RENESAS Tool News

RENESAS TOOL NEWS on January 16, 2005: RSO-SHC-050116D

The C/C++ Compiler Packages for the SuperH RISC engine family Revised to Their V.9.00 Release 01

We have revised the C/C++ compiler packages for the SuperH RISC engine family from their V.9.00 Release 00 to V.9.00 Release 01.

---

1. **Descriptions of Revision**

1.1 Functions Introduced and Improved

1.1.1 In Simulator Debugger (for Windows Version Only)

(1) Little-endian-type data transfer supported in the simulator debugger for the SH-2 core.

(2) Memory resources reserved automatically at loading the user program into the simulator.

(3) If a memory-access error detected, the address where the error has arisen displayed.

1.1.2 In Compiler

(1) The following options introduced:

(a) bss_order={declaration|definition}
- Specifies the order of memory assignments to the uninitialized variables.

(b) stuff[={bss|data|const}[,...]] and nostuff
- Assigns a segment to a variable according to its size.

For details of these options, see "SuperH RISC engine C/C++ Compiler, Assembler, Optimizing Linkage Editor Compiler Package V.9.00 User's Manual (Rev.1.01)".

(2) Little-endian-type data transfer supported in the simulator for the SH-2 core. Note, however, that this

function cannot be used in the devices not supporting little endian.

1.2 Problems Fixed

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

The problem on unloading ELF/DWARF2-formatted load modules has been fixed. (Unloading load modules generated in the ELF/DWARF2 format results in the High-performance Embedded Workshop's abnormal termination.)
For details, see RENESAS TOOL NEWS "A Note on Using Integrated Development Environment High-performance Embedded Workshop--On Unloading ELF/DWARF2-Formatted Load Modules," issued on November 1, 2004.

1.2.2 In Simulator Debugger (for Windows Version Only)

The following problems have been fixed.

(1) On creating projects for the SH7206 microprocessor: During simulations of projects for the SH7206 microprocessor, simulations may be halted or memory access errors may arise because necessary memory resources for operand caches cannot be acquired.
For details, see RENESAS TOOL NEWS "A Note on Using a C/C++ Compiler Package V.9.00 Release 00 for the SuperH RISC engine Family Devices," issued on October 16, 2004.

(2) On executing the RESBANK instruction in the SH2A-FPU Cycle Base Simulator: When the RESBANK instruction is executed in the SH2A-FPU Cycle Base Simulator, the number of cycles for executing the instruction becomes different from the specified ones.
For details, see RENESAS TOOL NEWS "A Note on Using a C/C++ Compiler Package V.9.00 Release 00 for the SuperH RISC engine Family Devices," issued on November 1, 2004.

1.2.3 In Compiler

The following 12 problems have been fixed:

(1) When the following functions in the DSP libraries are used in programs for the SH4AL-DSP (a CPU core), they may not operate properly:
FftComplex, FftReal, IfftComplex, IfftReal, FftInComplex, FftInReal, IfftInComplex, IfftInReal, Fir, ConvComplete, ConvCyclic, ConvPartial, Correlate, CorrCyclic, MatrixMult, MsPower, and Variance

As a result of this fixing, the file names of the libraries for the SH4AL-DSP have been changed as follows:

Option selected	Library file name
-pic=0 -endian=big	sh4aldspnb.lib
-pic=1 -endian=big	sh4aldsppb.lib
-pic=0 -endian=little	sh4aldspnl.lib
-pic=1 -endian=little	sh4aldsppl.lib

(2) Loops containing a controlling expression may be executed an incorrect number of times if the following conditions are all satisfied (SHC-0008):

Conditions

1. The optimize=1 option is selected.
2. A program loop exists.
3. The increment or decrement of the controlled variable in the loop in Condition 2 is 1.
4. The above loop contains an if statement.
5. Either condition 5a or 5b below is met.
   - 5a. The controlling expression in the if statement in Condition 4 compares the controlled variable in the loop in Condition 2 with an invariant in the loop (variable c in Example 1 below).
   - 5b. The initial value or the upper limit of the controlled variable in the loop in Condition 2 is an invariant in the loop (variable c in Example 2 below).
6. The invariant in Condition 5a or 5b is of type int.

Example 1:

```
-----------------------------------------------
int b[100];
unsigned int c=0;
void func1() {
  unsigned int i;
  for(i=0;i<=100;i++) {     // Executed infinite times.
     if(i != c) {
     b[i]=0;
     }
```

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

Example 2:

```
-----------------------------------------
int b[100];
unsigned int c=0;
void func2() {
 unsigned int i;
 for(i=0;i<c;i++) {       // Executed once or more even if c
= 0.
   if(i != 5) {
      b[i]=0;
   }
 }
}
-----------------------------------------
```

(3) Do-while loops to be executed only once may be repeated twice or more if the following conditions are all satisfied (SHC-0010):

Conditions

1. The optimize=1 option is selected.
2. A do-while loop exists.
3. The controlled variable in the loop in Condition 2 is of type int, signed int, long, or signed long.
4. The controlling expression in the loop in Condition 2 checks whether its controlled variable is less than, less than or equal to, greater than, or greater than or equal to a constant.
5. The comparison operator, the initial value of the controlled variable, its incremented or decremented value in the loop, and the constant to be compared in the controlling expression satisfy any of the following conditions, 5a through 5d:

   5a. If the controlled variable of the loop is less than the constant to be compared, the following relations are all held:

   - The incremented or decremented value is positive

- The constant to be compared is less than or equal to the initial value of the controlled variable.
- 0x00000000 <= (constant - initial value - 1) <= 0x7FFFFFFF.

5b. If the controlled variable of the loop is less than or equal to the constant to be compared, the following relations are all held:
- The incremented or decremented value is positive.
- The constant to be compared is less than the initial value of the controlled variable.
- 0x00000000 <= (constant - initial value - 1) <= 0x7FFFFFFF.

5c. If the controlled variable of the loop is greater than the constant to be compared, the following relations are all held:
- The incremented or decremented value is negative.
- The constant to be compared is greater than or equal to the initial value of the controlled variable.
- 0x00000000 <= (initial value - constant - 1) <= 0x7FFFFFFF.

5d. If the controlled variable of the loop is greater than or equal to the constant to be compared the following relations are all held:
- The incremented or decremented value is negative.
- The constant to be compared is greater than the initial value of the controlled variable.
- 0x00000000 <= (initial value - constant - 1) <= 0x7FFFFFFF.

Example:

```
----------------------------------------
int func() {
int count=0;
```

```
int limit=0x60000000;
do {
   count++;
   limit += 0x10000000;
} while(limit < -0x60000000);
                  // If executed correctly, the expression
                  // is FALSE after the first looping,
                  // and the loop is exited.
   return (count);        // "count" takes another value
                     // than the correct  1 .
}
----------------------------------------
```

(4) When 1 is compared with a signed bit field of 1 bit wide or with the result of an operation performed on that of any comparison, the incorrect result may be obtained if either of the following conditions is satisfied (SHC-0011):

Conditions

1. The conditions stated below are all met.
   - 1a. The optimize=1 option is selected.
   - 1b. A signed bit field of 1 bit wide is used.
   - 1c. Equality or inequality (== or !=) between 1 and the signed bit field in Condition 1b is tested.
2. The conditions stated below are all met.
   - 2a. The optimize=1 option is selected.
   - 2b. Any of the following operations is performed on the result of any comparison: (1) subtraction between it and 1, (2) XOR operation of it and 1, (3) its sign inversion, and (4) its bitwise inversion.
   - 2c. Equality or inequality (== or !=) between 1 and the result of the operation in Condition 2b is tested.

Example 1:

```
----------------------------------------
struct {
  char b0:1;
} ST;
void func() {
```

```
  if (ST.b0 != 1) {
  . . . . . .
  }
}
```

Example 2:

```
int a;
void func2() {
  int t;
  t = ((a & 0x40) == 0);
  t = t - 1;
  t = -t;
  if(~t==1) {
     a = 1;
  } else {
     a = 2;
  }
}
```

---

(5) When the result of an add or subtract operation between a variable and 0 or a multiply operation of a variable by 1 is used in another operation, a change may incorrectly be made to the value of the variable if the following conditions are all satisfied (SHC-0012):

Conditions

1. An addition/subtraction of 0 to/from a variable or a multiplication of a variable by 1 is performed.
2. The result of the operation in Condition 1 is used in another operation such as addition, subtraction, bitwise AND, bitwise OR, bitwise XOR, division, remainder or shift.

Example:

```
int a[4], b;
void func() {
      a[3&(b-0)]=0;
}
```

---

(6) When a double-type member of a structure or union for which "pack" is defined as 1 is referenced, a change may incorrectly be made to the value of register R2 if the following conditions are all satisfied (SHC-0013):

Conditions

1. A structure or union exists for which pack=1 is used.
2. The structure or union in Condition 1 contains a member of type double.
3. CPU options cpu=sh4, sh4a, and sh2afpu are used.
4. Option "size" or "unaligned=runtime" is selected.
5. The runtime-routine "_pack1_ld64" is called at referencing.

Example:

```
----------------------------------------
#pragma pack 1
struct {
        double d;
} ST;
int t;
double d[2];
void func() {
        d[t]=ST.d;
}
----------------------------------------
```

(7) In an expression containing both a multiplication and division by constants, an incorrect result may be obtained if the following conditions are all satisfied (SHC-0015):

Condition

1. A multiplication of an unsigned-type expression by a constant exists.
2. The expression in Condition 1 is divided by a positive factor of the constant in Condition 1.
3. The result of the multiplication in Condition 1 exceeds the maximum value allowed to the type of the expression.

Example:

```
----------------------------------------
unsigned int a=65536;
unsigned int b;
void func() {
```

```
    b=(a*65536)/8;   // The correct result b=0
                     // ((65536*65536)/8 =>> 0/8=0)
}                    // replaced by b=65535<<13.
----------------------------------------
```

(8) In shift operations, consider each shift count is less than the bit size of the value to be shifted. When such shifts are performed more than once, and the total number of shifts exceeds the bit size of the value to be shifted, an incorrect result of operation may be obtained if the following conditions are all satisfied (SHC-0016):

Conditions

1. Any CPU option parameters except cpu=[sh1|sh2|sh2e|sh2dsp] are used.
2. Either condition 2a or 2b below is met.
   - 2a. Left shifts and multiplications by powers of 2 are performed twice or more, where every shift count and exponent in powers of 2 is less than the bit size of the value to be shifted.
   - 2b. Right shifts and divisions by powers of 2 are performed twice or more, where every shift count and exponent in powers of 2 is less than the bit size of the value to be shifted.
3. The total sum of the shift counts and the exponents in powers of 2 in 2a or 2b exceeds the bit size of the value to be shifted.

Example:

```
----------------------------------------
int x,y;
void func() {
  x=y<<31<<1;  // The total of shift counts, 32, exceeds
               // the bit size of type int.
}
----------------------------------------
```

(9) When a function call is made at the end of a function, the address of the function to be called may incorrectly be loaded into register R0 if the following conditions are all satisfied:

Condition

1. A function call is made at the end of a function.
2. The definition of the function to be called and the description of the function call in Condition 1 are made in the same file.
3. The definition of the function to be called in Condition 2 and the description of the function call in Condition 1 are placed 4096 bytes or more apart or in different sections from each other.
4. The calling function makes only one function call stated in Condition 1.

Example:

```
----------------------------------------
char a[3];
void func2() {}
#pragma section A
void func() {
  ++a[2];
  func2();
}
----------------------------------------
```

(10) When an operation is performed on a member of a bit field through a pointer, an incorrect result may be obtained if the following conditions are all satisfied (SHC-0023):

Conditions

1. A structure or union is used.
2. A member of the structure or union in Condition 1 is referenced directly (un.b in Example below).
3. A pointer variable (p in Example) exists which points to a member of the structure or union (un.a in Example) in the same area as the member referenced in Condition 2.
4. The pointer in Condition 3 is a local variable.
5. An indirect reference using the pointer in Condition 3 (*p in Example) exists.
6. Updated is the value of the area where the member referenced in Conditions 2 and 3 exist.
7. The structure or union itself is not referenced in the function where references in Conditions 2 and 3 are made.

Example:

```
----------------------------------------
typedef union {
  unsigned int a;
  unsigned int b:32;
} UN;
UN un;
void func() {
  int *p=(int *)&un.a;
  un.b=1;
  *p+=1;
}
----------------------------------------
```

(11) Copying a structure or union may reserve the stack area more than necessary if the following conditions are all satisfied (SHC-0021):

Conditions

1. An array of structures or unions exists which include members of type structure or union.
2. The array in Condition 1 has two or more elements.
3. An assignment expression to a structure or union exists.
4. The left term of the assignment expression in Condition 3 is a member of a structure or union in the array of structures or unions.

Example:

```
----------------------------------------
typedef struct {
   unsigned char c;
} ST0;
typedef struct {
   ST0 s;
} ST;
extern ST A[1000];
extern unsigned short i;
void func(ST *d) {
   A[i-1].s=d->s;  // Stack reserved redundantly for
A[1000].
}
----------------------------------------
```

(12) The C4098 and C4099 internal errors may arise (SHC-0007, SHC-0017, SHC-0022, and SHC-0025).

1.2.4 In Assembler

The following three problems have been fixed which are encountered in creating programs for the SH-2A or SH2A-FPU core:

(1) Displacement values in extended or branch instructions (in the PC-relative addressing) may become incorrect if the following conditions are all satisfied:

Conditions

1. CPU option cpu=sh2a or cpu=sh2afpu is selected.
2. Before a delayed branch instruction exist an odd number of "MOV @(disp,Rn)" instructions, in which disp:12 is selectable as an addressing mode, but not selected.
3. The instruction following the delayed branch instruction in Condition 2 is a .ALIGN or .ORG directive one.
4. The directive instruction in Condition 3 makes an address adjustment of 2 bytes.
5. An extended or branch instruction (in the PC-relative addressing) follows the directive instruction in Condition 3.

Example:

```
----------------------------------------
    .ALIGN  4
    NOP
    MOV.L   @(AA,R1),R0
    BRA     LABEL
    .ALIGN  4
    BT      LABEL
. . . . . .
LABEL: NOP
AA:   .EQU    4
----------------------------------------
```

(2) The destination of an address branching may become incorrect if the following conditions are all satisfied:

Conditions

1. CPU option cpu=sh2a or cpu=sh2afpu is selected.
2. Before a delayed branch instruction exist an odd number of "MOV @(disp,Rn)" instructions, in which disp:12 is selectable as an addressing mode, but not selected.
3. The instruction following the delayed branch instruction in Condition 2 is a .ALIGN or .ORG directive one.
4. The directive instruction in Condition 3 makes an address adjustment of 2 bytes multiplied by the number of MOV @(disp,Rn)instructions.
5. An address branch instruction follows the directive instruction in Condition 3.

Example:

```
----------------------------------------
    .SECTION  SEC1,CODE
    MOV.L     @(AA,R1),R0  ; 2-byte instruction
    NOP
    BRA       L3
    .ALIGN    4            ; Location counter; adjusted
    BF        L1           ; Not delayed slot instruction
L3:   NOP
    .SECTION  SEC2,CODE
L1:   NOP
AA:   .EQU     4
----------------------------------------
```

(3) The result of operation performed using an even operator $even2 and an odd operator $odd2 may become incorrect.

1.2.5 In Optimizing Linkage Editor

The following five problems have been fixed:

(1) When step-execution is performed in the debugger by using debug information generated under the condition the optimization with the sama-code unification (optimize=same_code) is effective, the program may branch to an incorrect function.

(2) If a .stack directive command is not specified in a defined symbol and specified only in a referenced symbol, the information on the stack size of the symbol is not output to the sni file.

(3) When the optimization with the sama-code unification (optimize=same_code) is effective, reading a relocatable file at linking may generate incorrect code if the following conditions are all satisfied:

Conditions

1. The goptimize option is used at compilation.
2. The optimization with sama-code unification option (optimize=same_code) is effective.
3. The optimizing linkage editor V.8.00.03 or later is used.
4. A relocatable file is read at linking.
5. The codes in the relocatable file in Condition 4 are unified into 4n + 2 bytes by the optimization in Condition 2.

(4) When optimization by unifying the same constants and literals is effective, an improper error (L2330) may arise if the following conditions are all satisfied:

Conditions

1. The goptimize option is used at compilation.
2. The optimization by unifying the same constants and literals (optimize=string_unify) is effective.
3. The abs16 option is used at compilation or a variable of type const that is declared to be #pragma abs16 exists.

(5) An internal error arises in each of the following cases:

1. When optimization by deleting un-referenced symbols (optimize=symbol_delete) is effective, an output file is divided into more than one file using the output option. (Internal error 7041 arises.)
2. The binary option is used when the profile option has already been selected. (Internal error L4001 arises.)

2. How to Revise Yours and Order Revised Ones

2.1 Free-of-Charge Revision

(1) If you are using the Windows version, revise it online by visiting the Download Site.

(2) If you are using the Solaris version or the HP-UX version, please supply the following items of information to your local Renesas Technology sales office or distributor:

Product Types : V.9.00

Version No. : V.9.00

Release No. : Release 01

2.2 First Ordering

When you place an order for any of the products concerned, please supply the following information to your local Renesas Technology sales office or distributor:

Product Types : Windows, Solaris, or HP-UX Version

Version No. : V.9.00

Release No. : Release 01