RENESAS Tool News

RENESAS TOOL NEWS on March 1, 2007: 070301/tn4

Notes on Using the C/C++ Compiler Package V.4 through V.6 for the H8SX, H8S, and H8 Families of MCUs

Please take note of the twelve problems described below in using the C/C++ compiler package V.4 through V.6 for the H8SX, H8S, and H8 families of MCUs.

1. Versions Concerned

V.4.0 through V.6.01 Release 02

Product Types

V.4:

PS008CAS4-MWR (Windows edition)
PS008CAS4-SLR (Solaris edition)
PS008CAS4-H7R (HP-UX edition)

V.5:

PS008CAS5-MWR (Windows edition)

V.6:

R0C40008XSW06R (Windows edition)
R0C40008XSS06R (Solaris edition)
R0C40008XSH06R (HP-UX edition)

2. Problems

2.1 Problem 1: With Using the Same Subexpressions (H8C-0057)

Versions Concerned:

V.4.0 through V.4.0.09
V.5.0 through V.5.0.06
V.6.00 Release 00 through V.6.00 Release 03
V.6.01 Release 00 through V.6.01 Release 02

Description:

If the same two or more subexpressions are put in a controlling expression within a function, the destination may become incorrect.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) As a CPU option, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used

(for example, -cpu=H8SXN used in the command line).

(2) An optimizing option is used (no option used or -optimize=1 used in the command line).

(3) The same subexpressions are used twice or more times in one or more controlling expressions in any of the following selection statements or iteration statements within a function.

(a) an if statement

(b) a for statement

(c) a while statement

(d) a do statement

Example:

```
-------------------------------------------------------------------------
long a,b;
long sub(void)
{
    long rc;

    rc= -1;
    if ((a>10) && (b>0)){                // Condition (3)
        rc = 1;
    }
    else {
        if (b>0){                        // Condition (3)
            rc = 0;
        }
    }
    return (rc);
}
-------------------------------------------------------------------------
```

Workaround:

This problem can be avoided in either of the following ways:

(1) Use no optimizing option (use -optimize=0 in the command line).

(2) Use the extending function #pragma option nooptimize for the function concerned.

2.2 Problem 2: With Using the #pragma inline_asm and #pragma interrupt Directives (H8C-0058)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03

V.6.01 Release 00 through V.6.01 Release 02

Description:

If a call is made to a function to which the #pragma inline_asm directive is applied within a function to which #pragma interrupt applied, codes for saving and restoring registers may not be generated.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=2000N used in the command line).

(2) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).

(3) As an Object Type option, Assembly Programs is used (-code=asmcode used in the command line).

(4) A function exists to which #pragma interrupt or __interrupt is applied.

(5) Neither of the following extending functions is applied to the function in (4):

(a) #pragma regsave or __regsave

(b) #pragma asm

(6) A function exists to which #pragma inline_asm is applied.

(7) A call to the function in (6) is made within the function in (4).

(8) The function in (6) has no return value or its return value is not used in the function in (4).

Example:

```
-----------------------------------------------------------------
#pragma inline_asm(sub)                    // Condition (6)
#pragma interrupt(func)                    // Condition (4)
void sub(void)                          // Condition (7)
{
   MOV.W   #1,R0                          // Condition (9)
}
void func(void)
{
   sub();                          // Conditions (5) and (8)
}
-----------------------------------------------------------------
```

Workaround:

This problem can be avoided in any of the following ways:

(1) Apply #pragma regsave or __regsave to the function to which #pragma interrupt has been applied.

(2) Change #pragma inline_asm to __asm and #pragma inline.

(3) Create a function where no functions is expanded inline, and make a call to the function from another to which #pragma interrupt is applied.

2.3 Problem 3: With Expanding memcpy Functions Inline (H8C-0059)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03
V.6.01 Release 00 through V.6.01 Release 02

Description:

If a memcpy function is expanded inline, the number of times of data transfer may be less than specified.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) As a CPU option, H8SXA, H8SXX, or AE5 is used (for example, -cpu=H8SXA used in the command line).

(2) The Inline memcpy/strcpy option is used (-library=intrinsic used in the command line).

(3) A memcpy function is used in the source program and takes a value from 0x60001 to 0x60005 as its third argument.

Example:

```
--------------------------------------------------------------------
#include

char source[0x60001];
char destination[0x60001];

void test(void){
   memcpy(destination, source, 0x60001);           // Condition (3)
}
--------------------------------------------------------------------
```

Workaround:

This problem can be avoided in either of the following ways:

(1) Assign the third argument of the memcpy function to a volatile-qualified variable of type size_t; then use the variable as the third argument of the memcpy function.

Example:

```
---------------------------------------------------------------
#include

char source[0x60001];
char *destination;

void test(void){
   volatile size_t transfer_size = 0x60001;
   memcpy(destination, source, transfer_size);
}
---------------------------------------------------------------
```

(2) Do not use the Inline memcpy/strcpy option
(do not use -library=intrinsic in the command line).

2.4 Problem 4: With Using Identifiers Consisting of 255 Characters or More (H8C-0060)

Versions Concerned:

V.4.0 through V.4.0.09
V.5.0 through V.5.0.06
V.6.00 Release 00 through V.6.00 Release 03
V.6.01 Release 00 through V.6.01 Release 02

Description:

If the number of characters in an identifier (symbol name, section name, or file name) exceeds 244, incorrect objects may be generated.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) The number of characters in an identifier exceeds 244.
(2) The Generate File For Inter-module Optimization option is used
(-goptimize used in the command line).

Workaround:

This problem can be avoided in either of the following ways:

(1) Reduce the number of characters in every identifier to 254 or less.
(2) Do not use the Generate File For Inter-module Optimization option
(do not use -goptimize in the command line).

2.5 Problem 5: With Overflown Operation Concerning a Subscript to an Array (H8C-0061)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03
V.6.01 Release 00 through V.6.01 Release 02

Description:

If a result of operation concerning a subscript to an array is overflown, an incorrect address may be referenced.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=2000N used in the command line).
(2) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).
(3) An array-type variable is defined and declared.
(4) An addition or subtraction operation is performed between a subscript to the array in (3) and a constant; then the result of the operation is converted in type.

(5) Conditions (a) or (b) below is satisfied.
   (a) The type conversion in (4) is the extension to type unsigned long.
   (b) The type conversion in (4) is the extension to type unsigned int or unsigned short with 2000N, 2600N, H8SXN, or H8SXM being used as a CPU option.

(6) The result of the operation in (4) overflows.

Example:

```
------------------------------------------------------------------------
#include

unsigned int a = 10;
unsigned int array[100];                    // Condition (2)

void main(void){
   unsigned int i;

   for (i=0; i<100; i++){
      array[i] = 0;
   }

   array[4] = 1;
   array[0] = array[(unsigned long)(a + 65530u)];
                                  // Conditions (3)--(5)

   if (array[0] == array[4]){
      printf("correct¥n");
   }
}
------------------------------------------------------------------------
```

Workaround:

This problem can be avoided in any of the following ways:

(1) Assign the addition or subtraction operation in Condition (4) to a volatile-qualified variable to use it.

(2) Assign the constant in the addition or subtraction operation in Condition (4) to a volatile-qualified variable to use it.

(3) Modify the addition or subtraction operation in Condition (4) so that it might not overflow.

2.6 Problem 6: With Referencing const-Qualified Members of a Structure or Union (H8C-0062)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03

V.6.01 Release 00 through V.6.01 Release 02

Description:

If structure- or union-type variables qualified to be const are declared in an iteration statement, their members may be incorrectly referenced.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) An optimizing option is used (no option used or -optimize=1 used in the command line).

(2) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).

(3) In a function exists an iteration statement.

(4) In the iteration statement in (3), an assignment is made to a structure-type or union-type variable.

(5) The structure-type or union-type variable or their members in (4) are qualified to be const.

Example:

```
----------------------------------------------------------------------
typedef struct {
    char m1;
    char m2;
} S;

S s;

long func(void){

    long i;
    long val = 0;

    for (i=0; i<2; i++){            // Condition (3)
        const S t = s;           // Conditions (4) and (5)

        val += t.m1;
        val += t.m2;
    }

    return val;
}
----------------------------------------------------------------------
```

Workaround:

This problem can be avoided in any of the following ways:

(1) Use no optimizing option (use -optimize=0 in the command line).

(2) Do not qualify the structure-type or union-type variable or their members to be const.

(3) Make an assignment to the structure-type or union-type variable before the iteration statement.

2.7 Problem 7: With Adding a Volatile-Qualified Variable and a Constant (H8C-0063)

Versions Concerned:

V.6.01 Release 00 through V.6.01 Release 02

Description:

If a volatile-qualified variable and a constant are added, the number of accesses may be different from the one specified.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=2000N used in the command line).

(2) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).

(3) In a function exists an assignment expression that assigns an addition expression to a variable.

(4) The variable in the left term of the assignment expression in (3) is qualified to be volatile.

(5) The variable in (4) is of type unsigned long or signed long.

(6) The addition expression in the right term of the assignment expression in (3) is:
(a) a variable + a constant, or
(b) a constant + a variable.

(7) The variable in (6) is the same as the one in (4).

(8) The constant in (6) is 3, 5, 6, or 8.

Example:

```
---------------------------------------------------------------------
volatile unsigned long a;            // Condition (4)

void main(void){
   a = a + 3;                     // Conditions (3) and (5)--(8)
}
---------------------------------------------------------------------
```

Workaround:

To avoid this problem, assign the constant added to the variable to an external variable; then use this variable in the addition expression.

```
    ---------------------------------------------------------------
    volatile unsigned long a;
```

```
unsigned long b = 3;

void main(void){
   a = a + b;
}
-----------------------------------------------------------------
```

2.8 Problem 8: With Using a Structure-Type Variable of 3 Bytes Wide (H8C-0064)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03
V.6.01 Release 00 through V.6.01 Release 02

Description:

If transferred is a structure-type variable of 3 bytes wide that is a member of a structure-type variable of 4 bytes wide, data in the uppermost byte's area may be overwritten in error.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=2000N used in the command line).

(2) An optimizing option is used (no option used or -optimize=1 used in the command line).

(3) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).

(4) A structure-type variable of 4 bytes wide containing another of 3 bytes wide as a member of it is defined and declared in the source program.

(5) The structure-type variable in (4) is not volatile-qualified.

(6) The 3-byte member in (4) is transferred.

Example:

```
----------------------------------------------------------------------
typedef struct {
   char a[3];
} ST3;

typedef struct {                                    // Condition (4)
   ST3  st3;
   char x;
} ST;

ST3 stg;                                        // Condition (5)
```

```
void sub(ST);

void main(void){
    ST st;
    st.x = 10;
    st.st3 = stg;                                          // Condition (6)
    sub(st);
}
---------------------------------------------------------------------------
```

Workaround:

This problem can be avoided in either of the following ways:

(1) Use no optimizing option (use -optimize=0 in the command line).

(2) Qualify the structure-type variable of 4 bytes wide containing another of 3 bytes wide to be volatile.

2.9 Problem 9: With Performing Logical AND Operations For Each Bit(H8C-0065)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03,
V.6.01 Release 00 through V.6.01 Release 02

Description:

If the result of a logical AND operation for each bit between a variable and a constant is evaluated , the result of evaluation may become incorrect.

Conditions:

This problem may occur if the following conditions are all satisfied:

(1) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=2000N used in the command line).

(2) An optimizing option is used (no option used or -optimize=1 used in the command line).

(3) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).

(4) Condition (a), (b), (c), or (d) below is satisfied.

(a) All the following conditions are met:

- A logical AND operation is performed between a variable and a constant.
- The variable used in the above operation is a parameter of type unsigned long or signed long located in the stack area; to which the evenaccess keyword not added; and not volatile-qualified.
- The constant used in the above operation is equal to or less than 0xFFFF; or its lowermost 2 bytes is 0x0000.

- The result of the operation is compared with 0.
- The above comparison is used only in a conditional expression.

(b) A pointer-type variable is used to increment or decrement.

(c) A variable of type array, for example, references a continuous area.

(d) A parameter is located in the stack area.

Example:

```
---------------------------------------------------------------------
void func(long dummy1, long dummy2, signed long data1)
{
    if ((data1 & 0x00008000) == 0){              // Condition (4)-(a)
        ans1 = 10;
    }else{
        ans1 = 20;
    }
}
---------------------------------------------------------------------
```

Workaround:

This problem can be avoided in either of the following ways:

(1) Use no optimizing option (use -optimize=0 in the command line). Or apply #pragma option nooptimize to the functions where all the above conditions are satisfied.

(2) If Condition (4)-(a) is met, assign the variable used in the logical AND operation to another volatile-qualified variable with the evenaccess keyword; then perform the operation using this variable.

2.10 Problem 10: With Initializing Union-Type Variables (H8C-0066)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03,
V.6.01 Release 00 through V.6.01 Release 02

Description:

If the first member of a 3-byte union-type variable is less than 3 bytes in width, and an initializer is added to the variable, a code assigned to another member of the union is generated.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) As a CPU option, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=H8SXN used in the command line).

(2) In the source program is declared a union-type variable with an initializer.

(3) The union-type variable in (2) is 3 bytes wide.

(4) The first member of the union-type variable in (2) is a 1- or 2-byte variable.

Example:

```
----------------------------------------------------------------------
typedef union {                       // Condition (3)
    char a;                         // Condition (4)
    char b[3];
} UNI;

void sub(UNI);

void func(void){
    volatile UNI uni = {1};             // Condition (2)

    sub(uni);
}
----------------------------------------------------------------------
```

Workaround:

This problem can be avoided in either of the following ways:

(1) Declare and the initialize the union-type variable in different lines.

```
----------------------------------------------------------------------
typedef union {
    char a;
    char b[3];
} UNI;

void sub(UNI);

void func(void){
    volatile UNI uni;
    uni.a = 1;

    sub(uni);
}
----------------------------------------------------------------------
```

(2) Use an expression assigning the address of the union-type variable involved to a pointer-type variable.

```
----------------------------------------------------------------------
typedef union {
    char a;
    char b[3];
} UNI;

void sub(UNI);

void func(void){
   volatile UNI uni = {1};
   volatile UNI *p;
   p = &uni;

   sub(uni);
}
--------------------------------------------------------------------
```

2.11 Problem 11: With Using a Bit Field of 12 Bits Wide (H8C-0067)

Versions Concerned:

V.6.01 Release 00 through V.6.01 Release 02

Description:

If a value is assigned to a member of a structure-type variable defined as a bit field of 12 bits wide in a storage unit, the value of another bit field defined in the same storage unit is overwritten.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) As a CPU option, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=H8SXN used in the command line).

(2) The Optimization for Speed option or its Speed sub-option for arithmetic and comparison operations and assignment expressions is used (-speed or -speed=expression used in the command line).

(3) A structure-type variable is defined and declared.

(4) In the structure-type variable in (3) exists a bit field member of 12 bits wide.

(5) The bit field member in (4) is defined as of type signed int, unsigned int, signed short, or unsigned short.

(6) Before the bit field member in (4) exists a bit field of 1 bit wide, which has the same type as the bit field member in (4).

(7) An assignment is made to the bit field member in (4).

(8) After the assignment in (7), another bit field member in the same storage unit is referenced.

Example:

```
--------------------------------------------------------------------
#include

typedef struct {                        // Condition (3)
   int broken_data1:1;                     // Conditions (5) and (6)
```

```
    int target_data:12;                 // Conditions (4) and (5)
    int broken_data2:3;
} ST;

void main(void){
    ST st;                          // Condition (3)
    st.broken_data1 = -1;
    st.broken_data2 = -1;
    st.target_data  =  0;               // Condition (7)

    if (st.broken_data1 == -1              // Condition (8)
        && st.broken_data2 == -1){
        printf("correct¥n");
    }
}
----------------------------------------------------------------------
```

Workaround:

This problem can be avoided in either of the following ways:

(1) Do not use the Speed sub-option for arithmetic and comparison operations and assignment expressions (do not use -speed=expression in the command line).

(2) Use a value other than 1 bit as the offset of the bit field member of 12 bits wide.

```
-----------------------------------------------------------------
#include

typedef struct {
    int target_data:12;             // Order exchanged
    int broken_data1:1;              // between these two
    int broken_data2:3;
} ST;

void main(void){
    ST st;
    st.broken_data1 = -1;
    st.broken_data2 = -1;
    st.target_data  =  0;

    if (st.broken_data1 == -1
        && st.broken_data2 == -1){
        printf("correct¥n");
    }
}
```

2.12 Problem 12: With Using Embedded Assemble Functions (H8C-0068)

Versions Concerned:

V.6.00 Release 00 through V.6.00 Release 03,
V.6.01 Release 00 through V.6.01 Release 02

Description:

If embedded assemble functions are used, values of constants may be overwritten in an addressing mode with displacement, or variables located in the stack be incorrectly accessed.

Conditions:

A. The above problem may occur if the following conditions are all satisfied:

(a) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=2000N used in the command line).

(b) An address space of 1, 16, or 256 MB is used for the CPU option (:20, :24, or :28 used in the command line).

(c) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).

(d) __asm keyword is used.

(e) Any of the following addressing modes or instruction is used in the compound statement in (d):
- the MOVA instruction
- the register indirect mode with displacement
- the indexed register indirect mode with displacement

(f) A constant value equal to or greater than 0x10000 is used as the value of displacement in (e).

B. Or, the above problem may also occur if the following conditions are all satisfied:

(a) The compiler's version concerned is V.6.01 Release 02.

(b) As a CPU option, 2000N, 2000A, 2600N, 2600A, H8SXN, H8SXM, H8SXA, H8SXX, or AE5 is used (for example, -cpu=2000N used in the command line).

(c) The Ver.4.0 Optimization Technology Generation option is not used (-legacy=v4 not used in the command line).

(d) __asm keyword is used.

(e) A local variable or argument is located in the stack.

(f) In the function in (d), the local variable or argument in (e) is accessed using instructions in the register indirect addressing mode with displacement.

(g) The local variable or argument in (e) has an offset value from the stack pointer other than 0.

Example:

```
-------------------------------------------------------------
void func(void){
   __asm{                              // Condition (1-d)
      mov.l @(0x0010000, er0), er1      // Conditions (1-e) and (1-f)
   }
}
-------------------------------------------------------------
```

Workarounds:

(1) In the case in A above, do not use the MOVA instruction to avoid this problem.

```
---------------------------------------------------------
void func(void){
   __asm{
      mov.l er0, er4
      add.l #0x00010000:32, er4
      mov.l @er4, er1
   }
}
---------------------------------------------------------
```

(2) In the case in B above, this problem can be avoided in either of the following ways:

(a) Change the embedded assemble function from __asm to #pragma asm.

(b) Change the description in the embedded assemble function to a function to which #pragma inline_asm is applied; then make a call to this function.

3. Schedule of Fixing the Problems

We plan to fix all the problems described above in the release of the compiler package V.6.01 Release 03.

[Disclaimer]

The past news contents have been based on information at the time of publication. Now changed or invalid information may be included. The URLs in the Tool News also may be subject to change or become invalid without prior notice.