RENESAS ツールニュース

RENESAS TOOL NEWS 2011年10月16日：111016/tn2

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

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 5 件を連絡します。

- 関数呼び出し、またはvolatile修飾された変数をオペランドにもつ -1との論理和"|"、または0との論理積"&"演算の注意事項 (RXC#016)
- ループ内に、ループ制御変数を含むif文または条件演算子がある場合の注意事項 (RXC#017)
- if文または条件演算子を含むループ中で参照している変数の値を、代入文以外の文によって更新する場合の注意事項 (RXC#018)
- 配列をメンバに持つ構造体を構造体のポインタを用いて配列を参照した場合の注意事項 (RXC#019)
- 1つの関数内にchar型の同じ配列要素の参照が複数ある場合の注意事項 （RXC#020）

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. 関数呼び出し、またはvolatile修飾された変数をオペランドに持つ-1との論理和

"|"、または0との論理積"&"演算の注意事項 (RXC#016)

該当バージョン：
V.1.00 Release 00 ～ V.1.01 Release 00

内容:
関数呼び出し、または volatile 修飾された変数をオペランドに持つ
演算式において、その演算が-1との論理和"|"、または0との論理積"&"で
ある場合に、その関数呼び出しが行われない、またはvolatile 修飾した
変数へのアクセスが行われない場合があります。

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) 以下のいずれかの型のオペランドを持つビットごとの論理和"|"、または
ビットごとの論理積"&"演算がある。
long long, signed long long および unsigned long long
(2) (1)の演算は以下のいずれかを満たす。

(2-1) "|"の場合、-1をオペランドにもつ。
(2-2) "&"の場合、0をオペランドにもつ。
注： オペランドは、定数伝播 (const修飾された外部変数の定数伝播を含む) の最適化により変数が定数に置き換わった場合も含む。

(3) (2)の演算のもう一方のオペランドが、以下のいずれかである。
(3-1) volatile 修飾された変数
(3-2) -volatileオプションを使用している場合、外部変数
(3-3) 関数呼び出し
(3-4) (3-1)、(3-2)および(3-3)のいずれかを含む式

発生例:

```
-------------------------------------------------------------
long long a;
int sub();
main(){
  long long x;
  x = -1LL;                    // 発生条件(2-1)
  a = ((long long)(sub()+2)) | x;  // 発生条件(1)、(2-1)、および(3-4)
}
-------------------------------------------------------------
```

上記の例をコンパイルすると、((long long)(sub()+2)) | -1LL が誤って削除され、関数呼び出し sub() が行われません。

コンパイル結果:

```
-----------------------------------------
_main:
  MOV.L      #0FFFFFFFFH,R4 ; -1LL
  MOV.L      #_a,R5
  MOV.L      R4,[R5]        ; (part of) a
  MOV.L      R4,04H[R5]     ; (part of) a
  MOV.L      #00000000H,R1
  RTS
-----------------------------------------
```

回避策：

発生条件(2)の定数 -1 または 0 を、volatile 修飾された変数に代入し定数の代わりにその変数を使用する。

発生例の回避例：

```
-------------------------------------------------------------
 long long a;
 int sub();
 main(){
   volatile long long x;       // 発生条件(2)の定数の代わりに、
                               volatile 修飾された変数を使用する
```

```
  x = -1LL;                    // 発生条件(2-1)
  a = ((long long)(sub()+2)) | x;  // 発生条件(1)および(3-4)
 }
 ------------------------------------------------------
```

2. ループ内に、ループ制御変数を含むif文または条件演算子がある場合の注意事項

(RXC#017)

該当バージョン：
V.1.00 Release 00 ～ V.1.01 Release 00

内容:
ループ内に、ループ制御変数を含むif文または条件演算子がある場合、そのif文または条件演算子の判定を誤る場合があります。
注： ループ制御変数は、ループ内で繰り返しごとに値が一定に増加または減少し、かつループを繰り返すかどうかの判定式で参照される変数です。

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) -optimize=0 および -optimize=1 オプションを使用していない。
(2) ループ制御変数を持つループが存在する。
(3) (2)のループ制御変数の初期値および上限値はいずれも定数である。
　注： 定数は、定数伝播 (const修飾された外部変数の定数伝播を含む) の最適化により変数が定数に置き換わった場合も含む。
(4) (2)のループ内にif文または条件演算子「?:」がある。
(5) (4)のif文または条件演算子の制御式は、以下をすべて満たす。
　(5-1) 大小比較演算子「<」「>」「<=」または「>=」による比較式である。
　(5-2) 一方のオペランドは、(2)のループ制御変数である。
　(5-3) もう一方のオペランドは、ループ制御変数の上限値以下の整数定数である。
　　注： 整数定数は、定数伝播 (const修飾された外部変数の定数伝播を含む) の最適化により変数が定数に置き換わった場合も含む。

発生例:
```
 ------------------------------------------------------------------
  int main(void) {
   int  j;
   char a1[6],a2[6],a3[6];
   for ( j = 0; j < 6; j++ ){       // 発生条件(2)、(3)および(4)
    if ((j >= 0 && j < 2) || (j >= 4 && j < 6))  // 発生条件(5)
```

```
      a1[j] = 1;
    else
      a1[j] = 0;

    if (j < 4)                          // 発生条件(5)
      a2[j] = 2;
    else
      a2[j] = 0;

    if (j >= 0 && j < 1)                // 発生条件(5)
      a3[j] = 3;
    else
      a3[j] = 0;
   }
  }
```

上記の発生例をオプション -cpu=rx600 -speed を使用してコンパイルすると、
問題が発生します。
なお、本例では、発生条件(5)に該当する一つ目の制御式「j < 2」でのみ
問題が発生します。

コンパイル結果：

以下のコンパイル結果において、j(R5) が 2 のとき、(B)のR1 は -1になり
ます。この結果、(C)で L11 に分岐し、(A)で a1[3] に 1 が設定され、
誤った結果になります。
正しく動作した場合は、(C)で分岐せず、(D)でa1[j] に 0 が設定されます

```
    ADD       #0FFFFFFE8H,R0,R0
    MOV.L     #00000000H,R5
    MOV.L     #00000002H,R1
    MOV.L     R0,R2
    ADD       #08H,R0,R3
    ADD       #10H,R0,R4
  L11:
    MOV.B     #01H,[R3]    ; (A) 誤ってa1[3] に1が設定される
    MOV.B     #02H,[R2]
    CMP       #00H,R5
    BLT       L12
    CMP       #01H,R5
    BGE       L12
    MOV.B     #03H,[R4]
  L15:
    ADD       #01H,R5
    ADD       #01H,R4
    ADD       #01H,R3
```

```
        ADD       #01H,R2
        CMP       #06H,R5
        BGE       L16
        SUB       #01H,R1     ; (B) j(R6) が 3 の時、R1が-1になる
        BNE       L11         ; (C) 誤ってL11 に分岐
        CMP       #04H,R5
        BGE       L20
        MOV.B     #00H,[R3]  ; (D) 正しくはここでa1[3] に0が設定される
        MOV.B     #02H,[R2]
        BRA       L12
    L20:
        CMP       #06H,R5
        BLT       L22
        MOV.B     #00H,[R3]
        BRA       L23
    L22:
        MOV.B     #01H,[R3]
    L23:
        MOV.B     #00H,[R2]
    L12:
        MOV.B     #00H,[R4]
        BRA       L15
    L16:
        MOV.L     #00000000H,R1
        RTSD      #18H
```

回避策：

以下のいずれかの方法で回避してください。

(1) -optimize=0 または -optimize=1 を使用する。

(2) 発生条件(2)のループ制御変数を volatile 修飾する。

3. if文または条件演算子を含むループ中で参照している変数の値を、代入文以外の文によって更新する場合の注意事項 (RXC#018)

該当バージョン：

V.1.00 Release 00 ～ V.1.01 Release 00

内容:

if文または条件演算子を含むループがあり、そのループ内で参照している外部変数またはstatic変数の値を、この変数への代入文以外の文によって更新する場合、ループ抜け出し後に更新前の値に戻る場合があります。

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) -optimize=0 および -optimize=1 オプションを使用していない。
(2) -scopeオプションが有効である。
-optimize=2を使用もしくは-optimizeオプションを使用しない場合も、
-scopeオプションが有効になります。
(3) 最適化範囲が分割される関数がある。
注: -messageを使用した場合、上記の関数に対しては以下のメッセージが
出力される。
C0101 (I) Optimizing range divided in function "関数名"
"関数名"の最適化範囲が複数に分割されました。
(4) (3)の関数内にif文または条件演算子「?:」を含むループが存在する。
(5) (4)のループは内側に別のループを含まない、かつ、無限ループでない。
(6) 以下をすべて満たす外部変数またはstatic変数がある。
(6-1) (4)のループ内で定義および参照されない
(6-2) (3)の関数内で定義または参照されている
(7) (6)の外部変数およびstatic変数はvolatile修飾されていない。
かつ -volatileオプションを使用していない。
(8) (6)の外部変数は(4)のループ内で、この変数への代入文以外の文によって
更新される。(文の例： 関数呼び出し)

発生例：

```
-----------------------------------------------------
int aaa,xxx=0,yyy=0,n;       // 発生条件(7)
void sub(void);
void func(void)              //発生条件(3)
{
  int i;
. . . . . .
  aaa = 0;                   // 発生条件(6)
. . . . . .
  for (i = 0; i < n ;i++) {   // 発生条件(4)(5)
    if(xxx == yyy){
      sub();                 // 発生条件(8)
    }
  }
. . . . . .
}

void sub(void)
{
  aaa++;
}
-----------------------------------------------------
```

コンパイル結果：

以下のコンパイル結果において、(A)でロードした aaa の値を(C)で書き戻すため、(B)で変更されたaaa の値がループ抜け出し後にループ直前の値に戻ります。

```
-------------------------------------------------------------
  _func:
. . . . . . . . . . . . . . . . . . .
        MOV.L     #_aaa,R8
        MOV.L     [R8],R7       ; (A)
        MOV.L     #00000000H,R6
        BRA       L11
  L12:
        MOV.L     #_xxx,R2
        MOV.L     [R2],R4
        MOV.L     #_yyy,R3
        CMP       [R3],R4
        BNE       L14
  L13:
        BSR       _sub          ; aaa++; . . . (B)
  L14:
        ADD       #01H,R6
  L11:
        MOV.L     #_n,R1
        CMP       [R1],R6
        BLT       L12
  L15:
        MOV.L     R7,[R8]       ; (C)
. . . . . . . . . . . . . . . . . . .
  _sub:
        MOV.L     #_aaa,R4
        MOV.L     [R4],R5
        ADD       #01H,R5
        MOV.L     R5,[R4]
        RTS
-------------------------------------------------------------
```

回避策：

以下のいずれかの方法で回避してください。

(1) -optimize=0 または -optimize=1 オプションを使用する。
(2) -noscope オプションを使用する。
(3) -volatile オプションを使用する。
(4) 発生条件(6)の外部変数をvolatile修飾する。
(5) 発生条件(4)のループ内で発生条件(6)の変数を参照する。
(6) 発生条件(3)に該当する関数を、-messageオプションを指定して
　メッセージC0101(I)が出力されなくなるまで分割する。

4. 配列をメンバに持つ構造体を構造体のポインタを用いて配列を参照した場合の

注意事項 (RXC#019)

該当バージョン：
V.1.00 Release 00 ～ V.1.01 Release 00

内容:
配列をメンバに持つ構造体を構造体のポインタまたは配列の添え字に定数を使用して配列を参照するとC4098のInternal Errorが発生しコンパイルが正しく終了しない、または配列を参照できない場合があります。

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) -optimize=1, -optimize=2, -optimize=maxオプションのいずれかを使用している。
(2) 4バイトのスカラ型である配列型のメンバを持つ構造体または共用体がある。
(3) 以下の(a)または(b)のメンバをもつ、構造体または共用体がある。
(a) (2)の構造体または共用体のポインタ
(b) 4バイトのスカラ型
(4) (3)の構造体または共用体を仮引数に使用している関数がある。
(5) (4)の仮引数は、レジスタを用いて関数に渡される。
仮引数の型によって、関数に渡す時にレジスタを使用する場合と、使用しない場合があります。ユーザーズマニュアル8章8.2.3項「引数の設定、参照に関する規則」を参照ください。
(6) (5)の仮引数の内、(3)のメンバを以下の(a)または(b)の方法で参照している式がある。
(a) (3)のメンバが(2)の構造体または共用体のポインタで、型を変更せずに参照。
(b) (3)のメンバを(2)の構造体または共用体のポインタにキャストして参照。
(7) (6)の式をCとしたとき、C->D[E]に相当する式があり、Dは(2)の配列型のメンバ、Eは任意の定数である。
(8) (4)の仮引数のアドレスか、構造体または共用体のメンバのアドレスが(4)の関数内で参照されていない。

発生例1：
C4098のInternal Errorが発生する場合の例
```
------------------------------------------------------------
typedef struct { unsigned long ul[2]; } S_A; // 発生条件(2)
typedef struct {
  S_A *p;
```

```
} S_B;              // 発生条件(3)-(a)および(5)
int a;
void func(S_B pm)     // 発生条件(4)および(5)
{
                  // 発生条件(8)
   a = pm.p->ul[1];  // 発生条件(6)-(a)および(7)
}
```

発生例2:

配列を正しく参照できない例

```
typedef struct { unsigned long ul[2]; } S_A; // 発生条件(2)
typedef struct {
   S_A *p;
} S_B;                  // 発生条件(3)-(a)および(5)
int a;
void func(long m1, S_B pm2) // 発生条件(4)および(5)
{
   S_A *ptr;
   ptr = pm2.p;
                    // 発生条件(8)
   a = m1 + ptr->ul[0];    // 発生条件(6)-(a)および(7)
 }
```

本例では、ptr->ul[0] に対して、(unsigned long)pm2.p と等価なコードを誤って生成します。

回避策：

以下のいずれかの方法で回避できます。

(1) 発生条件(4)の構造体または共用体の仮引数のアドレスを取得し、発生条件(8)に該当しないようにする。

(2) 発生条件(2)の構造体または共用体の配列を指すポインタを経由し、配列の添字付け参照にして発生条件(7)に該当しないようにする。

(3) 発生条件(3)の構造体または共用体にダミーのメンバを追加して17バイト以上にし、発生条件(5)に該当しないようにする。

(4) optimize=0を指定する。

発生例1の回避例:

```
typedef struct { unsigned long ul[2]; } S_A;
typedef struct {
  S_A *p;
} S_B;
int a;
```

```
void func(S_B pm)
{
  unsigned long *ptr = pm.p->ul;   // 発生条件(2)の構造体または共用体
                                   の配列を指すポインタ
  a = ptr[1];                      // ポインタを用いてアクセスする
}
```

発生例1の回避例2:

```
typedef struct { unsigned long ul[2]; } S_A;
   typedef struct {
     S_A *p;
   long long dummy[2];    // メンバを追加して発生条件(3)の構造体または
                             共用体を17バイト以上にする
  } S_B;
  int a;
  void func(S_B pm)
  {
    a = pm.p->ul[1];
  }
```

発生例2の回避例:

```
typedef struct { unsigned long ul[2]; } S_A; // 発生条件(2)
typedef struct {
  S_A *p;
} S_B;
int a;
void func(long m1, S_B pm2)
{
  S_B *dummy_ptr = &pm2;  // 発生条件(3)の構造体まはた共用体の
                             仮引数のアドレスを取得する
   S_A *ptr;
   ptr = pm2.p;
   a = m1 + ptr->ul[0];
}
```

---

5. 1つの関数内にchar型の同じ配列要素の参照が複数ある場合の注意事項（RXC#020）

該当バージョン：

V.1.00 Release 00 ～ V.1.01 Release 00

内容:
1つの関数内にchar、signed char、または unsigned char 型の同じ配列
要素の参照が複数ある場合、配列要素に正しくアクセスできないことがあります。

発生条件:
以下の発生条件Aまたは発生条件Bを満たす場合に発生することがあります。

発生条件A：
以下の(A1)～(A6)をすべて満たす場合に発生します。
(A1) -optimize=0 および -optimize=1 オプションを使用していない。
(A2) char、signed char、または unsigned char 型の配列、または
それらの型を指すポインタ変数がある。
(A3) (A2)の配列の同じ要素を、1つの関数内で2回以上参照する。
なお要素は、配列参照と等価な間接参照式を記述した場合も含む。
(間接参照式の例： a[exp]と*(a+exp)は等価)
(A4) (A3)の配列要素の添え字式は、加算式または減算式であり、かつ
その一方のオペランドは変数で、かつもう一方のオペランドは0以外の
定数である。
(A3)の間接参照式の例の場合、expの部分が添え字式に相当します。
注： 定数は、定数伝播 (const修飾された外部変数の定数伝播を含む)
の最適化により変数が定数に置き換わった場合も含む。
(A5) (A4)の変数は、char、signed char、unsigned char、short、
signed short、または unsigned short 型である。
(A6) (A4)の添え字式の値が、(A5)の変数の型で表現可能な最大値より
大きい値になる。

発生条件Aの発生例:
```
------------------------------------------------------------
  signed char S,*ary;                   // 発生条件(A2)

  void func(signed char par)              // 発生条件(A5)
  {
    if (ary[par+1] > 10) {         // 発生条件(A3)および(A4)
      S = ary[par+1];              // 発生条件(A3)および(A4)
    }
    return;
  }
------------------------------------------------------------
```
上記の発生例では、par が 127 の場合に、発生条件(A6)に該当します。

コンパイル結果：
```
------------------------------------------------------------
 _func:
      MOV.L       #_ary,R4
```

```
    ADD         #01H,R1
    MOV.L       [R4],R3      ; ary
    MOVU.B      R1,R5        ; 発生例のpar+1を誤って符号拡張する
    MOVU.B      [R3,R5],R4   ; ary[]
    CMP         #0AH,R4      ; 終値10と比較
    BLE         L12
    MOV.L       #_S,R5
    MOV.B       R4,[R5]      ; S
L12:
    RTS
```

発生条件B：

以下の(B1)～(B6)をすべて満たす場合に発生します。

(B1) -optimize=0 および -optimize=1 オプションを使用していない。

(B2) char、signed char、または unsigned char 型の配列メンバを持つ構造体または共用体が存在する。

(B3) (B2)の配列メンバの同じ要素を、1つの関数内で2回以上参照する。

(B4) (B3)の配列要素の添え字式は、以下のいずれかである。

- (B4-1) 変数
- (B4-2) 加算式で、かつ一方のオペランドが変数で、もう一方のオペランドが定数
- (B4-3) 減算式で、かつ一方のオペランドが変数で、もう一方のオペランドが定数

注： (B4-2)と(B4-3)の定数は、定数伝播 (const修飾された外部変数の定数伝播を含む) の最適化により変数が定数に置き換わった場合も含む。

(B5) (B4)の変数は、char、signed char、unsigned char、short、signed short、または unsigned short 型である。

(B6) (B3)の配列要素の、(B2)の構造体または共用体先頭からのオフセットが、(B4)の変数の型の表現可能な最大値より大きい値となる。

発生条件Bの発生例:

```
signed char S;

struct {
  signed char dummy[127];
  signed char mem[8];                 // 発生条件(B6)
} *st;                          // 発生条件(B2)

void func(signed char par)              // 発生条件(B5)
{
  if (st->mem[par] > 10) {       // 発生条件(B3)および(B4-1)
    S = st->mem[par];           // 発生条件(B3)および(B4-1)
```

```
  }
  return;
 }
```

上記の発生例では、par が 1 以上 の場合に、発生条件(B6)に該当します。

コンパイル結果：

```
 _func:
    MOV.L      #_st,R4
    ADD        #7FH,R1,R5
    MOV.L      [R4],R3      ; st
    MOVU.B     R5,R5        ; 発生例のpar+127を誤って符号拡張する
    MOVU.B     [R3,R5],R4   ; st->mem[]
    CMP        #0AH,R4      ; 終値10と比較
    BLE        L15
    MOV.L      #_S,R5
    MOV.B      R4,[R5]      ; S
 L15:
    RTS
```

回避策：

以下のいずれかの方法で回避してください。

(1) -optimize=0 または -optimize=1 オプションを使用する。

(2) 以下のいずれかをvolatile修飾する。

- 発生条件(A2)の配列
- 発生条件(B2)の配列メンバ
- 発生条件(B2)の構造体または共用体
- 発生条件(A4)または(B4)の添え字式中の変数

(3) 発生条件(A4)または(B4)の添え字式中の変数を、発生条件(A5)または(B5)以外の型に変更する。

発生例Bの回避例：

```
    void func(signed int par)   // char をsigned int に変更
```
---

(4) 発生条件(B4-1)を満たす場合、添え字式中の変数を発生条件(B5)以外の型にキャストする。

回避例：

```
    st->mem[(signed int)par]   // par を signed int にキャスト
```
---

6. 恒久対策

本内容は、RX ファミリ C/C++コンパイラパッケージ V.1.02 Release 00 で
改修しました。
V.1.02 Release 00の詳細は、RENESAS TOOL NEWS 資料番号 111016/tn3 を
参照ください。以下のURLでも参照できます。(10月20日から公開予定)
https://www.renesas.com/search/keyword-search.html#genre=document&q=111016tn3
