RENESAS Tool News

RENESAS TOOL NEWS on September 1, 2007: 070901/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 eight problems described below in using the C/C++ compiler package for the H8SX, H8S, and H8 families of MCUs.

1. Product and Versions Concerned

C/C++ compiler package for the H8SX, H8S, and H8 families
V.4.0 through V.6.01 Release 03

2. Problems

2.1 With Casting Addresses to Be Referenced to the Volatile-Qualified Pointer Type (H8C-0069)

Versions Concerned:

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

Symptom:

If the referenced address of a variable is cast to the volatile-qualified pointer type, the access to the variable may be removed.

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 selected (for example, -cpu=2000n typed in the command line).
However, if the Ver.4.0 Optimization Technology Generation option is selected (-legacy=v4 typed in the command line), or the product of V.6.00 is used, 2000N, 2000A, 2600N, and 2600A are not involved in this problem.

(2) Optimization is used (-optimize=1 is typed in the command line; this option is valid by default).

(3) A local variable of any type except structure exists, and the

address of the variable is referenced.

(4) In the program exist two or more assignment expressions in each of which the referenced address of the variable in (3) is cast to *(volatile *). Here, is the same as the type of the variable in (3).

Example:

```
---------------------------------------------------------
void main(void){
   int X;
   *(volatile int *)(&X) = 0;  /* Conditions (3) and (4) */
   *(volatile int *)(&X) = 0;  /* Conditions (3) and (4) */
}
---------------------------------------------------------
 _main:
      RTS     ; No code of assignment expression generated.
;
---------------------------------------------------------
```

Workarounds:

Avoid this problem in any of the following methods:

(1) Use no optimization (type -optimize=0 in the command line).

(2) Apply the #pragma option nooptimize directive to the function involved.

(3) Replace the local variable in Condition (3) with a member of a structure, and reference the address of the structure instead of referencing that of the local variable.

Example Method (3) used:

```
---------------------------------------------------------
  struct {
     int X;
  } A;
  *(volatile int *)(&A) = 0;
---------------------------------------------------------
```

2.2 With Using include Function tas (H8C-0070)

Versions Concerned:

V.6.01 Release 00 through V.6.01 Release 03

Symptom:

Include function tas may incorrectly be performed.

Conditions:

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

(1) As a CPU option, 2000N, 2000A, 2600N, or 2600A, is selected

(for example, -cpu=2000n typed in the command line).
(2) The Ver.4.0 Optimization Technology Generation option is not selected (-legacy=v4 not typed in the command line).
(3) Include function tas() is used.

Example:

```
------------------------------------------------------------
#include <machine.h>
void test(long a, long b, char *p)
{
. . . . . .

   tas(p);     /* Condition (3) */
. . . . . .
}
------------------------------------------------------------
 _test:
. . . . . .
      TAS     @ER2   ; Register that cannot use TAS is
                     ; assigned to it.
. . . . . .
      RTS
------------------------------------------------------------
```

Workarounds:

Avoid this problem in either of the following methods:
(1) Use the Ver.4.0 Optimization Technology Generation option (type -legacy=v4 in the command line).
(2) Assign the register that can use the TAS instruction to it using the __asm keyword.

Example Method (2) used:

```
------------------------------------------------------------
void test(long a, long b, char *p)
{
    __asm {
      MOV.L @(p,sp), ER4
      TAS   @ER4
    }
}
------------------------------------------------------------
```

(3) Issue #pragma inline_asm; then use the TAS instruction.

Example Method (3) used:

```
-------------------------------------------------------
#pragma inline_asm tas2
static void tas2(char *p)
{
   TAS @ER0
}
void test(long a, long b, char *p)
{
   tas2(p); // Assembly function specified by #pragma inline_asm

}
-------------------------------------------------------
```

2.3 With Initial Values of Variables when the Inter-file inline expansion Option Used (H8C-0071)

Versions Concerned:

V.6.01 Release 00 through V.6.01 Release 03

Symptom:

When the definition of a variable exist in a file specified by the Inter-file inline expansion option, and if the variable is used as an initial value of another variable, the initial value may not be set.

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 selected (for example, -cpu=2000n typed in the command line).
However, if the Ver.4.0 Optimization Technology Generation option is selected (-legacy=v4 typed in the command line), 2000N, 2000A, 2600N, and 2600A are not involved in this problem.

(2) The Inter-file inline expansion option is selected (-file_inline typed in the command line).

(3) A global variable is used as the initial value of another.

(4) The global variable used as the initial value in (3) is declared to be extern.

(5) The global variable in (4) is defined in the file specified by the Inter-file inline expansion option and used only as the initial value in (3).

Example:

```
-------------------------------------------------------
 <test1.c>
```

```
  extern int aa;       /* Condition (4)    */
  const int *a = &aa;  /* Conditions (3) and (5) */
  <test2.c>
  int aa;              /* Condition (5)    */

  <Command line>
  ch38 -cpu=2600n -file_inline=test2.c  test1.c
-------------------------------------------------------------
    .EXPORT    _a
    .SECTION   D,DATA,ALIGN=2
  _a:          ; static: a
    .DATA.L    _aa  ; Instruction controlling import of symbol aa
               ; not generated.
    .END
-------------------------------------------------------------
```

Workarounds:

Avoid this problem in either of the following methods:

(1) Define a global variable in the same file in which it is used as the initial value of another.

Modification of the above example:

```
  <test1.c>
  int aa;
  const int *a = &aa;
```

(2) Do not use the Inter-file inline expansion option (do not type -file_inline in the command line).

2.4 With Referencing Members of a Structure in the Inside and Outside of __asm{ } (H8C-0072)

Versions Concerned:

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

Symptom:

If a local variable or argument of a structure type is declared, and members of the structure are referenced in the inside and outside of __asm{ }, incorrect addresses may be accessed for the members referenced in the outside of __asm{ }.

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 selected (for example, -cpu=2000n typed in the

command line).
(2) Optimization is used (-optimize=1 is typed in the command line; this option is valid by default).
(3) The Ver.4.0 Optimization Technology Generation option is not selected (-legacy=v4 not typed in the command line).
(4) A function containing __asm{ } is defined.
(5) A structure-type local variable is defined, or a structure-type argument is declared in the function in (4).
(6) The structure-type local variable in (5) is not saved at the top of the stack used by the function in (4).
(7) Any member of the structure in (5) is referenced in the inside of __asm{ }.
(8) Any member of the structure-type local variable including the member referenced in (7) is referenced in the outside of __asm{ }.

Example:

```
------------------------------------------------------------
struct st{
   long a;

   long b;
};
long func(){
    long l;                        /* Top of stack   */

    struct st str2 = {1L,2L};

   __asm{                          /* Condition (4)   */
      mov.l  @(str2.a :32, sp ) , ER0  /* Condition (7)   */
      mov.l  ER0, @(l:32,sp)
   }
   return l+str2.b;                 /* Condition (8)   */

}
------------------------------------------------------------
_func:
      PUSH.L     ER2
      SUB.W      #H'000C:16,R7
      MOV.L      SP,ER1
      ADDS.L     #4,ER1
      MOV.L      #L28,ER0
      SUB.L      ER2,ER2
      MOV.B      #8:8,R2L
```

```
    JSR       @$MVN$3:24
    MOV.L     @(4:32,SP),ER0
    MOV.L     ER0,@(0:32,SP)
    MOV.L     @SP,ER0
    MOV.L     @(4:16,SP),ER1 ; Incorrect area referenced.
                             ; Should be MOV.L @(8:16,SP),ER1
    ADD.L     ER1,ER0
    ADD.W      #H'000C:16,R7
    POP.L     ER2
    RTS
```

Workarounds:

Avoid this problem in any of the following methods:

(1) Use no optimization (type -optimize=0 in the command line).

(2) Apply the #pragma option nooptimize directive to the function involved.

(3) Assign the value of a member referenced in the inside of __asm{ } to a variable in the outside of __asm{ }; then reference the variable in the inside of __asm{ }.

Example Method (3) used:

```
---------------------------------------------------------
}
struct st{
  long a;
  long b;
};
long func(){
  long l;
  struct st str2 = {1L,2L};
  long ll = 0;        /* Variable to which assignment made */
  ll = str2.a;        /* Assignment made outside __asm{ } */
  __asm{
    mov.l  @(ll, sp ) , ER0 ; Value of variable to which
                            ; assignment made outside
                            ; __asm{ } is referenced.
    mov.l  ER0, @(l:32,sp)
  }
  str.a = ll;
  return l+str2.b;
}
---------------------------------------------------------
```

2.5 With Initializing an Array Having 0x8000 or More Elements in a Class of C++ (H8C-0073)

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 03

Symptom:

If an array consisting of 0x8000 or more elements is specified in a class having a constructor of C++, some elements of the array cannot be initialized by the constructor.

Conditions:

This problem occurs if the following conditions are all satisfied:

(1) As a CPU option, 300HA, 2000A, 2600A, H8SXA, H8SXX, or AE5 is selected (for example, -cpu=300ha typed in the command line).
(2) A class having a constructor is defined.
(3) A class-type array consisting of 0x8000 or more elements is defined.
(4) Either of the following warning messages is dispatched during compilation:
(a) C5068 (W) Integer conversion resulted in a change of sign
(b) C5069 (W) Integer conversion resulted in truncation

Example:

```
------------------------------------------------------------
class C {
 public:
   C();         /* Condition (2) */
};
C c[0x8000];     /* Condition (3) */
------------------------------------------------------------
```

Workaround:

To avoid this problem, use a template of class to initialize the elements in units of 0x7FFF or less.

Example:

```
------------------------------------------------------------
  class C {
  public:
     C();
  };
  template<class T>
```

```
class Array32K {
   T a1[0x4000];
   T a2[0x4000];
public:
   T& operator [] (size_t i)
   {
      if (i < 0x4000)
         return a1[i];
      else
         return a2[i - 0x4000];
   }
};
Array32K<C> c;     // Equivalent to C c[0x8000];
```

2.6 With Using a Function Containing Two or More Expressions Dealing with a Constant (H8C-0074)

Version Concerned:
V.6.01 Release 03

Symptom:

If two or more expressions dealing with a constant exist in a function, the program may not properly be executed.

Conditions:

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

(1) Optimization is used (-optimize=1 is typed in the command line; this option is valid by default).

(2) In a function exist two or more expressions meeting either of the following conditions:

(a) These expressions contain a 4-byte constant each, and the value of the upper or lower 2 bytes or more of a constant is the same as that of the others.

(b) These expressions contain a 2-byte constant each, and the value of the 2 bytes or the upper or lower byte of a constant is the same as that of the others.

Example:

```
---------------------------------------------------------
//-cpu=2600a -legacy=v4 -sp
long g;
f1()
{
   g = (long)0x07FFFE01;        /* Condition (2) */
```

```
    if(g != (long)0x07FFFE01){     /* Condition (2) */
      printf("g=%08lX : ", (unsigned long)g); return(FALSE);
    }
    return(TRUE);
}
------------------------------------------------------------
MOV.L      #134217217,ER0
MOV.L      ER0,@_g:32
SUB.B      R0H,R0H       ; These outputted in error.
RTS
MOV.L      #134217217,ER0
MOV.L      ER0,@_g:32
SUB.B      R0H,R0H
BNE        L23:8
. . . . . .
------------------------------------------------------------
```

Workarounds:

Avoid this problem in either of the following methods:

(1) Use no optimization (type -optimize=0 in the command line).

(2) Apply the #pragma option nooptimize directive to the function involved.

2.7 With Using Functions Returning Values of Type Structure, Union, or Class (H8C-0075)

Versions Concerned:

V.6.01 Release 02 through V.6.01 Release 03

Symptom:

Functions that return values of type structure, union, or class may not properly be performed.

Conditions:

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

(1) As a CPU option, 2000N, 2000A, 2600N, or 2600A, is selected (for example, -cpu=2000n typed in the command line).

(2) The Ver.4.0 Optimization Technology Generation option is selected (-legacy=v4 typed in the command line).

(3) No optimization is used (-optimize=0 is typed in the command line).

(4) The return value of a function is of type structure, union, or class.

Example:

```
------------------------------------------------------------
struct Test {
```

```
  char  cc[2];
} test;

struct Test func(void) {
  struct Test a;
  a.cc[0] = 1;
  return(a);          /* Condition (4) */
}
------------------------------------------------------------
_func:
    PUSH.L  ER6
    MOV.L   SP,ER6
    PUSH.L  ER2
    SUBS    #2,SP
    MOV.B   #1:8,R0L
    MOV.B   R0L,@(-6:16,ER6)
    MOV.L   ER6,ER0
    ADD.W    #H'FFFA:16,R0
    MOV.L   @(4:16,ER6),ER1  ; Incorrect area referenced.
                        ; Should be MOV.L @(8:16,ER6),ER1
    MOV.L   #2:32,ER2
    JSR     @$MVN$3:24
    BRA     P_0000002A:8
P_0000002a:
    ADDS    #2,SP
    POP.L   ER2
    POP.L   ER6
    RTS
------------------------------------------------------------
```

Workarounds:

Avoid this problem in any of the following methods:

(1) Use optimization (type -optimize=1 in the command line; this option is valid by default).

(2) Apply the #pragma option optimize directive to the function involved.

(3) Use the option "Register Allocation of Structure Parameters" (type -structreg in the command line) if the return value is 4 bytes or less in size.

(4) Pass the return value using a pointer.

Example Method (4) used:

```
------------------------------------------------------
struct Test {
```

```
  char  cc[2];
} test;

struct Test* func(void) {
  struct Test a;
  a.cc[0] = 1;
  return(&a);
}
```

2.8 With Using a Structure or Union Type of 3 Bytes in Size (H8C-0076)

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 03

Symptom:

Using a structure- or union-type variable of 3 bytes in size as an argument/parameter of a function may cause incorrect results.

Conditions:

This problem may occur if the following conditions are all satisfied:
(1) As a CPU option, 300HN, 300HA, 2000N, 2000A, 2600N, or 2600A is selected (for example, -cpu=300hn typed in the command line). Or When 2000N, 2000A, 2600N, or 2600A is selected in V.6.01, the Ver.4.0 Optimization Technology Generation option is selected (-legacy=v4 typed in the command line) at the same time.
(2) The option "Register Allocation of Structure Parameters" (-structreg typed in the command line) is selected.
(3) Declared is a structure- or union-type variable that is 3 bytes in size and has a boundary adjustment of 1.
(4) A function takes the structure- or union-type variable in (3) as an argument.
(5) Any register except (E)R0 is assigned to the argument in (4).
(6) The address of the argument in (4) is referenced within the function in (4).

Example:

```
struct tmp1 {                   /* Condition (3)     */
  char ta;
  char ts;
  char tt;
```

```
};
char c;
void func(char a1, struct tmp1 pa) {  /* Conditions (4) and (5) */
   a1++;
   pa.ts++;
   do {
      c=f2(&a1);
      c=f2(&pa.ta);              /* Condition (6)     */
   } while(c);
}
-------------------------------------------------------------
 _func:
    PUSH.L  ER6
    SUBS    #4,SP
    MOV.B   R0L,@(3:16,SP)
    MOV.L   ER1,@SP      ; This 4-byte transfer instruction
                         ; corrupts the result obtained by
                         ; the preceding one.
    MOV.L   #_C:32,ER6
    MOV.B   @(3:16,SP),R0L
    INC.B   R0L
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
-------------------------------------------------------------
```

Workarounds:

Avoid this problem in any of the following methods:

(1) Do not use the option "Register Allocation of Structure Parameters"
(do not type -structreg in the command line).

(2) Use a structure or union whose size is 4 bytes.

(3) Change the order of the arguments of the function so that
the (E)R0 register can be assigned to the structure- or union-
type argument.
Example Method (3) used:

```
-------------------------------------------------------------
 void func(struct tmp1 pa, char a1){
   a1++;
   pa.ts++;
   do {
      c=f2(&a1);
      c=f2(&pa.ta);
   } while(c);
 }
-------------------------------------------------------------
```

3. Schedule of Fixing the Problems

We plan to fix these problems in the product of V.6.02 Release 00 will be released on September 5.
