RENESAS Tool News

RENESAS TOOL NEWS on March 16, 2005: RSO-H8C-050316D

The C/C++ Compiler Package for the H8, H8S, and H8SX Families Revised to V.6.01 Release 00

We have revised the C/C++ compiler package for the H8, H8S, and H8SX families of MCUs from V.6.00 Release 03 to V.6.01 Release 00.

---

1. Descriptions of Revision

1.1 Functions Introduced and Improved

1.1.1 In the High-performance Embedded Workshop (Windows Version Only)

The High-performance Embedded Workshop updated to V.4.00.00.
For details, see RENESAS TOOL NEWS "The High-performance Embedded Workshop, an Integrated Development Environment, Revised to Its V.4.00.00" (No. RSO-HEW-050126D), issued on January 26, 2005.

1.1.2 In Simulator Debuggers (Windows Versions Only)

(1) In the simulator debugger for the H8SX series, the FETCHMD bit introduced to the SYSCR register. This bit enables you to select a fetch size out of 16 and 32 bits as in real silicons.

(2) In the simulator debugger for the H8S series, you are able to simulate the 16-bit timer pulse unit functions. For details, see Section 2, "Support for Timers (H8S Series)," in "Supplementary Information on H8S, H8/300 Series High-performance Embedded Workshop 3 User's Manual."

(3) In the simulator debugger for the H8S/2000 series, an interrupt mode was supported.

(4) Memory resources can automatically be reserved at downloading programs to simulator debuggers.

(5) If a memory access error arises, the address where the

error detected can be displayed.

1.1.3 In Compilers

(1) The AE-5 series of MPUs supported.

(2) When any CPU in the H8S series compiled, code can be generated using the same optimizations technology as in the H8SX series.

(3) The following items interpreted in conformance with ANSI:

1. Indexing in arrays

   Example:
   ```
   ----------------------------------------------------------
   ----
   int iarray[10], i=3;
   i[iarray] = 0;  /* Interpreted as iarray[i] = 0; */
       /* Hitherto interpreted as the C2200 (E)
                                 and C2233 (E) errors */
   ----------------------------------------------------------
   ----
   ```

2. Specifying bit fields in unions

   Example:
   ```
   ----------------------------------------------------------
   ----
   union u {
     int a:3;  /* Hitherto interpreted as the C2200 (E) error
   */
   };
   ----------------------------------------------------------
   ----
   ```

3. Operations between constants

   Example:
   ```
   ----------------------------------------------------------
   ----
   static int i=1||2/0;  /* Hitherto interpreted as
                                 the C2501 (E) error */
   ----------------------------------------------------------
   ----
   ```

(4) Library function strtoul and macro FOPEN_MAX in

conformance with ANSI introduced.

(5) The following options added:
   1. strict_ansi: Interprets the associative rule in floating-point operations in conformance with ANSI. When this option used, results of operations may differ from those in Ver.6.0.
   2. enable_register: Assigns registers to variables specified by the register storage-class specifier, with the highest priority.
   3. legacy=v4: When any MPU in the H8S series compiled, generates code using the same optimizations as in Ver.6.00 or earlier.

(6) Directive #pragma address introduced, which places a variable at an absolute address.

(7) The following limiting values raised (for the H8SX and H8S series of MPUs with the legacy=v4 option not selected):
   1. The maximum of nesting levels when iteration statements (while, do, or for) and selection statements (if or switch) are combined: from 32 to 4096.
   2. The maximum number of goto labels usable in a function: from 511 to 2147483646.
   3. The maximum of nesting levels in a switch statement: from 256 to 2048.
   4. The maximum number of case labels usable in s switch statement: from 511 to 2147483646.
   5. The maximum number of arguments usable in a function definition or a function call: from 63 to 2147483646.

1.1.4 In Assemblers

(1) The AE-5 series of MPUs supported.

(2) Assembler directive command .STACK supported.
This command, when put in an assembler source file to declare a stack size, allows the stack-analyzing tool to read the size automatically.

(3) The maximum number of characters of a replacement symbol in the DEFINE option and the .define assembler directive command raised from 32 to an unlimited number.

(4) All the source lines outputted in a listing file started as

new lines.

1.1.5 Optimizing Linkage Editors

The following functions introduced:

(1) Setting boundary-adjusting numbers for the specified sections
Boundary-adjusting numbers can be set for the sections selected by the binary option.

(2) Cross-referencing
By using the show=xreference option, cross-reference information that informs you from where variables and functions are referred to can be output to a linkage list.

(3) Signaling the existence of un-referred symbols
Option msg_unused has been introduced. This option is used to send an information message if any externally defined symbol that has not yet been referred to exists.

1.2 Problems Fixed

1.2.1 In the High-performance Embedded Workshop (Windows Version Only)

The following problem has been fixed: Consider the case where projects have been created in Normal mode for any CPU of the H8S/2600, H8S/2000, or H8/300H series, and standard I/O file file lowlvl.src generated. When this file is used, an I/O simulation error (with error message System Call Error) arises.

When you build a program containing the lowlvl.src file created in the same manner as stated above using an earlier version of the compiler package and then simulate the program, you may encounter the "System Call Error" message. If so, create a new project to re-create the lowlvl.src file.

1.2.2 In Compilers

The following 14 problems have been fixed:

(1) On performing operations on variables of type int with overflow (H8C-0004)
In the versions 6.00 Release 00 through 03, performing operations on variables of type int with overflows may bring incorrect results.

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

1. Out of the CPU options, H8SXA or H8SXX (-

cpu=h8sxa or h8sxx) is selected.
2. Out of the optimizing options, -optimize=1 is selected.
3. A variable of type int or short is added, and then the result is multiplied by a variable of type unsigned long.
4. The add operation in Condition 3 results in an overflow.

Example:

```
------------------------------------------------------------
signed int sub(){
   return 32767;
}
unsigned long ul1,ul2;
main(){
   signed int i1,i2;
   i1 = sub();
   ul1 = 2;
   i2 = 1;
   ul2 = ul1 * (i1 + i2); // ul2=2*(unsigned
long)(32767+1)
}
------------------------------------------------------------
```

(2) On accessing members of structures when the size of a structure-type array exceeds 32767 bytes (H8C-0005)
In the versions 6.00 Release 00 through 03, incorrect data may be accessed if a member of a structure in a structure-type array whose size exceeds 32767 bytes is referenced.

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

- Out of the CPU options, H8SXN, H8SXM, H8SXA or H8SXX (-cpu=h8sxn, h8sxm, h8sxa, or h8sxx) is selected.
- The sizes of all the structures that are the elements of an array are 2 or 4 bytes.
- The size of the array in Condition 2 exceeds 32767

bytes.

Example:

```
---------------------------------------------------------
-------
struct st2{
   char a;
   char b;
}st_2[32767];
void main(void){
   char i;
   for(i = 0 ; i < 10 ; i++){
      st_2[i].b = i;
         // This value is set at an area different from the
                    specified area in the structure.
   }
}
---------------------------------------------------------
-------
```

(3) On accessing an array in an iteration statement (H8C-0006)
In the versions 4.0 through 5.0.06, incorrect elements may be accessed if elements of an array is referenced in an iteration statement.

Conditions:
A. Any CPU of the 300HA, 2000HA, and 2600HA series selected This problem may occur if the following conditions are all satisfied:
1. Out of the CPU options, 300HA, 2000HA, or 2600HA (-cpu=300ha, 2000a, or 2600a) is selected.
2. Out of the optimizing and speed options, -optimize=1; and -speed and -speed=loop[1|2] are selected.
3. A variable of type unsigned short or unsigned int is declared.
4. The variable in Condition 3 is used in the following manner:
   - Initialized to a constant when the controlled variable in a loop is initialized.
   - Used as the controlled variable in a loop.

Used as a suffix to an array such a way as a constant minus a variable.

B. Any CPU of the H8SXX and H8SXA series selected
This problem may occur if the following conditions are all satisfied:

1. Out of the CPU options, H8SXX or H8SXA (-cpu=h8sxx or h8sxa) is selected.
2. Out of the Pointer Size Specification options, -ptr16 is selected.
3. A variable of type unsigned short or unsigned int is declared.
4. The variable in Condition 3 is used as a suffix to an array such a way as a constant minus a variable or a constant plus an expression, where the expression is negative.

Example 1:

```
----------------------------------------------------------
-------
extern unsigned char a[20];
void sub(void){
   unsigned short j;
   for(j=0; j<10; j++){
      a[20-j] = 10; // References the outside of the
array area.
   }
}
----------------------------------------------------------
-------
```

Example 2:

```
----------------------------------------------------------
-------
unsigned short a[20];
unsigned short j;
void sub3(void){
   a[20-j] = 10;
}
----------------------------------------------------------
-------
```

(4) On expanding a switch statement to a table (H8C-0007)
In the versions 6.00 Release 00 through 03, if a variable is set to the same value inside and outside a switch

statement expanded to a table, the value may become indefinite.

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

1. Out of the CPU options, H8SXX or H8SXA (-cpu=h8sxx or h8sxa) is selected.
2. There exist codes automatically expanded to a table by selecting Auto (-case=auto) out of the Switch statement output code selection method options. Or Table output code selection method (-case=table) is selected.
3. The variable to be set is an array of type char, unsigned char, short, unsigned short, int or unsigned int.
4. Several constants with the same value are used before and after a switch statement.

(5) On passing arguments to a function that takes a variable number of arguments (H8C-0008)
In the versions 6.00 Release 00 through 03, if the Register allocation of structure parameters option (-structreg) is selected, and a structure that is 4 bytes or less in size is used as an argument to a function, the argument to be passed via the stack may be passed via a register.

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

1. Out of the CPU options, H8SXN, H8SXM, H8SXA or H8SXX (-cpu=h8sxn, h8sxm, h8sxa or h8sxx) is selected.
2. The Register allocation of structure parameters option (-structreg) is selected.
3. There exists a function that takes a variable number of arguments, which include structure-type variables that are 4 bytes or less in size.
4. The function in Condition 3 is declared to be one with ellipsis, and the structure-type variables stated there are passed to the function via the stack.
5. In the registers assigned to arguments, there is

available space for storing the structures in Condition 3.

Example:

```
--------------------------------------------------------
-------
struct A{
   int is_keyword ;
} flags;
void sub(const char *,...);
void main(void){
   sub("test", flags); // Copies the contents of the flags.
}
--------------------------------------------------------
-------
```

(6) On generating incorrect section names at expanding a switch statement to a table (H8C-0009)
In the versions 4.0 through 5.0.06, if a switch statement is expanded to a table with the Short absolute address option being selected, incorrect section names may be generated.

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

1. Out of the CPU options, 300HN, 2000N or 2600N (-cpu=300hn, 2000n, or 2600n) is selected.
2. Out of the Short absolute addressing mode options, -abs16 is selected.
3. There exist codes automatically expanded to a table by selecting Auto (-case=auto) out of the Switch statement output code selection methodoptions. Or Table output code selection method (-case=table) is selected.
4. Out of the Object type options, the Machine code option (-code=machinecode) is selected.

Example:

```
--------------------------------------------------------
-------
char c;
void func(void){
   switch (c) {
        case 0:
          c-=2;
          break;
        case 1:
          c--;
          break;
        case 2:
          c++;
          break;
        case 3:
          c+=2;
          break;
        case 4:
          c+=3;
          break;
        case 5:
          c+=4;
          break;
      }
    }
    --------------------------------------------------------
    -------
```

(7) On performing multiply operations when speed has a higher priority (H8C-0010)

In the versions 4.0 through 5.0.06, if a multiply operation is performed between a variable of type long and that of type int, incorrect objects may be generated or internal errors may arise.

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

1. Out of the CPU options, 300HN, 300HA, 2000N, 2000A, 2600N, or 2600A (-cpu=300hn, 300ha, 2000n, 2000a, 2600n, or 2600a) is selected.
2. The Optimization for Speed option (-speed); or the one for arithmetic and comparison operations and assignment expressions (-speed=expression) is selected.
3. A multiply operation between a variable of type long and that of type short or int is performed.
4. Both the multiplicand and multiplier are function

calls or operational expressions.

Example:

```
--------------------------------------------------------------
-------
long  l;
short func_s(void);
long  func_l(void);
void func(void){
  l = func_s() * func_l();
     // A multiplication between a variable of type short
                                              and that of type long
}
--------------------------------------------------------------
-------
```

(8) On using assembler directive command DATA within the __asm{} function (H8C-0011)
In the versions 6.00 Release 00 through 03, if a variable is defined using .DATA.B or .DATA.W within the __asm{} function, and a value greater than the one allowed by the declared type is specified, the value may incorrectly be compiled.

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

1. Assembler directive command .DATA is written in the __asm block.
2. The .DATA command in Condition 1 is declared to be .DATA.B or .DATA.W.
3. The integer specified by the .DATA command in condition 1 is greater than the one allowed by the declared type.

Example:

```
--------------------------------------------------------------
-------
void func(void){
   __asm{
      L1: .data.w "0x12345678"
         //  0xffff or less is allowable.
      L2: .data.b "0x1234"
         //  0xff or less is allowable.
```

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

(9) On using intrinsic functions for condition code operation (H8C-0012)
In the versions 6.00 Release 00 through 03, using intrinsic function ovfadd or ovfsub may cause incorrect overflow to be generated.

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

1. Intrinsic function ovfadd or ovfsub is used.
2. Either of the following operations is performed:
   Addition of the negative-signed maximum value to the positive-signed maximum value when ovfadd used.
   Subtraction of the negative-signed maximum value from the positive-signed maximum value when ovfsub used.
   Here, the positive-signed maximum value is 2147483647 (0x7FFFFFFF) and the negative-signed maximum value is ?2147483648 (0x80000000).

Example:

```
----------------------------------------------------------
-------
#include<stdio.h>
#include<machine.h>
void main(void){
    if (!ovfaddl(0x7fffffff,0x80000000,0)){
        printf("OK¥n");
    } else {
        printf("NG¥n");
    }
}
----------------------------------------------------------
-------
```

(10) On assigning a pointer to a member of a structure

(H8C-0013)
In the versions 6.00 Release 00 through 03, if a pointer pointing to a structure is assigned to a member of the structure, incorrect accessing may occur.

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

1. Out of the CPU options, H8SXN, H8SXM, H8SXA or H8SXX (-cpu=h8sxn, h8sxm, h8sxa or h8sxx) is selected.
2. Out of the optimizing options, -optimize=1 is selected.
3. The type of a member of a structure is a pointer pointing to the structure.
4. The address of the structure in Condition 3 is assigned to the member in Condition 3.

Example:
```
------------------------------------------------------------
struct data {
  struct data *p ;
};
struct data *P1, *P2 ;
void fnc( void ){
  P1->p = P2->p = P2 ;
}
------------------------------------------------------------
```

(11) On multiply operations with the cpuexpand option selected (H8C-0014)
In the versions 6.00 Release 00 through 03, multiply operations with the cpuexpand option selected may bring incorrect results.

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

1. Out of the CPU options, H8SXN, H8SXM, H8SXA or H8SXX (-cpu=h8sxn, h8sxm, h8sxa or h8sxx) is selected.

2. The Operation size expanded interpretation option (-cpuexpand) is selected.
3. An expression of multiplication exists.

Example:

```
------------------------------------------------------------
------
void (void){
  int a,b;
  long c,d;
  a=b=d=0;
  ++d;
  ++a;
  b+=2;
  c=(volatile long)(a*b);// a*a will be performed.
  ++d;
  if ((d==(volatile long)(a*b)) && (c==2)){
     printf("t026_04 OK¥n");
  } else {
     printf("t026_04 NG¥n");
  }
}
------------------------------------------------------------
------
```

(12) On generating incorrect section names when the abs16 option selected (H8C-0015)
In the versions 6.00 through 03, if a source program containing a switch statement that is expanded to a table is compiled for any CPU of the H8SXN or H8SXM series, section names with prefix ABS16 may be generated.

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

1. Out of the CPU options, H8SXN or H8SXM (-cpu=h8sxn or h8sxm) is selected.
2. Out of the Short absolute addressing mode options, -abs16 is selected.
3. There exist codes automatically expanded to a table by selecting Auto (-case=auto) out of the Switch statement output code selection method

options. Or Table output code selection method (-case=table) is selected.

Example:

```
------------------------------------------------------
------
void main(void){
switch(l){
  case -3:
  case -1:
     l = 10;
     break;
  }
}
------------------------------------------------------
------
```

(13) On assigning or referencing values to members of a structure (H8C-0016)
In the versions 6.00 Release 00 through 03, incorrect values may be assigned to members of a structure.

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

1. Out of the CPU options, H8SXN, H8SXM, H8SXA or H8SXX (-cpu=h8sxn, h8sxm, h8sxa or h8sxx) is selected.
2. Out of the optimizing options, -optimize=1 is selected.
3. A register is allocaetd with variables (A's) of type structure, where the structure is 4 bytes or less in size (each member of the structure is less than 4 bytes).
4. To members of the variables (A's) in Condition 3, references are made or values are assigned.
5. The variables (A's) loaded in the register in Condition 3 are stored on the stack to load other variables (B's) on this register.
6. There exists an expression that accesses the variables (A's) saved in the stack in Condition 5.

Example:

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

```
------
struct _ST{
  char c;
  int i;
}st1;
void f(){
  struct _ST lst1;
  . . . . . . . . .
  lst1.i=1;
  . . . . . . . . .
  switch(x){
     . . . . . . .
     switch(y){
        case 3:
           . . . . . .
           lst1.i =2;
           . . . . . .
           break;
     }
     . . . . . . .
  }
  . . . . . . . . .
  st1=lst1;
  . . . . . . . . .
}
--------------------------------------------------------
------
```

(14) On the others

1. Compiling or linking programs in which the try-catch clause is used may cause errors to arise.
2. Compiling programs may cause (C)4098 errors to arise.
3. Incorrect values may be displayed in the Watch or Locals window in debuggers.
4. The stack analyzing tool may show an icon that represents "the refernece source is unknown".

1.2.3 Optimizing Linkage Editors

The following 6 problems have been fixed:

(1) On using debug information with the Optimization with the same-code unification option (optimize=same_code) selected

If any debugger performs step executions using debug information created when the Optimization with the same-code unification option is valid, a current PC may show incorrect line on the debugger.

(2) On displaying an incorrect warning message with input options used in a sub-command file
If input options are used as shown in Example in a sub-command file, warning message L1010 is incorrectly displayed.

Example:
```
-----------------
  -input=a.obj
  -input=b.obj
  -input=
-----------------
```

(3) On using the Optimization with the same-code unification option
If the Optimization with the same-code unification option is valid, incorrect code may be generated, and if other optimizations at linking are valid, internal errors may arise.

Conditions:
This problem may occur if the following conditions are all satisfied:
1. The goptimize option is selected at compilation.
2. The Optimization with the same-code unification option (optimize=same_code) is valid.
3. The linkage editor optlnk V.8.00.03 or later is used.
4. Object files or library modules in which no accesses to external variables or function calls are made are linked.
5. Optimization is made to the code in the files or modules in Condition 4.

(4) On optimization using short absolute addressing mode
If optimization using short absolute addressing mode is valid, error L2330 may arise incorrectly.

Conditions:
This problem may occur if the following conditions are all satisfied:
1. The goptimize option is selected at compilation.
2. The short absolute addressing mode option (optimize=variable_access) is valid.
3. Either of the following conditions is met:
   a. The address specified by the sbr option is outside of the 16-bit absolute address area.
   b. Variables of type array and structure are placed inside of the 16-bit absolute address area.

(5) On optimizing code for saving and restoring the contents of registers
If a register for storing arguments is specified at compilation, optimizing code for saving and restoring the contents of the register may cause incorrect results to be generated.

Conditions:
This problem may occur if the following conditions are all satisfied:
1. The goptimize option is selected at compilation.
2. The regparam=3option is selected at compilation, or a function that uses the __regparam3 keyword exists.
3. The optimization with register save/restore option (optimize=register) is valid at linking.

(6) If any of the following conditions is met, an internal error arises:
1. The output option is used to output code into separate files with the unreferenced symbols deletion option (optimize=symbol_delete) being valid. (Internal error L4000-7041 arises.)
2. The short absolute addressing mode option (optimize=variable_access) is valid. (Internal error L4000-8996 arises.)
3. The optimization with register save/restore option (optimize=register) is valid. (Internal error L4000-8416 arises.)

2. How to Revise Your Product and Order the Revised One

2.1 Revision (without Charge)
When you are using the product concerned, free-of-charge revision is available.

(1) For Windows version
Download the revised product from the Software Download Site (available on and after middle of this April).

(2) For Solaris and HP-UX version
Please supply the following items of information to your local Renesas Technology sales office or distributor. We will send you the latest version of the product package by return:

Product Type	:	Solaris or HP-UX version
Version No.	:	V.6.01
Release No.	:	Release 00

2.2 First Ordering

When you place an order for the product you want, supply the following items of information to your local Renesas Technology sales office or distributor:

Product Type	:	Windows, Solaris, or HP-UX version
Version No.	:	V.6.01
Release No.	:	Release 00