RENESAS Tool News

RENESAS TOOL NEWS on September 1, 2010: 100901/tn1

Notes on Using the C/C++ Compiler Package V.9s for the SuperH RISC engine Family of MCUs

When you use the C/C++ compiler package V.9s for the SuperH RISC engine family of MCUs, take note of the following problems:

- With expanding many called functions inline (SHC-0081)
- With using the linker options data_stuff and nooptimize (LNK-0006)

---

1. Product and Versions Concerned

The C/C++ compiler package for the SuperH RISC engine MCU family
V.9.00 Release 00 through V.9.03 Release 02

2. Problem in the C/C++ Compiler

2.1 Problem with Expanding Many Called Functions Inline (SHC-0081)

Description:
If many called functions are expanded inline, incorrect code may be generated. (See NOTE below.)

NOTE:
In the V.9.02 and earlier products, memory consumption is restricted in inline expansion, so this problem arises with a lower probability than in the V.9.03 and later products.

Example:
In the C/C++ compiler V.9.03 Release 02, we have observed this problem to arise in the program that satisfy the following:

Number of source lines:	about 12,000
Number of functions:	about 600
Number of execution of inline expansion:	about 150

Conditions:
This problem may arise if either of the following conditions is satisfied:

(1) The inline=value option is used, and 1 or a greater values is selected as the value. (See NOTES 1 and 2 below.)

NOTES:

1. If you use the inline=value option together with the speed option, inline=20 is the default.
2. If you use the inline=value option together with the nospeed or the size,optimize=0 or the optimize=debug_only option, noinline (the same as inline=0) is the default.

(2) In the program exist the functions declared by using #pragma inline.

Workaround:

To avoid this problem, use the -noinline option and do not use #pragma inline.

You may also avoid this problem by reducing the number of called functions to be expanded inline. To do so, adopt any of the following methods:

(1) Use a smaller value in the inline=value option. However, this method has no effect on the functions declared by using #pragma inline.
(2) Do not use the file_inline option.
(3) Reduce the number of functions declared by using #pragma inline.
(4) Do not use the noscope option.
(5) Split the file into two or more to reduce the number of functions in one file. As a result, the number of functions to be expanded inline can be reduced.

3. Problem in the Optimizing Linkage Editor

3.1 Problem with Using the Linker Options data_stuff and nooptimize (LNK-0006)

Description:

If the linker options data_stuff and nooptimize are selected, two or more symbols may be stored in the same address.

Conditions:

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

(1) The CPU type is any member of the SuperH family.
(2) The linker option data_stuff is selected at linking.
(3) The linker option nooptimize is selected at linking.
(4) The compiler option code=machinecode is selected at compilation, and under this condition the .obj file that has directly been generated is inputted to the optimizing linkage editor.
(5) In the .obj file in (4) exists a data section.
(6) In the data section in (5) are contained all the symbols of the

following sizes:
- Odd-numbered bytes (including 1 byte)
- 2 bytes
- Multiples of 4 bytes

(7) In the data section in (5) are contiguously stored an odd number of symbols of odd-numbered bytes.

(8) Immediately after (7) is stored a symbol with 2 bytes long.

(9) In a smaller address area than that occupied by the odd-numbered symbols contiguously stored in (7), the following conditions are both satisfied:
(9-a) In this area exists a symbol of 2 bytes long.
(9-b) In this area exists no symbol of a multiple of 4 bytes long.

(10) In a larger address than that occupied by the symbol of 2 bytes long in (9-a), a symbol of a multiple of 4 bytes long is stored.

(11) Two or more .obj files other than the .obj file in (4) are inputted to the optimizing linkage editor.

(12) In at least one .obj file in (11) exists a data section with the same name as in the .obj file in (4).

Examples:

Source code:

```
-----------------------------------------------------------------
-----data.c-------
// Condition (5)
char  char1;   // Condition (6)
char  char2;
char  char3;
char  char4;
short short1;  // Conditions (6) and (9-a)
char  char5;   // Conditions (7) and (9-b)
short short2;  // Condition (8)
char  char6;
char  char7;
char  char8;
long  long1;   // Conditions (6) and (10)

-----dummy.c-------
// Condition (12)
long dummy;
-----------------------------------------------------------------
```

Command line:

```
-----------------------------------------------------------------
// Conditions (1), (2), (3), (4), and (11)
shc  dummy.c data.c
```

```
optlnk  dummy.obj data.obj -data_stuff -nooptimize
-----------------------------------------------------------------
```

Result of compilation (linkage list file):

```
-----------------------------------------------------------------
; Symbols _char8 and _long1 are stored in address 0x00000010.

. . . . . . . . . . . . . . . . . . . . . . . . . . . .

FILE=data.obj
              00000008  0000000f       8

. . . . . . . . . . . . . . . . . . . . . . . . . . . .

 _char7
              0000000f       1   data ,g       0
 _char8
              00000010       1   data ,g       0
FILE=data.obj
              00000010  00000013       4
 _long1
              00000010       4   data ,g       0
-----------------------------------------------------------------
```

Workarounds:

To avoid this problem, use either of the following ways:

(1) Do not select the linker option data_stuff at linking.

(2) Generate the .obj file described in Condition (4) from the C/C++ source file by using the following procedure:

(a) Use the compiler option code=asmcode at compilation, and generate an assembler source program.

(b) Then assemble the assembler source program to generate the .obj file.

4. Schedule of Fixing the Problems

We plan to fix these problems in the C/C++ compiler package V.9.04 Release 00 for the SuperH RISC engine family of MCUs.