To our customers,

Old Company Name in Catalogs and Other Documents

On April 1\textsuperscript{st}, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: 

April 1\textsuperscript{st}, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)
Send any inquiries to http://www.renesas.com/inquiry.
We have updated the H8S, H8/300 series C/C++ compiler package to be Ver. 5.0.06 (PS008CAS5-MWR), Ver. 4.0.05 (PS008CAS4-MWR), and Ver. 4.0.09 (PS008CAS4-SLR and PS008CAS4-H7R), respectively. For details on this update, see PS008CAS5-031113E attached to this technical update.

If you have the compiler package of the Windows® version, download the update tool from the following URL:


If you have the compiler package of the UNIX version, request an update to an authorized product distributor.

Attached document: PS008CAS5-031113E

H8S, H8/300 Series C/C++ Compiler Package Update
H8S, H8/300 Series C/C++ Compiler Package Update

Details on this update (problems fixed) are listed below.

1. High-performance Embedded Workshop (only for PS008CAS5-MWR)
   1.1 HEW Network Database Message Box on Windows® Me
   Display of the network database error message box on Windows® Me is prevented.

1.2 Addition and Modification for Data Generated by the Project Generator
   The following CPUs are newly supported for generating projects:
   The I/O definition file (iodefine.h) of the following CPU has been modified:
   H8/3687

2. Compiler
   [Improved Function]
   1) Number of switch Statements Allowed
      The number of switch statements allowed in one file has been changed from 256 to 2048.

   [Problems Fixed]
   1) Illegal Initial Value of union Type
      Fixed the problem in which no error is generated when a string literal without { } that exceeds the size of an array is specified as the initial value of a union-type auto variable that has a char-type array as the member. A code to copy the whole string literal is created and this destroys the stack.

      [Example]
      
      ```c
      void test()
      {
      union {char c[4];} x = "long string"; /* No error is generated though the string literal exceeds an array */
      }
      
      [Conditions]
      This problem occurs when both of the following conditions are satisfied.
      a) There is a union-type auto variable that has a char/unsigned char-type array as the member.
      b) A string literal without { } that exceeds the size of an array has been specified as the initial value of the variable mentioned in a).

      2) Illegal Unnamed Bit-field
      Fixed the problem in which even if the initial value is specified to a structure that has an unnamed bit-field at the top when the member immediately after this structure is an array or a structure, the gap of the unnamed bit-field is not output.
[Example]
struct S {
    char :1;
    char a[3];
} s = {"abc"};

[Output Code]
• Correct object
  \_s:
    .data.l h'00616263

• Incorrect object
  \_s:
    .data.l h'61626300 ; The gap of the unnamed bit-field is not output.

[Conditions]
This problem occurs when all of the following conditions are satisfied.
a) An unnamed bit-field is declared as the first member in a structure.
b) The member immediately after a) is declared as an array or a structure.
c) The initial value is specified as a variable of the structure mentioned above.

3) Illegal Casting to a Pointer Type
Fixed the problem in which when a constant value is cast to a pointer type in a constant-expression operation when the H8/300 or normal mode is in use, the upper word of the constant value is not cleared. After this, the result of the operation will be illegal.

[Example]
long x = (long)(char *)0x12345678;

[Output Code]
.section D,data
\_x:
    .data.l h'12345678 /* The upper two bytes are not cleared */

[Conditions]
This problem occurs when all of the following conditions are satisfied.
a) cpu=300, 300l, 300hn, 2000n, or 2600n has been specified.
b) A constant value is cast to a pointer type.
c) The constant value of b) exceeds two bytes.
4) Illegal Branch Destination
   Fixed the problem in which an illegal branch occurs when a switch statement is described.

   [Example]
   typedef struct { char a; char* b; char* c; } st;
   void func(st x) {
   int i;
   switch(x.a) {
   case 0:
       func1(x->b); /* Common expression */
       func1(x->c);
   :break;
   case 1:
       func1(x->b); /* Common expression */
   :break;
   case 2:
       func1(x->b); /* Common expression */
   break;
   case 3:
       func1(x->b); /* Common expression */
   break;
   <Following codes are omitted>

   [Illegal Code]
   switch(x.a)
   MOV.W @ER6,R1
   CMP.W #-3879,R1
   BEQ L1647:16
   CMP.W #-3884,R1
   BEQ L1671:16
   <----- (This should branch to L1673)
   CMP.W #-3883,R1
   BEQ L1671:16

   L1671:
   L1673:

   [Condition]
   This problem may occur when there are several case statements in a switch statement and an expression in each case statement includes common expressions.
5) Illegal Object by Testing the Bit-field Value

Fixed the problem in which when there is an expression to test the value of a 1-bit-field and another expression to set 0 is described before or after the said expression, the object may be illegal by deleting the code that sets 0.

[Example]

```c
unsigned char X,Y;
struct {
    unsigned char F1:1;
    unsigned char F0:1;
}BIT1,BIT2;
void test(void){
    X = 0;
    if( BIT1.F0 ){ /* Tests the value of a 1-bit-field. */
        if( BIT2.F0 != (Y==0 ? 0 : 1) ){ /* An expression to assign 0 is described in the */
            BIT2.F0 = (Y==0 ? 0 : 1); /* ternary operator. The object may be illegal. */
        }
    }
}
```

[Illegal Code]

```c
BIT2.F0 = (Y==0 ? 0 : 1);
; Illegally deletes instruction sub.b r0l,r0l
mov.b r1l,r0h
beq l12:8
mov.b #1,r0l
l12:
    bld.b #0,r0l
    bst.b #6,@_bit2:32
```

[Conditions]

This problem may occur when both of the following conditions are satisfied.

a) There is an expression to test the value of a 1-bit-field.

b) Another expression to set 0 is described before or after the expression in a).
6) Illegal Object in Comparison of unsigned long
   Fixed the problem in which the result may be illegal when a variable converted to unsigned long type is compared to an unsigned long-type constant (<, <=, >, or >=).

   [Example]
   ```
   unsigned int ui1;
   int sub()
   {
     if (((unsigned long)ui1) > 0x80000000L )
       return(0);
     return(1);
   }
   ```

   [Illegal Code]
   ```
   _sub:
   sub.w  r0,r0
   ; The test expression is always true and the return value is illegal.
   rts
   ```

   [Conditions]
   This problem may occur when all of the following conditions are satisfied.
   a) An unsigned long variable is compared to an unsigned long constant (<, <=, >, or >=).
   b) The variable mentioned in a) is of unsigned char, unsigned short, or unsigned int type.
   c) The constant is within the range of 0x80000000 to 0xFFFFFFFF (-2147483648 to -1).
      (except for <= -1 or > -1 of 0xFFFFFFFF(-1))

7) Illegal Operation for Bit-fields
   Fixed the problem in which an illegal object may be generated when settings for bit-fields within the same data area are consecutively described.

   [Example]
   ```
   struct{
     int s1 :1;
     int s2 :3;
     int s3 :1;
     int s4 :3;
   )bit;
   void func()
   {
     if (x)
       bit.s1=1;
     else
       bit.s2=2;
   }
   ```
[Output Code]

<Incorrect>                      <Correct>
\_func:  \_func:
    mov.w r0,r0                  mov.w r0,r0
    beq 15:8                     beq 15:8
    mov.l #\_bit,er0             bset.b #7,0\_bit:32  ; Values are illegally set to the both bits.
    mov.w @er0,r1
    and.w #-20481,r1
    bra 17:8                     rts
15:                                          15:
    mov.l #\_bit,er0             mov.b 0\_bit:32,r01
    mov.w @er0,r1
    or.w #-24576,r1
    l7:                           l7:
    mov.w r1,\_er0               mov.b r01,0\_bit:32
    rts                           rts

[Conditions]
This problem may occur when settings for bit-fields within the same data area are described in the form of (A) and (B) under one of the following conditions:

a) if statement
   if(---)
      (A)
   else
      (B)

b) while statement
   while((A))
      (B)

c) conditional expression
   ((A) ? (B) : EXP)
   (EXP ? (A) : (B))
8) Illegal Setting for a Bit-field
Fixed the problem in which a constant value may not be correctly set to a long-type bit-field (bit offset + bit size <= 16 or bit offset >16).

<Example>
#include <stdio.h>

struct ST {
    char C1;
    char C2;
    long UL1:6 ;
    long L1 :4 ;
}st;
void main(){
    st.C1 = 10;
    st.C2 = 10;
    st.L1 = 7 ;
    printf("st.L1 : %ld \n",st.L1);
}

[Output Code]
.cpu 2600a:24
.section p,code,align=2
_main:
    push.l er6
    /* st.L1 = 7 ; */
    /* Incorrect */     /* Correct */
    and.b #-4,r0h     mov.w @(2:16,er6),e0    ; Code for reference and setting
    and.b #63,r0l     and.w #-961,e0       ; of a member is illegally deleted.
    or.b #1,r0h       or.w #448,e0
    or.b #-64,r0l     mov.w e0,@(2:16,er6)
    ;
    pop.l er6
    rts

[Conditions]
This problem may occur when all of the following conditions are satisfied.
  a) The code is for setting to a long-type or unsigned-long-type bit-field and bit offset + bit size <= 16 or bit offset >16.
  b) The specified value is a constant.
  c) Optimization (-optimize=1, default) is specified.
  d) A CPU other than 300 or 300L is specified.
9) Compound Assignment of Multiplication to a Bit-field
   Fixed the problem in which the generated code may be illegal when a compound assignment expression of multiplication that assigns to a bit-field is described.

   [Example]
   struct ST {
       unsigned int bit1 :6;
       signed int bit2 :3;
   } st1, st2, *stp;

   sub() {
       //Omitted
       stp->bit1 *= st2.bit2;
       //Omitted
   }

   [Illegal Code]
   mov.w @_st2:32,r0
   mov.w #1539,r1
   jsr @$bfsi$3:24 // Sets the value of st2.bit2 to r0
   mov.l @_stp:32,er2
   mov.b @er2,r1l
   shlr.b #2,r1l
   extu.w r1 // Sets the value of stp->bit1 to r1
   mov.w r0, @(2:16,sp) // Saves r0
   mov.w r1, r0 // Destroys r0
   mulxu.w r0, er0 // Uses the destroyed r0 as st2.bit2
   
   [Conditions]
   This problem may occur when all of the following conditions are satisfied.
   a) There is a compound assignment expression of multiplication.
   b) The variable assigned to in the expression of a) is a bit-field and this bit-field is a pointer-type structure.
   c) -regexpansion (default) is specified.
   d) Optimization (-optimize=1, default) is specified.
   e) Register variables are used in all of [E]R3 to [E]R6.

10) No Output of Code for Assignment to Variables in a Comma Expression
   Fixed the problem in which when the left operand of a comma expression includes (&, ^, or |) of bitwise operation and its result is not to be used, code for assignment to variables included in the bit operation expression will be deleted.

   [Example]
   int  i1, i2, i3;
   sub() {
       ((long)(i2++) & (i3=1)), i1 = 8; /* i2++ and i3=1 are not executed */
   }
[Conditions]
This problem may occur when all of the following conditions are satisfied.
a) The left operand of a comma expression includes bitwise operation and its result is not to be used.
b) The left and right operands of a) include expressions to update variables (this falls under when either
of these operands describes a structure member).
c) The left or right operand of a) describes type conversion (cast).

11) Illegal Debugging Information on Local Variables
Fixed the problem in which debugging information on local variables or parameters allocated to a stack are
incorrect in a function that includes an endless loop and this causes symbol debugging to be incorrect.

[Example]
extern int a;
void func()
{
    char c[4]; /* Symbol debugging by variable c is incorrect */
    c[3] = 1;

    for (;;) {
        a++;
    }
}

[Conditions]
This problem may occur when both of the following conditions are satisfied.
a) Options debug and optimize are specified.
b) A function includes an endless loop and local variables or parameters are allocated to a stack.