RENESAS ツールニュース

RENESAS TOOL NEWS 2010年08月05日：100805/tn1

RXファミリ用C/C++コンパイラパッケージ V.1.00 Release 00 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 V.1.00 Release 00の使用上の 注意事項 6件を連絡します。

1. autoまたは関数内staticの構造体型変数のメンバに関する注意事項
2. __evenaccessおよびconst修飾子に関する注意事項
3. 1または2バイト長の整数型であるループ制御変数に関する注意事項
4. 整数型同士の型変換に関する注意事項
5. ビットフィールドメンバを持つ構造体に関する注意事項
6. ポインタ間接参照による構造体メンバへの代入に関する注意事項

---

1. autoまたは関数内staticの構造体型変数のメンバに関する注意事項

1.1 内容

記憶クラスがautoまたは関数内staticで、構造体型もしくは配列型変数のメンバを別の変数のアドレスを使って初期化した場合、初期化に使った変数の値を正しく読み出すことができない場合があります。

1.2 発生条件

以下の条件をすべて満たす場合に発生する可能性があります。

(1) -optimize=1, -optimize=2または-optimize=maxを使用している。
(2) 記憶クラスがautoまたは関数内staticである構造体型もしくは配列型変数のメンバを別の変数のアドレス値を使って初期化している。
(3) (2)で初期化しているメンバは、構造体または配列の二番目以降のメンバである。
(4) (3)のメンバを用いて、(2)で初期化に使った変数へ値を代入している。
(5) (4)の代入の後に(3)の初期化で使用した変数の値を参照している。

発生例:

```
------------------------------------------------
// ccrx -cpu=rx600 -optimize=1
#include <stdio.h>
```

```
int x;
typedef struct ST{
 int a;
 int* b;
}St;

void main(){
 St st = {    // 発生条件(2)
  0,
  &x         // 発生条件(3)
 };

 x   = 1;
 *st.b  = 2;   // 発生条件(4)

 if( x == 2 ){ // 発生条件(5)
  //
 }
 else{
  // 発生条件を満たすためelseブロックが実行される
 }
}
-----------------------------------------------
```

1.3 回避策

以下のいずれかの方法で回避してください。

(1) 構造体型変数を関数外で定義して初期化する。

発生例の場合：

```
-----------------------------------------------
int x;
typedef struct ST{
 int a;
 int* b;
}St;

St st = {  //局所変数ではなく大域変数として定義して初期化
 0,
 &x
};
void main(){
 x = 1;
. . . . . .
-----------------------------------------------
```

(2) 構造体メンバまたはメンバ配列の定義と同時に初期化する代わりに
変数のアドレスを代入する。

発生例の場合：

```
-----------------------------------------------
void main(){
  St st;      //構造体型変数定義時に初期化しない
  st.a = 0;
  st.b = &x;  //変数のアドレスを代入
. . . . . . . . . . . . . . . . . . . . . . .
-----------------------------------------------
```

2. __evenaccessおよびconst修飾子に関する注意事項

2.1 内容

宣言時に指定した__evenaccessまたはconst修飾子が無視されることがあります。

2.2 発生条件

以下の(1)～(3)のいずれかの条件を満たすオブジェクトで、それぞれの
条件を満たす__evenaccessまたはconst修飾子の指定が無効になります。

(1) C89またはC99言語でコンパイルされた、__evenaccess修飾子を含んだ型で
宣言された配列。

(2) typedefで定義された型宣言子がconst修飾子を含んでおり、
この型宣言子を用いて宣言された変数。

(3) typedefで定義された型宣言子がconst修飾子を含んでおり、この型宣言子に
constを付加した型で宣言された変数。

発生例1： 発生条件(1)に該当するソース例

```
-----------------------------------------------
__evenaccess char ary1[4];    // 発生条件(1)
void func1(void)
{
 // __evenaccess修飾が無視されるため、
 // 下記のary1[0]～ary1[3]への代入がひとつにまとめられる
 ary1[0] = 0;
 ary1[1] = 1;
 ary1[2] = 2;
 ary1[3] = 3;
}
-----------------------------------------------
```

発生例2： 発生条件(2)および(3)に該当するソース例

```
-----------------------------------------------
```

```
typedef const long CARRAY[2];
CARRAY ary2 = { 0, 1 };         // 発生条件(2)
const CARRAY ary3 = { 2, 3 };   // 発生条件(3)
// ary2とary3は、const修飾が無効になりDセクションに割り付けられる
-------------------------------------------------
```

2.3 回避策

発生条件(1)に該当する場合：
以下のいずれかの方法で回避してください。
(1) C++言語でコンパイルする
(2) 配列要素と同じ型のポインタを経由してアクセスする

発生例1の回避例

```
-------------------------------------------------
__evenaccess char ary1[4];
void func1(void)
{
  __evenaccess char *ptr1 = ary1;
  // ポインタptr1を用いたアクセスに変更
  ptr1[0] = 0;
  ptr1[1] = 1;
  ptr1[2] = 2;
  ptr1[3] = 3;
}
-------------------------------------------------
```

発生条件(2)または(3)に該当する場合：
typedef宣言でconst修飾子を付加せず、変数の宣言で付加してください。

発生例2の回避例

```
-------------------------------------------------
typedef long CARRAY[2];       // 型定義からconstをはずす
const CARRAY ary2 = { 0, 1 }; // 宣言時にconstを付ける
const CARRAY ary3 = { 2, 3 };
-------------------------------------------------
```

3. 1または2バイト長の整数型であるループ制御変数に関する注意事項

3.1 内容

1または2バイト長の整数型であるループ制御変数を関数呼び出しの実引数に
用いると、実引数をスタックに正しく格納しません。

3.2 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を指定している。または、-optimize

オプションを指定していない。
(2) 1または2バイト長の整数であるループ制御変数を持つループがある。
(3) (2)のループ制御変数は、volatile修飾していない。
(4) 引数が複数あって、(2)のループ制御変数を実引数で使用する関数の
呼び出しがあり、ループ制御変数に当たる実引数が以下の(4-1)～(4-4)の
すべての条件を満たしている。
(4-1) 関数原型は、(2)のループ制御変数と互換性のある型である。(注1)
(4-2) 引数はスタック渡しである。(注2)
(4-3) 複数ある引数のうち最後の引数ではない。
(4-4) ループ制御変数に当たる引数およびその右隣りの引数は、
可変個引数ではない。

注1： 整数型で、型サイズと符号のありなしが共に一致している。
注2： 関数引数の割り付け規則は、ユーザーズマニュアル 8.2.3項「引数の
設定、参照に関する規則」を参照してください。

(5) 以下の(5-1)～(5-3)いずれかの条件を少なくともひとつ満たしている。
(5-1) (4)の実引数のスタック上のオフセットが4の倍数ではない。(注2)
(5-2) (4)の実引数のすぐ右の引数は1または2バイト長の整数である。
(5-3) (5-1)および(5-2)をいずれも満たさず、かつ-endian=bigを指定して
いる。

発生例:

```
------------------------------------------------
typedef unsigned char UC;
void func2(int, int, int, int, int, UC, UC, int);
void foo2(void)
{
  UC loop = 0;                  // 発生条件(3)
  while (loop < 8) {            // 発生条件(2)
    func2(1,2,3,4,5,loop,6,7);   // 発生条件(4)および(5-2)
    ++loop;
  }
}
------------------------------------------------
```

発生例の展開結果:

```
------------------------------------------------
関数呼び出しfunc2(1,2,3,4,5,loop,6,7);の部分
  MOV.L  #00000007H,R5
  PUSH.L R5
  SUB    #04H,R0
  MOV.L  #00000005H,R15
  MOV.B  #06H,01H[R0]     ; 第7引数
  MOV.L  R6,[R0]          ; 第7引数の内容を上書き
```

```
  PUSH.L R15
  MOV.L  #00000004H,R4
  MOV.L  #00000003H,R3
  MOV.L  #00000002H,R2
  MOV.L  #00000001H,R1
  BSR    _func2
-------------------------------------------------
```

3.3 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を指定する。

(2) ループ制御変数を4バイト長にする。

例:

```
-------------------------------------------------
void foo2(void)
{
  unsigned long loop = 0; // サイズが4バイト長の型にする
  while (loop < 8) {
・・・・・・・・・・・・・・・・・・・・・・・・・
-------------------------------------------------
```

(3) ループ制御変数をvolatile修飾する。

例:

```
-------------------------------------------------
void foo2(void)
{
  volatile UC loop = 0; // volatile修飾する
  while (loop < 8) {
・・・・・・・・・・・・・・・・・・・・・・・・・
-------------------------------------------------
```

(4) ループ制御変数を型が異なる別の変数に代入し、その変数を実引数として渡す。

(5) ループ制御変数を異なる型にキャストして実引数として渡す。

4. 整数型同士の型変換に関する注意事項

4.1 内容

1バイトと2バイト整数のビットごとの論理積演算(&)の前後でサイズの異なる型変換が行われる場合、ゼロ拡張が正しく行われないことがあります。

4.2 発生条件

以下の条件をすべて満たす場合に発生する可能性があります。

(1) -optimize=2または-optimize=maxを指定している。

(2) -speedを指定していない。

(3) unsigned char型とunsigned short型をそれぞれオペランドとする、
　ビットごとの論理積演算(&)を行っている。
(4) 論理積演算の前後でサイズの異なる型変換が行われる。

発生例:

```
------------------------------------------------
int a;
unsigned short b;
void func()
{
  a = (unsigned char)a & b; // 発生条件(3) および(4)
                            // aをunsigned char型に変換して、
                            // unsigned short型のbとANDを取って
                            // いるので発生条件に該当
}
------------------------------------------------
```

発生例の展開結果:

```
------------------------------------------------
_func:
  MOV.L  #_a,R2
  MOV.L  [R2],R5     ; 変数'a'の値をR5に格納
  MOV.L  #_b,R3
  AND    [R3].UW,R5  ; 論理積演算のオペランドをゼロ拡張して
                     ;いないため上位3バイトの値は誤った値
  MOV.L  R5,[R2]
------------------------------------------------
```

4.3 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を指定する。
(2) -speedを指定する。

5. ビットフィールドメンバを持つ構造体に関する注意事項

5.1 内容

ビットフィールドメンバを持つ構造体をメンバに持つ構造体を戻り値とする関数の呼び出しがあるとき、関数の戻り値から直接ビットフィールドメンバを参照することができません。

5.2 発生条件:

以下の条件をすべて満たす場合に発生する可能性があります。
(1) ビットフィールドをメンバに持つ構造体がある。
(2) (1)の構造体を直接メンバとして持つか、あるいは2重以上にネストした
　構造体のいずれかに(1)の構造体をメンバとして持つ構造体がある。

(3) (2)の構造体を戻り値として持つ関数の呼び出しがある。
(4) (3)の関数呼び出しの戻り値から直接、(1)のビットフィールドメンバを
参照している。

発生例:

```
------------------------------------------------
#include <stdio.h>
typedef struct{ struct{ int a : 8; }st; }ST;  // 発生条件(1)および(2)
ST func(ST st){ return st; }                  // 発生条件(3)
void main(){
 int ret = 0;
 ST St;
 St.st.a = 1;
 ret = func( St ).st.a;     //  発生条件(3)および(4)
                            //  retに正しい値が返されない
 if( ret == 1 ){
  printf( "OK¥n" );
 }
 else{
  printf( "NG(%d)¥n", ret );
 }
}
------------------------------------------------
```

発生例の展開結果:

```
------------------------------------------------
_main:
・・・・・・・・・・・・・・・・・・・・・・・・・
 BSR       _func
 MOV.L     R1,08H[R0]
 MOV.B     08H[R0],R4  ; 間違い。本来はMOVU.B  08H[R0],R5 が正しい。
 MOVU.B  [R4],R5
 CMP       #01H,R5
 BEQ       L12
・・・・・・・・・・・・・・・・・・・・・・・・・
------------------------------------------------
```

5.3 回避策

関数の戻り値をテンポラリの構造体に一時格納してから参照してください。

発生例の場合：

```
  ret = func( St ).st.a;
```

を

```
  ST temp;
  temp = func( St );
```

ret = temp.st.a;
に変更。

6. ポインタ間接参照による構造体メンバへの代入に関する注意事項

6.1 内容

ポインタ間接参照による構造体メンバへの定数値代入と、関数呼び出しの実行順序が入れ替わる場合があります。

6.2 発生条件

以下の条件をすべて満たす場合に発生する可能性があります。
(1) -optimize=2または-optimize=maxを指定している。
(2) 構造体に2つ以上のメンバがあり、そのうち2つは領域が連続している
(3) (2)の領域が連続している2つのメンバは同じサイズの整数型で、かつサイズは1または2バイトである。
(4) (2)の構造体へポインタを使って2つのメンバそれぞれに定数を代入する文がある。
(5) 関数呼び出しがある。
(6) (4)の定数値代入と(5)の関数呼び出しが同じブロック内にある。

発生例:

```
------------------------------------------------
struct STR {              // 発生条件(2)
  unsigned short  member1;  // 発生条件(3)
  unsigned short  member2;  // 発生条件(3)
} s;

void main(struct STR * ps, int flg)
{
  if (flg)
  {
    ps->member1 = 7;        // 発生条件(4)および(6)
    ps->member2 = 16;       // 発生条件(4)および(6)

    func1(ps);              // 発生条件(5)および(6)
    func1(ps);
  }
}
------------------------------------------------
```

発生例の展開結果:

```
------------------------------------------------
_main:
  PUSH.L      R6
  MOV.L       R1,R6
```

```
  CMP       #00H,R2
  BEQ       L12
L11:
  MOV.L     R6,R1
  BSR       _func1
  MOV.L     R6,R1
  BSR       _func1
  MOV.L     #00100007H,[R6] ; ps->member1、ps->member2 への代入結果
                            ; だが、func1()の関数呼び出しの後に、
                            ; 移動しているのでこのコードは正しくない。
  RTSD      #04H,R6-R6
-----------------------------------------------
```

6.3 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を指定する。

(2) 発生条件(2)の構造体、またはメンバをvolatile指定する。

(3) -noscheduleを指定する。

(4) #pragma option を使用して、本問題が発生している関数に-optimize=0、
-optimize=1または-noscheduleのいずれかを指定する。

7. 恒久対策

6件の問題はすべてV.1.00 Release 01で改修済みです。

V.1.00 Release 01の詳細は、RENESAS TOOL NEWS 資料番号100805/tn2を
参照ください。以下のURLでも参照できます。(8月5日から公開予定)

https://www.renesas.com/search/keyword-search.html#genre=document&q=100805tn2