOV.L   [R4],R5
    CMP     #00H,R5
    BEQ     L12
L11:
    ADD     #04H,R0
    BRA     _sub
L12:
    .ASSERT   'tsk' >> ..file@.mrc
    INT     #05H
            ; RTS命令が抜けている。
_dummy:
    MOV.L   #_a,R4
    MOV.L   #0000000AH,[R4]
    RTS
```

発生例2:
アセンブル時にエラーになるケース

```
#pragma oscall /s=0 5 tsk()
void tsk(void);
void sub(void);
int a;
void func()
{
  int b[1] = {0};
  if (a) {
    tsk();  /* 発生条件(2)および(3) */
  }else{
    sub();  /* 発生条件(2),(3)および(4) */
```

```
    }
}
```

発生例2に該当する場合、アセンブル時または-output=objの
オブジェクトファイルを生成するコンパイルを行うと、
以下のアセンブルエラーが発生します。
 A2111 (E) Symbol is undefined

発生例2のコンパイル結果:
-optimize=2, -size および -output=src 指定時の例

```
func:
        .ASSERT    'tsk' >> ..file@.mrc
        INT        #05H
        BRA        L13  ; 存在しないラベル(L13)を指定
L12:
        ADD        #04H,R0
        BRA        _sub
```

2.4 回避策

以下のいずれかの方法で回避してください。
(1) optimzie=0またはoptimize=1を使用する。
(2) 該当する関数のいずれかの出口の直前に、ダミーの命令となるコードを
   挿入する。
    例: 組込み関数nop()

発生例1 に対する回避策(2)の適用例:

```
#include <machine.h>    /* 追加: 組込み関数nop()を有効にする */
#pragma oscall /s=0 5 tsk()
void tsk(void);
void sub(void);
int a;
void func()
{
    int b[1] = {0};
    if (a){
        sub();
        /* ここにnop()の呼び出しを追加しても回避可能 */
    }else{
        tsk();
        /* ここにnop()の呼び出しを追加しても回避可能 */
    }
        nop();   /* 追加: nop()の呼び出し */
```

```
}
void dummy(void){a=10;}
------------------------------------------------------------
```

発生例2 に対する回避策(2)の適用例:

```
------------------------------------------------------------
#include <machine.h>    /* 追加: 組込み関数nop()を有効にする */
#pragma oscall /s=0 5 tsk()
void tsk(void);
void sub(void);
int a;
void func()
{
   int b[1] = {0};
   if (a) {
         tsk();
         /* ここにnop()の呼び出しを追加しても回避可能 */
   }else {
         sub();
         /* ここにnop()の呼び出しを追加しても回避可能 */
   }
   nop();   /* 追加: nop()の呼び出し */
}
------------------------------------------------------------
```

3. 呼び出し後の処理が同一である関数呼び出しに関する注意事項 (RXC#028)

3.1 該当製品およびバージョン

- CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1
  CC-RXコンパイラ V1.02.00 ～ V1.02.01
- High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラ
  パッケージ V.1.00 Release 00 ～ V.1.02 Release 01

注: CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1 の受注型名は、
R0C5RX00QSW01D および R0C5RX00QSW01Nです。
High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラ
パッケージの受注型名は、R0C5RX00XSW01Rです。

3.2 内容

呼び出し後の処理が同一である関数呼び出しが複数ある場合、リンク時
最適化における関数の呼び出し先が正しくならずに、誤ったコードを
生成する場合があります。

3.3 発生条件

以下の条件をすべて満たす場合に発生することがあります。
(1) コンパイル時のオプション指定が、以下のすべてを満たしている。

(a) -goptimizeオプションを使用している。
(b) -optimize=2 (-optimize指定がない場合を含む) または -optimize=max を指定している。
(c) -speedオプションを使用している。
(d) -branch=32を指定している。

(2) リンク時のオプション指定が、次のいずれかに該当する。
(a) -optimize=symbol_delete 以外の -optimizeオプションの指定がある。
(b) -optimize または -nooptimize のどちらの指定もない。

(3) 関数内に同一関数の呼び出し (注) が複数ある。

(4) (3)のそれぞれの関数呼び出し直後から関数出口までの全経路で処理内容が同じである。

注: 実行時ルーチンの呼び出しも含みます。
次の演算が、それぞれのオプション指定でコンパイルされた場合に、
実行時ルーチンの呼び出しに置き換わります。
1. -nofpuまたはRX200選択時の単精度浮動小数点型の演算
2. 倍精度浮動小数点型の演算
3. 64ビット整数型の、加減算以外の演算
4. -nouse_div_inst選択時の32ビット整数型および単精度浮動小数点型の除算および剰余算

問題が発生すると、リンク時の最適化で用いる情報が異常となります。
その結果、最適化処理により発生条件(3)に該当する関数呼び出しの分岐先アドレスで正しくないコードが生成されます。

発生例:
コンパイル時オプションに -goptimize, -optimize=2, -speed および -branch=32が指定されていて、かつ、リンク時オプションの -nooptimize または -optimize のどちらの指定もない場合

```
int a;
void func(int b)
{
   if (b) {
      a = sub(1);   // 発生条件(3)および(4)
   }else {
      a = sub(2);   // 発生条件(3)および(4)
   }
}
```

上記例の場合、リンク後に一つ目のsub(1)への呼び出しのアドレスが自分自身に分岐して永久ループになる誤ったコードが生成されます。

3.4 回避策

以下のいずれかの方法で回避してください。

(1) コンパイル時に -goptimizeオプションを指定しない。
(2) コンパイル時に次のいずれかのオプション指定をする。
  (a) -optimize=0 または -opttimize=1
  (b) -size
  (c) -branch=24 または -branch=16
(3) リンク時に次のオプションを指定する。
  -nooptimize
(4) 発生条件(4)の経路のうちの一つに、ダミーの命令を追加する。

回避策(4)の回避例1:
ダミーのグローバル変数の値を更新するコードを挿入する例

```
---------------------------------------------------------------
int dummy;    // ダミーのグローバル変数
int a;
void func(int b)
{
   if (b) {
      a = sub(1);
      dummy = 0;   // ダミーのグローバル変数に値を代入
   } else {
      a = sub(2);
   }
}
---------------------------------------------------------------
```

回避策(4)の回避例2:
組込み関数 nop()を挿入する例

```
---------------------------------------------------------------
#include <machine.h>  // nop() を使用するために必要
int a;
void func(int b)
{
   if (b) {
      a = sub(1);
      nop();    // 組み込み関数 nop() をダミーとして挿入
   } else {
      a = sub(2);
   }
}
---------------------------------------------------------------
```

4. &演算子の使用に関する注意事項 (RXC#030)

4.1 該当製品およびバージョン

- CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1
  CC-RXコンパイラ V1.02.00 ～ V1.02.01

- High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラパッケージ V.1.00 Release 00 ～ V.1.02 Release 01

注: CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1 の受注型名は、R0C5RX00QSW01D および R0C5RX00QSW01Nです。
High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラパッケージの受注型名は、R0C5RX00XSW01Rです。

4.2 内容

変数と定数値との、&演算子を用いたビット論理積演算 (&演算)で誤った結果となる場合があります。

4.3 発生条件

以下の条件をすべて満たす場合に発生することがあります。
(1) コンパイル時に-optimize=2 (-optimize指定がない場合を含む) または -optimize=max を指定している。
(2) 1または2バイト型変数への代入を行っている。
(3) (2)で代入する値は、&演算の結果である。
(4) (3)の&演算の両オペランドは、どちらも8バイト型ではない。
(5) (3)の&演算の両オペランドのうち、一方は定数値である。
(6) (5)の定数値のうち、(2)の代入先の変数内で有効になるビット列の中に0であるものが1ビットだけある。 (発生例の注1で該当する例を記載しています)
(7) (2)の代入後に、代入先変数をよりサイズの大きい型に拡張している。
(8) (3)の&演算の定数値でないオペランドは、(2)の代入先の型の範囲外の値となる可能性がある。 (発生例の注2で該当しない例を記載しています)

発生例:

```
--------------------------------------------------------------------------
unsigned long yyy;
void func(unsigned long xxx) {          // 発生条件(8)(注2)
   unsigned char aaa = xxx & 0x000000fe;  // 発生条件(2),(3),(4),
                                  // (5)および(6)(注1)

   yyy = aaa;                          // 発生条件(7)
}
--------------------------------------------------------------------------
```

上記の例では、&演算子による上位の24ビット分のマスクが行われないコードが生成されます。
このため、当該コードの実行時にxxxの値が0x00000100から0xffffffffの範囲にある場合、本来生成されるべきコードと異なる結果となります。

注1:
発生条件(6)に該当する例は次の通りです。
代入先変数が1バイトの場合:

```
-----------------------------------------------------------------
unsigned char  a1 = xxx & 0x012345fe; // 下位1バイトは1ビットだけ0
unsigned char  a2 = xxx & 0x000000fb; // 下位1バイトは1ビットだけ0
-----------------------------------------------------------------
```

代入先変数が2バイトの場合:

```
-----------------------------------------------------------------
unsigned short a3 = xxx & 0x0123fffe; // 下位2バイトは1ビットだけ0
-----------------------------------------------------------------
```

注2:

発生条件(8)に該当しない例を挙げます。
下記例は、条件を満たすfunc関数に対する実行コードには問題が
ありますが、foo関数から呼び出した場合に限り、誤った実行結果に
なることはありません。

```
-----------------------------------------------------------------
unsigned long yyy;
void func(unsigned long xxx) {          // 発生条件(8)
   unsigned char aaa = xxx & 0x000000fe; // 発生条件(2),(3),(4),
                                         // (5)および(6)
   yyy = aaa;                            // 発生条件(7)
}
. . . . . .
unsigned char zzz;
void foo(void)
{
   func(zzz);  // zzzの値が代入先aaaの範囲を超えることはないため、
               // この呼び出しに限り、誤った結果にならない。
}
-----------------------------------------------------------------
```

4.4 回避策

以下のいずれかの方法で回避してください。
(1) 発生条件(2)の代入先変数を4バイト型に変更する。
(2) 発生条件(2)の代入先変数をvolatile修飾する。
(3) コンパイル時に-optimize=0または-optimize=1オプションを指定する。

回避策(1)の回避例:

```
-----------------------------------------------------------------
unsigned long yyy;
void func(unsigned long xxx) {
   unsigned long aaa = xxx & 0x000000fe; // 4バイト型に変更
   yyy = aaa;
}
-----------------------------------------------------------------
```

5. 恒久対策

CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1 および
High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラ
パッケージでの改修予定はありません。回避策を適用してください。

技術的な内容に関するご不明点は、以下のWebページからお問い合わせください。
技術問合せ: https://www.renesas.com/support/contact.html

なお、これら4件の注意事項はCubeSuite+版 RXファミリ用C/C++コンパイラ
パッケージ V2 では改修済みです。
CubeSuite+版 RXファミリ用C/C++コンパイラパッケージ V1 を使用されている
お客様は V2 へのバージョンアップ (注) もご検討ください。

High-performance Embedded Workshop版 RXファミリ用C/C++コンパイラ
パッケージは V1 のみで、V2 はありません。
High-performance Embedded Workshop版を使用されているお客様は、
CubeSuite+版または統合開発環境なし版へ移行し、コンパイラのバージョンを
V1 から V2 へバージョンアップ (注) することもご検討ください。

注: 有償でのバージョンアップになります。