RENESAS Tool News

RENESAS TOOL NEWS on March 1, 2006: RSO-SHC-060301D

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

Please take note of the twelve problems described below in using the C/C++ compiler package V.9 for the SuperH RISC engine family of MCUs.

---

1. **Versions Concerned**

   The C/C++ compiler package for the SuperH RISC engine family V.9.00 Release 00--V.9.00 Release 03

2. **Problems**

   2.1 **On Using Two or more Floating Constants (SHC-0052)**

   Using two or more floating constants within a block may generates incorrect instruction codes.

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

   (1) The optimize=1 option is selected.

   (2) The cpu=sh2e option is selected; or the fpu=single option is selected together with any one of the options cpu=sh2afpu, cpu=sh4, and cpu=sh4a.

   (3) As shown in the table below, the two floating constants given by any one of the sets (a), (b), (c), (d) and (e) are used in the specified order. Lines containing no floating constants are allowed to be inserted in between.

	Order	
Set	**1st**	**2nd**
(a)	0.0f	1.0f
(b)	0.0f	-1.0f

(c)	0.0f	2.0f
(d)	1.0f	0.0f
(e)	1.0f	-0.0f

Example:

```
-------------------------------------------
float f1,f2;
void func() {
      :
   f1 = 0.0f;  // Condition (3)-(a)
   f2 = 1.0f;  // Condition (3)-(a)
     :
}
-------------------------------------------
```

Result of compilation:

```
-------------------------------------------
_func:
          :
       MOV.L     L11+2,R1   ; _d1
       MOV.L     L11+6,R4   ; _d2
       FLDI0     FR8
       FMOV.S    FR8,@R1
       ADD       #0,FR8     ; Incorrect instruction generated.
       FMOV.S    FR8,@R4
          :
-------------------------------------------
```

Workarounds:
This problem can be circumvented either of the following ways:

(1) Define an external variable that takes either of the floating constants you are using as an initial value; then replace the constant with the external variable.

(2) Use the optimize=0 option.

2.2 On Performing Product-Sum Operations (SHC-0053)

When a product-sum operation is performed in the loop body of an iteration statement, an incorrect result is obtained (Problem 2A), or the MACH register is not saved and restored at the

beginning and the ending of a function (Problem 2B).

Conditions:
Problem 2A may occur if the conditions (1) through (7) below are all satisfied, and Problem 2B if the conditions (1) through (6), (8) and (9) below are all satisfied.

(1) The optimize=1 option is selected.

(2) The cpu=sh1 option is not selected.

(3) In the program exists an iteration statement.

(4) A product-sum operation is performed in the loop body of the iteration statement in (3) (multiplication and addition may be in different expressions).

(5) Both the operands in the product-sum operation in (4) are the same type of signed short, signed int, or signed long.

(6) Both the operands in (5) above are variables of type pointer or array.

(7) In the loop body of the iteration statement in (3) exists an instruction that update either or both of the MACH and MACL registers in addition to the product-sum operation in (4) (necessary to Problem 2A only).

(8) The macsave=0 option is unselected (necessary to Problem 2B only).

(9) In the function containing the iteration statement in (3) exists none of the following expressions (necessary to Problem 2B only):

  (a) Include functions macw(), macwl(), macl(), macll(), dmulu_h(), dmulu_l(), dmuls_h(), and dmuls_l()

  (b) A multiplicative expression the type of whose operands are neither signed char nor unsigned char but int; and that converts the type of at least one operand to signed long long or unsigned long long.

Example of Problem 2A:

```
-----------------------------------------------
long d[10],e[10];
long long sum;
int func1(short *p, short *q) {
   int i,ret=0;
   for(i=0;i<10;i++) {              // Condition (3)
```

```
      ret += *p++ * *q++;           // Conditions (4)--(6)
      sum += (long long)d[i] * e[i];  // Condition (7)
   }
   return ret;
}
```

Result of compilation:

```
_func1:
           :
        MOV        #0,R2
        LDS        R2,MACL
           :
L11:
        MAC.W      @R10+,@R11+ ; Result of product-sum
operation
                           ; stored in MACL.
        MOV.L      @(4,R9),R1
        MOV.L      @R9,R7
        MOV.L      R1,@-R15
        MOV.L      R7,@-R15
        ADD        #-8,R15
        MOV.L      @R13+,R1
        MOV.L      @R12+,R4
        DMULS.L    R1,R4       ; MACL register updated.
        STS        MACH,R2
        STS        MACL,R5
        MOV.L      R2,@R15
        MOV.L      R5,@(4,R15)
        JSR        @R8
        MOV.L      R9,@-R15
        DT         R14
        BF/S       L11
        ADD        #20,R15
        STS        MACL,R0     ; Incorrect result of product-sum
                           ; operation obtained.
           :
```

Example of Problem 2B:

```
int func2(int *p, int *q) {
```

```
    int i,ret=0;
    for(i=0;i<10;i++) {        // Condition (3)
        ret += *p++ * *q++;    // Conditions (4)--(6)
    }
    return ret;
}
```

Result of compilation:

```
_func1:
        STS.L      MACL,@-R15
                           ; Instruction for saving MACH
                           ; register not generated.
        MOV         #0,R6
        LDS        R6,MACL
        MOV         #10,R2
L11:
        DT         R2
        BF/S        L11
        MAC.L       @R4+,@R5+   ; MACL register updated.
        STS        MACL,R0
                           ; Instruction for restoring
                           ; MACH register not generated.
        RTS
        LDS.L      @R15+,MACL
```

Workarounds:

Those problems can be circumvented any of the following ways:

(1) Assign either of the variables to be used in a product-sum operation into a volatile-qualified variable; then use the latter variable in the product-sum operation.

(2) Use the optimize=0 option.

(3) Use the macsave=0 option (effective against Problem 2B only).

2.3 On Using do-while Statements (SHC-0054)

Even if the number of iterations is one, do-while statements may be iterated twice or more.

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

(1) The optimize=1 option is selected.

(2) In the program exists a do-while statement.

(3) In the loop body of the do-while statement in (2) exists a controlled variable whose type is signed int, unsigned int, signed long, or unsigned long.

(4) The combination of the initial value and the increment of the controlled variable and the controlling expression in the do-while statement is either (a) or (b) below.

  (a) The initial value is 0, the increment is 1, and the controlling expression is such a comparison as Controlled variable < Non-constant expression, or Non-constant expression > Controlled variable

  (b) The initial value is 1, the increment is 1, and the controlling expression is such a comparison as Controlled variable <= Non-constsnt expression, or Non-constant expression >= Controlled variable

(5) The result of evaluation of the non-constant expression in (4) is a negative integer.

Example:

```
-----------------------------------------------
unsigned long N = 0;         // Condition (5)
void func(void) {
   unsigned long count = 0;  // Conditions (3) and (4)-(a)
   do {                      // Condition (2)
      count++;
   } while (count < N);      // Condition (4)-(a)
}
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
_func:
        MOV.L       L13+2,R2   ; _N
        MOV.L       @R2,R2     ; R2 is set to 0.
```

```
L11:
        DT      R2      ; R2 becomes 0xFFFFFFFF and
                        ; bit T becomes 0 after 1st looping.
        BF      L11     ; Bit T being 0 after 1st looping
                        ; makes program jump to L11.
        RTS
        NOP
-------------------------------------------
```

Workarounds:
This problem can be circumvented any of the following ways:

(1) Make changes to the initial value, the increment, and the controlling expression so that they might meet Condition (4).

(2) Change the do-while statement to a for or while statement.

(3) Change the type of the controlled variable in Condition (3) to signed char, unsigned char, signed short, or unsigned short.

(4) Qualify the controlled variable in Condition (3) to be volatile.

(5) Use the optimize=0 option.

2.4 On Using a Function Containing the Same Two or More Expressions (SHC-0055)

When a function contains the same two or more expressions, and a runtime-routine is used to evaluate any of the expressions, the result of evaluation by the runtime-routine may be saved on an incorrect stack area.

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

(1) The optimize=1 option is selected.

(2) A function contains the same two or more expressions.

(3) Each of those expressions in (2) contains any of the following variables and constants:

  (a) a variable of type long long or unsigned long long

(b) an integer constant outside the range of -2147483648 through 2147483647

(c) a variable or constant of type double except when option cpu=sh2afpu, cpu=sh4, cpu=sh4a, or double=float used

(d) a variable or constant of type long double except when option cpu=sh2afpu, cpu=sh4, or cpu=sh4a used

Example:

```
-----------------------------------------------
extern void g();
unsigned long b[4];

void func() {
   static long c[4][4][4];
   long d[4];
   int i,j,k;
   for(i=0; i<4; i++) {
      for(j=0; j<4; j++){
         for(k=0; k<4; k++){
            c[i][j][k] = 0;
         }
      }
   }
   for(i=0; i<4; i++) {
      d[i]=0;
   }
   for(i=0; i<4; i++) {
      b[i] = 2147483648;          // Conditions (2) and (3)-(b)
   }
   for(i=0; i<4; i++) {
      if (b[i] == 2147483648u) {    // Conditions (2) and (3)-(b)
         g();
      }
   }
}
-----------------------------------------------
```

Result of compilation:

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

```
_func:
        MOV.L      R8,@-R15
        MOV.L      R9,@-R15
        MOV.L      R10,@-R15
        MOV.L      R11,@-R15
        MOV.L      R12,@-R15

        MOV.L      R13,@-R15
        MOV.L      R14,@-R15
        STS.L      PR,@-R15
        ADD        #-12,R15
        MOV        #-128,R1   ; H'FFFFFF80
        SHLL8      R1
        SHLL16     R1
        MOV.L      R1,@-R15
        MOV        #0,R4      ; H'00000000
        MOV.L      L24+2,R5   ; __conv64u
        JSR        @R5
        MOV.L      R4,@-R15
        MOV.L      R0,@R15    ; Must be MOV.L   R0,@(8,R15).
        ADD        #8,R15
            :
```

Workaround:
Use the optimize=0 option to circumvent this problem.

2.5 On Using a Parameter within a Division or Remainder Operation (SHC-0056)

Using a parameter as a divisor or dividend of a division or remainder operation may give an incorrect result.

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

(1) In the program exists a division or remainder operation.

(2) Either or both of the divisor and dividend of the division or remainder operation in (1) are parameters.

(3) The parameters in (2) are of type signed char, unsigned char, signed short, or unsigned short.

(4) The divisor and the dividend of the division or remainder operation in (1) are different in types.

Example:

```
-------------------------------------------
short func(unsigned short y) {  // Condition (3)
   short x=-200;
   x /= y;                  // Conditions (1), (2), and (4)
   return x;
}
-------------------------------------------
```

Result of compilation:

```
-------------------------------------------
_func:
        MOV.L      L11+4,R2   ; __divwu


        MOV.W      L11,R1     ; H'FF38
        JMP        @R2        ; Unsigned division of 2 bites wide
                              ; performed in place of signed
                              ; division of 4 bites wide.
        EXTU.W     R4,R0
-------------------------------------------
```

Workarounds:

This problem can be circumvented either of the following ways:

(1) When using a parameter within a division operation, explicitly convert its type to the one that has been used at its declaration. In Example above, the conversion is as follows:

```
x /= (unsigned short)y;
```

(2) Assign the parameter to a variable; then use the variable within a division operation.

2.6 On Using a Subtraction Expression within the Loop Body of an Iteration Statement (SHC-0057)

When a subtraction expression exists in the loop body of an iteration statement, and the subtrahend of the subtraction is a induction variable of type signed char, unsigned char, signed short, or unsigned short, the evaluation of the subtraction expression may bring an incorrect result.

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

(1) The optimize=1 option is selected.

(2) In the program exists an iteration statement.

(3) The iteration statement in (2) contains an induction variable of type signed char, unsigned char, signed short, or unsigned short.

(4) The loop body of the iteration statement in (2) contains a subtraction expression.

(5) The subtrahend of the subtraction expression in (4) is the induction variable in (3).

Example:

```
------------------------------------------------
int A[10];
int X = 10;
void func()
{
   unsigned char i;          // Condition (3)
   for (i = 0; i < X; i++) {  // Condition (2)
      A[2 - i] = 1;         // Conditions (4) and (5)

   }

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

Result of compilation:

```
------------------------------------------------
_func:
        MOV.L      L14+2,R2   ; _X
        MOV        #0,R5      ; H'00000000
        MOV.L      @R2,R1
        BRA        L11
        MOV        #1,R7      ; H'00000001
L12:
        NEG        R5,R2
        EXTU.B     R2,R0      ; Result of NEG instruction
                              ; zero-expanded excessively.
        MOV.L      L14+6,R4   ; _A
```

```
        ADD       #2,R0
        SHLL2     R0
        MOV.L     R7,@(R0,R4)
        ADD       #1,R5
L11:
        EXTU.B    R5,R2
        CMP/GE    R1,R2
        BF        L12
-------------------------------------------
```

Workarounds:
This problem can be circumvented any of the following ways:

(1) Change the type of the recursive variable to signed int, unsigned int, signed long, or unsigned long.

(2) Change the subtraction expression to an addition expression by interchanging the two operands. For instance, expression A[2+-i] in the above example will be changed to A[-i+2].

(3) Qualify the induction variable to be volatile.

(4) Use the optimize=0 option.

2.7 On Declaring Pointer-Type Members within a Structure or Union Having an Alignment Number of 1 (SHC-0058)

When a pointer-type member is declared in a structure or union having an alignment number of 1 and passed as an argument to a strcpy() function, performing this function may bring an incorrect result. When the above member is type-cast to int and then used in an operation, performing the operation may also bring an incorrect result.

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

(1) The unaligned=runtime option is selected. Or, the size option is selected with the unaligned option not selected.

(2) In the program exists a structure or union.

(3) The pack=1 option is selected. Or, the structure or union in (2) is declared to be #pragma pack 1.

(4) The structure or union in (2) contains a member of type pointer.

(5) The member in (4) meets condition (a) or (b) below.

(a) It is passed to a strcpy() function as an argument.

(b) It is type-cast to int and then used as an operand of a multiplication, division, remainder, or shift operation.

Example

```
-----------------------------------------------
#include <string.h>
#pragma pack 1                // Condition (3)
struct ST {
   char *string;              // Condition (4)
} st;                         // Condition (2)
void func (const char *ptr) {
   strcpy(st.string, ptr);    // Condition (5)-(a)
}
-----------------------------------------------
```

Result of compilation:

```
-----------------------------------------------
_func:
        STS.L      PR,@-R15
        MOV.L      L11,R1     ; _st
        MOV.L      L11+4,R3   ; __pack1_ld32
        MOV.L      L11+8,R2   ; __slow_strcpy
        JSR        @R3
        NOP                   ; Because MOV R4,R1 not generated,
                              ; address of st passed to strcpy()
                              ; as 2nd argument.
        JMP        @R2
        LDS.L      @R15+,PR
-----------------------------------------------
```

Workarounds:

This problem can be circumvented either of the following ways:

(1) Declare the structure or union concerned without #pragma pack 1, and do not use the pack=1 option.

(2) Use the unaligned=inline option.

2.8 On Performing an Operation Resulting in an Overflow (SHC-0059)

When the result of an operation is assigned to a variable, an incorrect result may be obtained if the correct result cannot be expressed within the range specified by the type of the variable.

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

(1) The optimize=1 option is selected.

(2) Either of the group conditions (a) or (b) is met.

  (a) The following conditions, (a-1) through (a-4), are all met:

    (a-1) An addition, subtraction, multiplication, left shift, or unary minus operation is performed.

    (a-2) The result of operation in (a-1) is assigned to a variable of type signed char, unsigned char, signed short, or unsigned short.

    (a-3) The result of operation in (a-1) cannot be expressed within the range specified by the type of the variable in (a-2).

    (a-4) The variable in (a-2) is not qualified to be volatile.

  (b) The following conditions, (b-1) through (b-4), are all met:

    (b-1) An assignment expression whose right term is a division expression exists.

    (b-2) The divisor and dividend of the division expression and the destination of the assignment in (b-1) are the same type, signed char or signed short.

    (b-3) The divisor and dividend of the division expression in (b-1) are explicitly type-cast to unsigned long or unsigned int.

    (b-4) The values of the divisor and dividend in (b-3) are both negative before type-cast.

Example

```
-----------------------------------------------
// In Y, correct value of -128 replaced with incorrect 128.
char X = -128;
int Y, Z = 0;
void func() {
   char t;
   t = 0 - X;  // Condition (2)-(a's)
   Y = t + Z;
}
-----------------------------------------------
// In Z, correct value of 0 replaced with incorrect -1.
signed char X=2,Y=-2,Z;
void func() {
   Z = (unsigned int)X / (unsigned int)Y;  // Condition (2)-(b's)
}
-----------------------------------------------
```

Workarounds:
This problem can be circumvented any of the following ways:

(1) Use the optimize=0 option.

(2) Qualify to be volatile the variable in Conditions (a-2) to which an assignment is made if Conditions (2)-(a's) are met.

(3) Assign either the divisor or the dividend to a variable of type unsigned long or unsigned int, and then use the variable within the division expression if Conditions (2)-(b's) are met.

(4) Or, type-cast both the divisor and the dividend to unsigned short if Conditions (2)-(b's) are met.

2.9 On Using Special Loops (SHC-0060)

When such a special loop as its controlled variable overflows, either or both of the total number of iterations and the value of the controlled variable in iterating may become incorrect.

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

(1) The optimize=1 option is selected.

(2) In the program exists an iteration statement.

(3) The iteration statement in (2) meets any of the following conditions, (a) through (g):

(a) The controlled variable in the iteration statement in (2) overflows during iterations.

(b) The increment value of the controlled variable in the iteration statement in (2) is 0x80000000, the minimum value of type signed int.

(c) The following conditions, (c-1), (c-2), and (c-3), are all met:

(c-1) The controlling expression of the iteration statement in (2) is a comparison, in which the controlled variable is type-cast.

(c-2) The inequality operator != is used in the comparison in (c-1).

(c-3) If the inequality operator used in the comparison in (c-1) is replaced with the less-than or greater-than operator, the evaluation of TRUE or FALSE of the controlling expression at the first iteration is inverted according to whether the controlled variable is type-cast or not.

(d) The following conditions, (d-1), (d-2), and (d-3), are all met:

(d-1) The controlling expression of the iteration statement in (2) is a comparison, in which the controlled variable is type-cast.

(d-2) The relational operator, <, <=, >=, or > , is used in the comparison in (d-1).

(d-3) The evaluation of TRUE or FALSE of the controlling expression at the first iteration is inverted according to whether the controlled variable is type-cast or not.

(e) If the initial and increment values of the controlled

variable are I and S, and the maximum number of iterations is F in the iteration statement in (2), the value of (F-1)-I or ((F-1)-I)+S cannot be expressed within the range specified by the type of the controlled variable.

(f) The following conditions, (f-1) and (f-2), are both met:

(f-1) If the initial and increment values of the controlled variable are I and S, and the maximum number of iterations is F in the iteration statement in (2), the value of F-1 or F-S cannot be expressed within the range specified by the type of the controlled variable.

(f-2) In the loop body of the iteration statement in (2) exists an if statement, and its controlling expression is a comparison of the controlled variable with a constant.

(g) The following conditions, (g-1), (g-2), and (g-3), are all met:

(g-1) In the loop body of the iteration statement in (2) exists a multiplication or left-shift operation containing an induction variable.

(g-2) The result of evaluation of the multiplication or left-shift operation in (g-1) may not be expressed within the range specified by the type of the operation.

(g-3) The induction variable in (g-1) is referenced after the iteration statement in (2).

(4) The controlled variable or recursive variable are not qualified to be volatile.

Example:

```
-----------------------------------------
// Single iteration replaced with no iteration.
int a;
```

```
void func() {
   short i;
   a = 0;
   for (i = -129; (char)i != -128; i++) { // Condition (3)-(c)

      a++;
   }
}
-----------------------------------------
// No iteration replaced with one or more iterations.
int a;
void func() {
   unsigned int i;
   for (i = 1; i < 0; i++) { // Condition (3)(e)
                       // When F=0 and S=1, F-1 and F-S beyond
                       // range specified by unsigned int
      if (i < 20) {
         a++;
      }
   }
}
-----------------------------------------
```

Workarounds:
This problem can be circumvented any of the following ways:

(1) Use the optimize=0 option.

(2) If Conditions (3)-(a) through -(f) are met, qualify the controlled variable of the iteration statement to be volatile.

(3) If Conditions (3)-(g) met, qualify the induction variable to be volatile.

2.10 On Using the opt_range Option (SHC-0061)

When the opt_range=noblock or the opt_range=noloop option is selected, optimization of external variables may be performed so much that the limitations imposed on the option can be exceeded.

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

(1) The optimize=1 option is selected.

(2) The opt_range=noblock or the opt_range=noloop option is selected.

(3) In the program exists an iteration statement.

(4) An external variable exists in the loop body or the controlling expression of the iteration statement in (3).

Example:

```
-----------------------------------------------
// -opt_range=noblock selected.
int a[100];
int X,Y,Z;
void func() {
   int i;
   for (i=0; i<100; i++) {  // Condition (3)

      a[i] = X+Y;         // Condition (4)
      if (X) {
         a[i] = Y+Z;

      }

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

Image of source program after optimization:

```
-----------------------------------------------
int a[100];
int X,Y,Z;
void func() {
   int i;
   for (i=0; i<100; i++) {
      temp = X+Y;

      if (X) {

         temp = Y+Z;
      }
      a[i] = temp;    // Assignment to a[i] performed only once
                      // regardless of evaluation of if statement.
   }
```

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

Workaround:
To circumvent this problem, use the optimize=0 option.

2.11 On using the volatile_loop Option (SHC-0062)

Using options volatile_loop and infinite_loop=1 may delete the reference to an external variable in the controlling expression of an iteration statement.

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

(1) The optimize=1 option is selected.

(2) The volatile_loop option is selected.

(3) The infinite_loop=1 option is selected.

(4) In the program exists an iteration statement.

(5) No variables whose values are updated in the loop body exist in the controlling expression of the iteration statement in (4).

(6) An external variable exists in the controlling expression of the iteration statement in (4).

(7) An assignment expression to the external variable in (6) is placed before the iteration statement in (4).

Example:

```
-----------------------------------------
signed int n;
signed int a[100];
void func() {
   n = 0;            // Condition (7)
   while (n == 1) {    // Conditions (4)--(6)
      a[n] = 0;
   }
}
-----------------------------------------
```

Result of compilation:

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

```
_func:
        MOV.L      L11,R6     ; _n
        MOV        #0,R2      ; H'00000000
                              ; References to n and
                              ; while statement deleted.
        RTS
        MOV.L      R2,@R6
```

Workarounds:
This problem can be circumvented any of the following ways:

(1) Place an nop() include function between the iteration statement in Condition (4) and the assignment expression in Condition (7).

(2) Use the infinite_loop=0 option.

(3) Use the optimize=0 option.

2.12 On Using the infinite_loop=0 Option (SHC-0063)

When the infinite_loop=0 option is selected, and an infinite loop is formed by a swith and a goto statement, the reference to an external variable placed immediately before the infinite loop may be deleted.

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

(1) The optimize=1 option is selected.

(2) The infinite_loop=0 option is selected.

(3) An infinite loop is formed by a switch and a goto statement.

(4) An external variable is referenced before the switch statement in (3).

(5) The external variable in (4) is not qualified to be volatile.

Example:

```
int a;
void func() {
```

```
   a=1;              // Condition (4)
Label:
   switch (1) {      // Condition (3)
   case 1:

      goto Label;
   default:
      break;
   }
}
```

Result of compilation:

```
_func:
L15:
                        ; Assignment to a deleted.
       BRA       L15       ; Infinite loop
       NOP
       RTS
       NOP
```

Workarounds:
This problem can be circumvented either of the following ways:

(1) Qualify the external variable in Condition (4) to be volatile.

(2) Use the optimize=0 option.

3. Schedule of Fixing the problem

We plan to fix this problem in the next release, V.9.00 Release 04, of the product.
