Microcontroller Technical Information

Integrated Development Environment Package CubeSuite Usage Restrictions	Document No.	ZMT-F35-10-0011	1/3
	Date issued	March 23, 2011	
	Issued by	MCU Tool Product Marketing Department MCU Software Division MCU Business Unit Renesas Electronics Corporation	
Related documents None	Notification classification	√	Usage restriction
			Upgrade
			Document modification
			Other notification

1. Affected products

   CubeSuite (Product name: QS78K, QS850)

   For the affected versions, see the attachment for each product.

2. New restrictions

   The following restrictions have been added:

   - CA78K0 (Attachment 3)

     No. 75 Macro expansion using the `##` operator results in an error

     No. 76 Symbol information for an interrupt function is not output to the assembler source

   - CX (Attachment 5)

     No. 8 A variable referenced after being changed in a function call becomes an invalid value.

     No. 9 A variable referenced after indirect assignment becomes an invalid value.

   - CA78K0R (Attachment 9)

     No. 30 The C0101 error occurs when the `-qj` option is specified

     No. 31 Restriction on type-casting addresses in the `far` area to a `long` or `unsigned long`

     No. 32 Macro expansion using the `##` operator results in an error

     No. 33 Symbol information for an interrupt function is not output to the assembler source

     No. 34 Code that specifies the ES register settings might not be output

3. Workarounds

   The workarounds for the above restrictions are shown below. For details, see the attachment.

   - CA78K0 (Attachment 3)

     No. 75 Do either of the following:

     1. Do not use the `##` operator.
     2. Place a function-like macro parameter immediately after the `##` operator.

     No. 76 Define the interrupt function or output the object module file.

- CX (Attachment 5)

No. 8	Do either of the following: 1. Specify an optimization option other than `-O`, `-Osize`, and `-Ospeed`. 2. Insert `__asm("\n");` in any line in function `F1`. 3. Declare variable `V` as `volatile`.
No. 9	Do either of the following: 1. Specify an optimization option other than `-O`, `-Osize`, and `-Ospeed`. 2. Insert `__asm("\n");` in any line in function `F1`. 3. Declare variable `V` as `volatile`.

- CA78K0R (Attachment 9)

No. 30	Do either of the following: 1. Disable the`-qj` option. 2. Delete the code that is not executed or embed the code between `#if 0` and `#endif` so as not to make it subject to compiling.
No. 31	Do not type-cast an address to a `long` or `unsigned long` if the initial value is used for the address.
No. 32	Do either of the following: 1. Do not use the `##` operator. 2. Place a function-like macro parameter immediately after the `##` operator.
No. 33	Define the interrupt function or output the object module file.
No. 34	Immediately after incrementing or decrementing a floating point variable, insert a dummy code that references the variable by using a pointer.

4. Modification schedule

The planned modifications are described after each item.

5. List of restrictions

For the history of restrictions and their details, see Attachments 1 to 12.

Attachment 1:	Debugging Tool Usage Restrictions
Attachment 2:	Usage Restrictions on Code Generator for 78K0R/Kx3
Attachment 3:	CA78K0 Usage Restrictions
Attachment 4:	CA850 Usage Restrictions
Attachment 5:	CX Usage Restrictions
Attachment 6:	CX Utility Usage Restrictions
Attachment 7:	Stack Usage Tracer Usage Restrictions
Attachment 8:	Usage Restrictions on Simulator for 78K0R/Kx3
Attachment 9:	78K0R Build Tool CA78K0R Usage Restrictions
Attachment 10:	Restrictions on Using with Real-time OS
Attachment 11:	Building Tool Usage Restrictions
Attachment 12:	Programming Tool Usage Restrictions

6. Document revision history

Integrated Development Environment Package CubeSuite - Usage Restrictions

Document Number	Date Issued	Description
ZBG-CD-09-0007	January 21, 2009	1st edition
ZBG-CD-09-0013	February 18, 2009	Addition of restriction on debugging tools (No. 9)
ZBG-CD-09-0023	May 11, 2009	Addition of restriction on debugging tools (No. 10) Addition of restrictions on CA78K0R (No. 27 and No. 28) Addition of restriction on using with real-time OS (No. 1)
ZBG-CD-09-0029	June 11, 2009	Addition of restrictions on CA850 (No. 104 to No. 110)
ZBG-CD-09-0037	July 15, 2009	Addition of restriction on debugging tools (No. 11) Addition of restrictions on building tools (No. 1 and No. 2)
ZBG-CD-09-0052	September 7, 2009	Addition of restriction on debugging tools (No. 12)
ZBG-CD-09-0059	November 9, 2009	Addition of restriction on programming tools (No. 1)
ZBG-CD-10-0011	March 11, 2010	Addition of restriction on debugging tools (No. 13)
ZBG-CD-10-0013	March 18, 2010	Addition of restrictions on building tools (No. 14 and No. 15)
ZBG-CD-10-0020	May 20, 2010	Addition of restrictions on building tools (No. 19 and No. 20) Addition of restriction on CX (No. 6) Addition of restriction on CA78K0R (No. 29)
ZBG-CD-10-0025	August 16, 2010	Addition of restrictions on CA78K0 (No. 65 to No. 74) Addition of restriction on CX utility (No. 1)
ZMT-F35-10-0003	November 11, 2010	Addition of restriction on CA78K0 (No. 1) Addition of restrictions on CA850 (No. 111 and No. 112) Addition of restriction on CX (No. 7)
ZMT-F35-10-0011	March 23, 2011	Addition of restrictions on CA78K0 (No. 75 and No. 76) Addition of restrictions on CX (No. 8 and No. 9) Addition of restriction on CA78K0R (No. 30, No. 31, No. 32, No. 33, and No. 34)

Debugging Tool Usage Restrictions

1. Product History

No.	Affected Tool	Affected Device	Usage Restrictions	CubeSuite Package									
				V1.00	V1.10		V1.12				V1.20		
				CubeSuite									
				V1.00	V1.10	V1.11	V1.12	V1.13	V1.14	V1.15	V1.20	V1.21	1.30/ 1.31
1	MINICUBE2	V850	Restriction on configuration of communication method for MINICUBE2	×	×	×	×	×	×	×	○	○	○
2	All	78K0R	Restriction on watch display for pointer variables	×	×	×	×	×	×	×	○	○	○
3	All	78K0	Restriction on stack-trace display	×	×	×	×	×	×	×	×	×	×
4	All	78K0	Restriction on stepping into main bank	×	×	×	×	×	×	×	×	×	×
5	All	78K0, 78K0R	Restriction on local-variable display	×	×	×	×	×	×	×	×	×	×
6	All	78K0	Restriction on disassemble window	×	×	×	×	×	×	×	×	×	×
7	All	All	Breakpoint and other settings become invalid	×	×	×	×	×	×	×	×	×	×
8	IECUBE, MINICUBE2	V850	Restriction on conflicting breakpoints	×	×	×	×	×	×	×	×	×	×
9	All	78K0, 78K0R	Restrictions for CPU Register Display	×	○	○	○	○	○	○	○	○	○
10	All	All	Restriction on operations on the main panel	×	×	○	○	○	○	○	○	○	○
11	Simulators	78K0R	Restriction on specifying breakpoints	×	×	×	○	○	○	○	○	○	○
12	IECUBE	All	Restriction on saving trace data	×	×	×	○	○	○	○	○	○	○
13	MINICUBE2	78K0R	Restriction on wide voltage mode	×	×	×	×	×	×	○	○	○	○
14	MINICUBE2	78K0	Restriction on the DMM feature	×	×	×	×	×	×	×	○	○	○
15	IECUBE, simulators	All	Restriction on specifying trace points	×	×	×	×	×	×	×	○	○	○
16	MINICUBE	V850E2M	Restrictions for RRM and DMM functions	–	–	–	–	–	–	–	×	×	×
17	MINICUBE	V850E2M	Restrictions for flash options	–	–	–	–	–	–	–	×	×	×
18	MINICUBE	V850E2M	Restrictions for variable values during step execution	–	–	–	–	–	–	–	×	×	×
19	MINICUBE2, simulators	78K0R	Restriction when executing an instruction (mov, movw, etc) that uses word[BC] as an operand	×	×	×	×	×	×	×	×	○	○
20	IECUBE, MINICUBE2	78K0R	Error in the settings for stopping peripheral emulation upon breaking	×	×	×	×	×	×	×	×	○	○

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 Restriction on configuration of communication method for MINICUBE2

[Applies to]

MINICUBE2, V850

[Description]

In the debugging tool properties, on the [Connection Properties] tab, select the number in [Connection between MINICUBE2 and target board] with reference to the table below.

Target Device	Setting
V850ES/Kx1+, V850ES/Kx2	UART0: 0, CSI0: 1
V850ES/Jx2, V850ES/Jx3, V850ES/Jx3-L	UARTA0: 0, CSIB0: 1, CSIB3: 2
V850ES/Hx2, V850ES/IE2, V850E/MA3 V850E/IA3, V850E/IA4, V850E/Ix3	UARTA0: 0, CSIB0: 1
V850ES/Jx3-H, V850ES/Jx3-U	UARTC0: 0, CSIF0: 1, CSIF3: 2
V850ES/Fx3, V850ES/Fx3-L	UARTD0: 0, CSIB0: 1

[Workaround]

None.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.20.

No. 2 Restriction on watch display for pointer variables

[Applies to]

All debugging tools, 78K0R

[Description]

If you have performed optimization, then the watch display of pointer variables may be invalid.

[Workaround]

If you will mainly be performing debugging, use the prioritize debugging flag (-qg).

[Fix]

This issue has been corrected in CubeSuite Ver. 1.20.

No. 3 Restriction on stack-trace display

[Applies to]

All debugging tools, 78K0

[Description]

The stack-frame display function may fail to correctly display up to the **main** function if a function is used that does not push the frame pointer (HL) onto the stack (e.g. **noauto** or **norec** function), or if the memory bank is used.

Additionally, a free-run state may occur if a return is executed from a function that does not push the frame pointer (HL) onto the stack (e.g. **noauto** or **norec** function), or if a memory bank function is used.

[Workaround]

None.

[Fix]

In planning

No. 4 Restriction on stepping into main bank

[Applies to]

All debugging tools, 78K0

[Description]

If you step into a user-defined library function or function without debugging information in the memory bank at the source level, execution will break in the bank-switching library.

[Workaround]

None.

[Fix]

In planning

No. 5 Restriction on local-variable display

[Applies to]

All debugging tools, 78K0, 78K0R

[Description]

Local variables outside the scope of the current PC are not displayed correctly in the stack trace panel.

[Workaround]

None.

[Fix]

In planning

No. 6 Restriction on disassemble window

[Applies to]

All debugging tools, 78K0

[Description]

When displaying instructions in the common area in the disassemble window, if the displayed instruction uses a symbol in the memory bank area, a symbol from a different bank may be displayed.

[Workaround]

There is no workaround.

[Fix]

In planning

No. 7 Breakpoint and other settings become invalid

[Applies to]

Common to all debugging tools and devices

[Description]

If you differentiate function or variable names by leading underscores, then the debugger may misrecognize them, and convert symbols or make breakpoint settings invalid.

This applies for cases like when you have two functions, one named **_reset** and the other named

__reset.

[Workaround]

Do not use leading underscores alone to differentiate similar function or variable names.

[Fix]

In planning

No. 8 Restriction on conflicting breakpoints

[Applies to]

IECUBE/MINICUBE/MINICUBE2, V850

[Description]

If there is a conflict between a software breakpoint and one of the following hardware breakpoints, the PC value may be invalidly corrected.

(1) Trace full break

(2) Non-map break

(3) Write-protect break

(4) Illegal I/O access break

(5) Forced break due to Stop button press

(6) Event break (hardware break)

(7) Timeout break

[Workaround]

Use a hardware break instead of a software break.

[Fix]

In planning

No. 9 Restrictions for CPU Register Display

[Applies to]

All debugging tools, 78K0, 78K0R

[Description]

(1) Toggling the register bank selection flag (RBS0, RBS1) of the control register will not cause the changed bank value to be displayed in the current register bank display on the CPU register panel. Bank 0 will always be displayed. Bank 0 is also always shown when a general-purpose register is added to the watch panel.

(2) If the register bank selection flag (RBS0, RBS1) of the control register is toggled, the HL display of general-purpose registers will be invalid.

[Workaround]

The following apply to restrictions (1) and (2), respectively:

(1) To check a general-purpose register, view that register on the CPU register panel.

(2) There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.10.

No. 10 Restriction on operations on the main panel

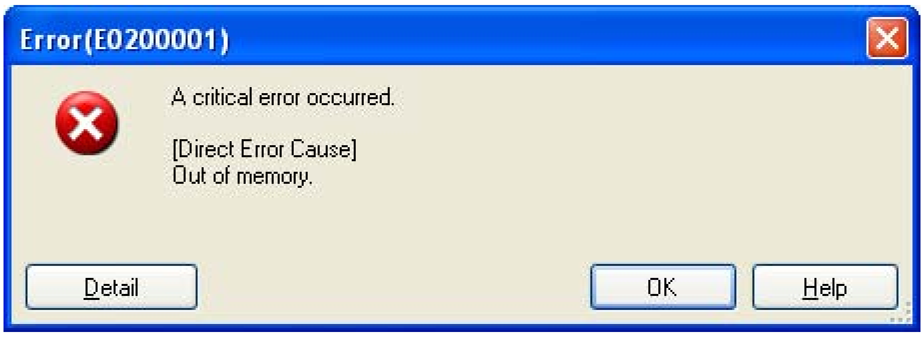
[Applies to]

Common to all debugging tools and devices

[Description]

If the operations listed below are executed while the **Source** tab is active on the main panel, the host memory that CubeSuite needs as a buffer is not released. Consequently, the error message shown in the message box below occurs due to insufficient memory and CubeSuite might not operate normally.

- Re-downloading the program
- Editing the source
- Scrolling on the source
- Changing the trace memory size (applies only to the simulator)


Error(E0200001)

A critical error occurred.

[Direct Error Cause]
Out of memory.

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.11.

No. 11 Restriction on specifying breakpoints

[Applies to]

Simulators, 78K0R

[Description]

If a program is executed while the following three conditions are satisfied, execution stops at the specified breakpoint, regardless of whether the condition for skipping is satisfied:

(1) A breakpoint is specified for the instruction following a conditional skip instruction (SKC, SKNC, SKZ, SKNZ, SKH, or SKNZ).

(2) No breakpoint is specified for the skip instruction described in (1).

(3) The **Watch**, **Memory**, **Disassemble**, **Local Variables**, or **Call Stack** panel is displayed.

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.12.

No. 12 Restriction on saving trace data

[Applies to]

IECUBE , All devices

[Description]

When saving trace data using IECUBE as the debug tool,host PC may be restarted, or the Windows error (blue screen error) may be displayed.

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.12.

No. 13 Restriction on wide voltage mode

[Applies to]

MINICUBE2, 78K0R

[Description]

On the **Property** panel of the debug tool, even if [Yes] is selected for the [Use wide voltage mode] property in the [Flash] category on the [Connect Settings] tab, the wide voltage mode is not set. Instead, the full speed mode is set.

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.15.

No. 14 Restriction on the DMM feature

[Applies to]

MINICUBE2, 78K0

[Description]

On the **Property** panel of the debug tool, even if [Yes] is selected for the [Access by stopping execution] and [Set update display during the execution automatically] properties in the [Access Memory While Running] category on the [Debug Tool Settings] tab, the values written to the **Watch** or **Memory** panel during execution are not applied.

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.20.

No. 15 Restriction on specifying trace points

[Applies to]

IECUBE, simulators, and all devices

[Description]

If three or more points are specified for tracing variables, an error occurs when the project is next loaded or a load module is downloaded, and the specified trace points might be deleted.

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.20.

No. 16 Restrictions for RRM and DMM functions

[Applies to]

MINICUBE, V850E2M

[Description]

The RRM and DMM functions are not available. Using them will cause a parameter error.

[Workaround]

There is no workaround.

[Fix]

In planning

No. 17 Restrictions for flash options

[Applies to]

MINICUBE, V850E2M

[Description]

The security settings and boot-block cluster settings of the Flash Option Settings property are not supported. Any settings made to them are ignored.

[Workaround]

There is no workaround.

[Fix]

In planning

No. 18 Restrictions for variable values during step execution

[Applies to]

MINICUBE, V850E2M

[Description]

If a variable that persists across a function call is assigned to a register, then if that register is referenced in the debugger between the time immediately prior to the call of the function until after the call, an invalid value may be displayed.

[Workaround]

There is no workaround.

[Fix]

In planning

No. 19 Restriction when executing an instruction (mov, movw, etc) that uses word[BC] as an operand

[Applies to]

MINICUBE2, simulators, 78K0R

[Description]

When an assembler instruction that has "general-purpose register + offset"note as the access addressing operand is executed, and this access addressing operand exceeds 10000H, the result of execution is incorrect.

- MINICUBE2: The restriction occurs only when you use step execution.
- Simulator: The restriction occurs when you use execution or step execution.

Note In case the instruction which includes below as the operand is executed.
[HL+byte], [DE+byte], [SP+byte], word [B], word [C], word [BC], [HL+B], [HL+C] (not including ES:[HL+byte], ES:[DE+byte], ES:word[B], ES:word[C], ES:word[BC], ES:[HL+B], ES:[HL+C])

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.21.

No. 20 Error in the settings for stopping peripheral emulation upon breaking

[Applies to]

IECUBE, MINICUBE2, 78K0R

[Description]

On the [Debug Tool Settings] tab in the debug tool properties, the functions of the settings for stopping peripheral emulation, which are under [Break], are reversed.

- If "Yes" is selected for [Stop emulation of timer group when stopping], [Stop emulation of serial group when stopping] is set to "Yes".
- If "Yes" is selected for [Stop emulation of serial group when stopping], [Stop emulation of timer group when stopping] is set to "Yes".

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.21.

Usage Restrictions on Code Generator for 78K0R/Kx3

1. Product History

No.	Usage Restrictions	CubeSuite Package							
		V1.00	V1.10		V1.12		V1.20		
		Code Generator for 78K0R/Kx3							
		V1.00	V1.10	V1.11	V1.21	V2.00			
1	Restrictions for IIC0 serial interface	×		○					

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 Restrictions for IIC0 serial interface

[Description]

The procedure for initializing the IIC0 serial interface is different from the procedure described in the user's manual of the device.

The following functions in the output code pertain to the IIC0 serial interface.

- IIC0_Init( ) --- Initialization
- IIC0_MasterSendStart( ) --- Master sending
- IIC0_MasterReceiveStart( ) --- Master receiving
- IIC0_SlaveSendStart( ) --- Slave sending
- IIC0_SlaveReceiveStart( ) --- Slave receiving

[Workaround]

Correct the sequence for setting the output latch to 0 and enabling operation described in pages 60 and 61 as follows.

This is described in the output code for master sending.

```
void IIC0_Init(void)
{
            :
    /* Set INTIIC0 low priority */
    IICPR10 = 1U;
    IICPR00 = 1U;
    /* Set SCL0 pin */
    P6 &= 0xFEU;
    /* Set SDA0 pin */
    P6 &= 0xFDU;
    IICCL0 = _00_IIC0_CLOCK0 | _00_IIC0_FILTER_OFF;
    IICX0 = _00_IIC0_EXPENSION0;
    SVA0 = _10_IIC0_MASTERADDRESS;
    STCEN = 1U;
    IICRSV = 1U;
```

move)
setting the output latch to 0

```
        SPIE0 = 0U;
        WTIM0 = 1U;
        ACKE0 = 1U;                                              move)
        IICMK0 = 0U;                                             enable operation
        IICE0 = 1U;         /* Enable IIC0 operation */
        /* Set SCL0 pin */
        PM6 &= 0xFEU;                          delete)
        ~~P6 &= 0xFEU;~~
        /* Set SDA0 pin */
        PM6 &= 0xFDU;                          delete)
        ~~P6 &= 0xFDU;~~
    }

    MD_STATUS IIC0_MasterSendStart(parameter)
    {
        MD_STATUS status = MD_OK;              delete)
        ~~IICE0 = 1U;~~
        LREL0 = 1U;
            :
    }
```

[Action]

This issue has been corrected in code generator Ver. 1.11 for the 78K0R/Kx3.

CA78K0 Usage Restrictions

1. Product History

(1) List of restrictions for assembler part

No.	Usage Restrictions	CubeSuite Package							
		V1.00	V1.10		V1.12		V1.20		V1.40
		CA78K0							
		V1.00						V1.10	V1.11
1	An error occurs if a control statement is crossed in a structured assembly language description.	×						×	×
5	The assembler performs illegal processing if the label receiving the effect of optimization is described in the *saddr* part when an EQU definition is performed for a bit symbol with the value *saddr.bit*.	×						×	×
9	Concatenate (&) will not be linked if the macro quasi directive IRP is nested.	–						–	–

×: Applicable, ○: Not applicable, –: Not relevant, *****: Check tool available

Remark The item numbers are the same as those for the RA78K0 and are not sequential.

(2) List of restrictions for compiler part (1/2)

No.	Usage Restrictions	CubeSuite Package							
		V1.00	V1.10		V1.12		V1.20		V1.40
		CA78K0							
		V1.00						V1.10	V1.11
12	Linking a variable with the same name to a variable declared *extern* in the block is sometimes illegal	×						○	○
14	A multidimensional array with an undefined size may not operate properly.	×						○	○
16	Bit fields with type *signed* are handled as unsigned bit fields.	×						×	×
24	W0503 is output when the array name of an automatic variable is referenced.	×						○	○
41	When initializing an array whose size is not defined when elements in the initializer braces are enclosed inconsistently, the size of the secured area becomes invalid.	×						○	○
43	Output conversion on I/O functions in the standard libraries causes illegal behavior.	×						×	×
44	The size of the minimum value (-32768) of types *int/short* is 4.	×						×	×
45	An error is output if a function name or a function pointer is described as the second and third operands of a conditional operation, and then the function is called.	×						○	○
47	If the parameter type and the type of the identifier in a function definition do not match, an error is output.	×						×	×
48	In an identifier list in a function definition, a parameter that is not declared is not handled as type *int*, and an error results.	×						×	×
49	The # operator cannot be expanded correctly.	×						×	×
65	Invalid code might be output when referencing 1-byte data pointed to by a pointer to which `++` or `--` has been suffixed.	×*						○	○
66	Invalid code is output if the last element of an initializer list for a `char`, `signed char`, or `unsigned char` array is a character string and one or more constants or character constants are placed before the character string.	×*						○	○
67	Invalid code is output if referencing a pointer that has an offset obtained by subtracting one pointer from another.	×*						○	○
68	Invalid code is output if the `-qc` option has not been specified (sign extension is specified to be `int` type).	×*						○	○
69	Invalid code is output as the result of the BCD operation function `adbcdw` or `sbbcdw`.	×						○	○
70	An error occurs if the `-ng` option is specified and a branch instruction is used in a function that includes the `asm` statement.	×						○	○

×: Applicable, ○: Not applicable, –: Not relevant, *: Check tool available

Remark The item numbers are the same as those for the CC78K0 and are not sequential.

(2) List of restrictions for compiler part (2/2)

No.	Usage Restrictions	CubeSuite Package							
		V1.00	V1.10		V1.12		V1.20		V1.40
		CA78K0							
		V1.00						V1.10	V1.11
71	The line number is not output for a statement that immediately follows a nested `if` statement and is outside that statement's conditional block.	×						○	○
72	Invalid code is output when a value is assigned to a `long` variable in an interrupt function.	×*						○	○
73	Bank function calling code might not be output.	×*						○	○
74	Invalid code might be output if using a 1-byte parameter or `auto` variable for a `norec` function.	×*						○	○
75	Macro expansion using the `##` operator results in an error	×						×	×
76	Symbol information for an interrupt function is not output to the assembler source	×						×	×

×: Applicable, ○: Not applicable, –: Not relevant, *: Check tool available

Remark The item numbers are the same as those for the CC78K0 and are not sequential.

(3) File missing from the installation package

No.	Usage Restrictions	CubeSuite Package							
		V1.00	V1.10		V1.12		V1.20		V1.40
		CA78K0							
		V1.00						V1.10	V1.11
1	[Set Variables Relocation] tab error	–						×	○

×: Applicable, ○: Not applicable, –: Not relevant, *: Check tool available

2. Restriction Details

(1) Details of restrictions for assembler part

No. 1 A multidimensional array without the size defined may exhibit incorrect behavior

[Description]

If a control statement is divided or crossed by code between *#ifdef* and *#endif*, an error occurs if *#ifdef* is true.

Example:

```
    switch(mode)
#ifdef    stsw        <------  Between #ifdef and #else or #endif
    case 1:           <------  Between case and next case/default/ends
            break
#endif
    default:
            break
ends
```

[Workaround]

Nesting will not cause an error. Rewrite the source so that the scopes of the control statements do not cross.

Example:

```
#ifdef stsw           <------  Between #ifdef and #else or #endif
  switch(mode)
       case 1:        <------  Between case and next case/default/ends
           break
       default:
           break
  ends
#else
  switch(mode)
       default:
          break
  ends
#endif
```

[Action]

Correction is not planned. Regard this item as a specification.

No. 5 The assembler performs illegal processing if the label receiving the effect of optimization is described in the *saddr* part when an EQU definition is performed for a bit symbol with the value *saddr.bit*.

[Description]

The assembler performs illegal processing if the label receiving the effect of optimization is described in the *saddr* part when an EQU definition is performed for a bit symbol with the value *saddr.bit*.

(1) When *sadder.bit* is 0FD20H, path 1 of a label is outside the area, and path 2 is inside the area, an error is output in path 1 for the EQU definition line, but not in path 2. At this time, the object is created but it is incorrect.

(2) When *saddr* is 0FF1FH, path 1 of a label is inside the area, and path 2 is outside the area, no error is output in path 1 for the EQU definition line, while an error is output in path 2. The following assembly error will be output for a label that is defined after this EQU symbol has been referenced.

[F410 Phase error]

When this label is referenced, the object becomes incorrect.

[Workaround]

None.

[Action]

Correction is not planned. Regard this item as a specification.

No. 9 Concatenate (&) will not be linked if the macro quasi directive IRP is nested.

[Description]

Nesting IRP that includes the character string concatenation symbol "&" will render the macro expansion results invalid, since the parameter is not converted.

Example:

```
IRP   ZZZ,<1,2,3>
      IRP   XXX,<4,5,6>
            LABEL&ZZZ&XXX:    ; Expansion results are invalid.
      ENDM
ENDM
```

[Workaround]

Do not use the "&" operator when nesting IRP.

[Action]

Correction is not planned. Regard this item as a specification.

(2) Details of restrictions for compiler part

No. 12 Linking a variable with the same name to a variable declared *extern* in the block is sometimes illegal

[Description]

Linking a variable with the same name to a variable declared *extern* in the block is illegal in any of the following cases.

(1) A variable declared with *extern* in a block and a variable subsequently declared with *static* outside the block have the same name. Since no error occurs and linking is not performed, an illegal code is output when this variable is referenced.

Example:

```
void f ( void ) {
        extern int i ;
        i = 1 ;                 /* Illegal code output */
}
static int i ;
```

(2) A variable declared with *extern* in a block and a variable declared subsequently without *static* outside the block have the same name. In this case, linking is not performed, and an illegal code is output.

Example:

```
void f ( void ) {
        extern int i ;
        i = 1 ;                 /* Illegal code output */
}
int i ;
```

(3) When a variable declared with *extern* in a block, and a variable declared previously outside the block without extern have the same name, and an automatic variable declared in a block containing the block with the variable declared with extern also has the same name.
In this case, the variable outside the block and the variable declared with extern in the block are not linked, and an illegal code is output.

Example:

```
int i = 1 ;
void f ( void ) {
       int i ;
       {
              extern int i ;
              i = 1 ;    /* Illegal code output */
       }
}
```

(4) A variable declared with *extern* in a block and a variable declared with *extern* in another block have the same name. In this case, linking is not performed, and an illegal code is output.

Example:

```
void f1( void ) {
        extern int i ;
        i = 2 ;
}
void f2( void ) {
```

```
            extern int i ;
            i = 3 ;
    }
```

[Workaround]

There is not workaround.

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 14 A multidimensional array without the size defined may exhibit incorrect behavior

[Description]

A multidimensional array with an undefined size may not operate properly.

Example 1:

```
        char c [ ] [3] = { { 1 }, 2, 3, 4, 5 } ;        /* Illegal code */
```

Example 2:

```
        qchar c [ ] [2] [3] = { "ab", "cd", "ef" } ; /* Error (E0756) */
```

[Workaround]

Define the size of the multidimensional array.

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 16 Bit fields with type signed are handled as unsigned bit fields.

[Description]

Bit fields with type signed are handled as unsigned bit fields.

[Workaround]

None.

[Action]

Correction is not planned. Regard this item as a specification.

No. 24 W0503 is output when the array name of an automatic variable is referenced.

[Description]

W0503 is output when the array name of an automatic variable without initialization is referenced.

W0503 Possible use of "variable name" before definition

Note:

"Initialization" means a declaration such as "int a[2] = {0,0};". It does not include assignment expressions such as "a[0] = 0; a[1] = 0;". An "array name" is the 'a' in "int a[2]". It does not include *a[0]*, *a[1]*, or *&a[0]*.

Example:

```
void func(void)
{
        int a[2];
        int *b;
        a[0] = 0;
        a[1] = 0;
        b = a;      /* W0503 is output on this line */
}
```

[Workaround]

If W0503 is output, check the relevant location. Ignore the message if initialization has been performed in the statement.

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 41 When initializing an array whose size is not defined when elements in the initializer braces are enclosed inconsistently, the size of the secured area becomes invalid.

[Description]

When initializing an array whose size is not defined when elements in the initializer braces are enclosed inconsistently, the size of the secured area becomes invalid.

Example:

```
struct t {
        int a;
        int b;
} x[ ] = {1, 2, {3, 4}};
```

[Workaround]

You can avoid this by doing either of the following:

(1) Unify the brace enclosing method.

```
struct t {
        int a;
        int b;
} x[ ] = {{1, 2}, {3, 4}};
```

(2) Define the size of the array.

```
struct t {
        int a;
        int b;
} x[2] = {1, 2, {3, 4}};
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 43 Output conversion on I/O functions in the standard libraries causes illegal behavior.

[Description]

When output conversion is performed for the *printf, sprintf, vprintf*, and *vsprintf* functions, operation will become illegal under the following conditions.

If precision is specified as ".2" for the *d, i, o, u, x* or *X* conversion specifier, the 0 flag will not be ignored.

Example:

```
#include <stdio.h>
void func()
{
        printf("%04.2d\n", 77);
}
```

Remark Illegal operation: "0077"
Correct operation: " 77"

For the *g*, and *G* conversion specifiers, the result is "specified precision + 1".

Example:

```
#include <stdio.h>
void func()
{
        printf("%.2g", 12.3456789);
}
```

Remark Illegal operation: "12.3"
Correct operation: "12"

[Workaround]

None.

[Action]

Correction is not planned. Regard this item as a specification.

No. 44 The size of the minimum value (-32768) of types *int/short* is 4

[Description]

The size of the minimum value (-32768) of types *int/short* is 4.

Example:

```
int x;
void func()
{
        x = sizeof(-32768);
}
```

Remark Illegal operation: The value of x is 4
Correct operation: The value of x is 2

[Workaround]

Write as (-32767-1).

[Action]

Correction is not planned. Regard this item as a specification.

No. 45 An error is output if a function name or a function pointer is described as the second and third operands of a conditional operation, and then the function is called.

[Description]

Error E0307 is output if a function name or a function pointer is described as the second and third operands of a conditional operation, and then the function is called.

Example:

```
void f1(), f2();
int x;
void func()
{
        (x ? f1 : f2)();
}
```

[Workaround]

Use an *if* statement instead of the conditional operation.

```
(x ? f1 : f2)();
        ↓
if (x) {
        f1();
}
else {
        f2();
}
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 47 If the parameter type and the type of the identifier in a function definition do not match, an error is output.

[Description]

Because argument promotion is not performed for the type of an identifier in a function definition, the parameter type and the type of the identifier in the function definition do not match, thus causing the E0747 error.

Example:

```
int fn_char(int);
int fn_char(c)
char c;
```

```
{
        return 98;
}
```

[Workaround]

Make sure that the type of the parameter matches that of the identifier in the function definition.

[Action]

Correction is not planned. Regard this item as a specification.

No. 48 In an identifier list in a function definition, a parameter that is not declared is not handled as type *int*, and an error results.

[Description]

In an identifier list in a function definition, a parameter that is not declared is not handled as type *int*, thus causing the E0706 error.

Example:

```
void func(x1, x2, f, x3, lp, fp)
int (*fp)( );
long *lp;
float f;
{
        :
}
```

[Workaround]

Declare all parameters in a function definition.

[Action]

Correction is not planned. Regard this item as a specification.

No. 49 The # operator cannot be expanded correctly.

[Description]

Expansion will not be performed correctly under either of the following conditions.

1. [""] cannot be expanded correctly with the # operator, causing a compile-time error.

Example for condition 1:

```
#include <string.h>
#define str( a) (# a)
int x;
void func()
{
        if (strcmp(str(""), "\"\"") == 0) x++;
}
```

Remark: Illegal operation: Compile-time error

Correct operation: if (strcmp(("\"\"") , "\"\"") == 0) x++;

2. Macros that contain a # operator and a nested structure cannot be expanded correctly.

Example for condition 2:

```
#define str(a) #a
#define xstr(a) str(a)
#define EXP 1
char *p;
void func()
{
        p = xstr(12EEXP);
}
```

Remark: Illegal operation: "p = ("12E1");"

Correct operation: "p = ("12EEXP");"

[Workaround]

None.

[Action]

Correction is not planned. Regard this item as a specification.

No. 65 Invalid code might be output when referencing 1-byte data pointed to by a pointer to which ++ or -- has been suffixed.

[Description]

Invalid code might be output if, immediately after referencing 1 byte to which a pointer points, the increment or decrement operator is suffixed to the pointer, and the memory to which the pointer points is referenced again.

Example:

```
void func()
{
          unsigned char tmp, *src, dst;
                 *src = tmp + 0x80;
                 dst += *src++;
}
```

[Workaround]

Do either of the following:

(1) Divide the assignment and increment operations into different expressions:

```
dst += *src;
src++;
```

(2) Use temporary variables to divide the expressions:

```
tmp2 = tmp + 0x80;
*src = tmp2;
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 66 Invalid code is output if the last element of an initializer list for a `char`, `signed char`, or `unsigned char` array is a character string and one or more constants or character constants are placed before the character string.

[Description]

If the last element of an initializer list for a `char`, `signed char`, or `unsigned char` array is a character string and one or more constants or character constants are placed before the character string, no error is output, but invalid code might be output.

Only string literals or constants can be used to initialize `char`, `signed char`, or `unsigned char` arrays.

Example:

```
[.c]
const char a1[ ] = {0x01, “abc”};
char *const TBL[3] = { a1 };
char *ptr1;
void func()
{
              ptr1 = TBL[0];
}

[.asm]
@@CNST        CSEG        UNITP
_a1:          DB          01H      ; 1
              DB          'ab'
_TBL:         DW          _a1      ; _TBL is an odd address
              DB          (4)

@@DATA        DSEG        UNITP
_ptr1:        DS          (2)

; line     1 : const char a1[] = { 0x01, "abc" };
; line     2 : char *const TBL[3] = { a1 };
; line     3 : char *ptr1;
; line     4 : void func()
; line     5 : {

@@CODEL       CSEG
_func:
; line     6 : ptr1 = TBL[0];
              movw   ax,!_TBL      ; Reference an odd address
              movw   !_ptr1,ax
```

[Workaround]

Correctly specify the initial value.

```
[.c]
const char a1[ ] = { 0x01, 'a', 'b', 'c', '\0' };
char *const TBL[3] = { a1 };
char *const *ptr1;
void func()
```

```
    {
                    ptr1 = TBL[0];
    }
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 67 Invalid code is output if referencing a pointer that has an offset obtained by subtracting one pointer from another.

[Description]

Invalid code is output if all the following conditions are satisfied:

(1) A pointer to which an offset is added is referenced.

(2) The offset mentioned in (1) is the result of subtracting one pointer from another.

(3) A pointer mentioned in (2) has an offset.

Example:

```
[.c]
void main(void)
{
      char  *p1;
      char  *p2;
      char  *p3;
           *p1 = *(p2 + (p1 - (p3 + 2)));
}
```

[Workaround]

Divide the expression.

```
[.c]
void main(void)
{
      char  *p1;
      char  *p2;
      char  *p3;
      int tmp;                  /* Prepare a temporary variable */
      tmp = (p1 - (p3 + 2));   /* Assign the value to the temporary variable */
      *p1 = *(p2 + tmp);
}
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 68 Invalid code is output if the `-qc` option has not been specified (sign extension is specified to be `int` type).

[Description]

Invalid code is output if all the following conditions are satisfied:

(1) The -qc option is not specified (sign extension is specified to be the int type)

(2) One of the following combinations of operands (regardless of whether they are right or left operands) is multiplied:

- An sreg unsigned char to which a constant from 0 to 255 is assigned and a constant from 0 to 255
- More than one sreg unsigned char to which a constant from 0 to 255 is assigned
- An sreg unsigned char to which a constant from 0 to 255 is assigned and an sreg char or signed char to which a constant from 0 to 127 is assigned

(3) The multiplication result is 256 or larger (which cannot be expressed as an unsigned char).

(4) The operation result is handled as an int.

In the following example, the value of Temp1 should be 0x1FE but it is output as 0xFE.

Example:

```
[.c]
unsigned int   Temp1;
      _sreg unsigned char  Byte1;
      Temp1 = (Byte1 = 255) * 2;
```

[Workaround]

Type-cast the variable to an int or unsigned int.

```
[.c]
unsigned int   Temp1;
unsigned char  Byte1;

Temp1 = (unsigned int) (Byte1 = 255) * 2;
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 69 Invalid code is output as the result of the BCD operation function adbcdw or sbbcdw.

[Description]

Invalid code is output if one of the following conditions is satisfied while the BCD operation function adbcdw or sbbcdw is used:

(1) The return value of adbcdw or sbbcdw is assigned to an array or pointer.

(2) Another operation is executed before assigning the return value of adbcdw or sbbcdw to a temporary variable.

(3) A temporary variable to which the return value of adbcdw or sbbcdw has been assigned is used as is for other operations such as a conditional expression.

Example:

```
[.c]
void func()
{
      unsigned int   tmp1[3];
      unsigned int   tmp2, tmp3;
```

```
        unsigned int   a = 10, i = 0, *p;

        tmp1[i] = adbcdw(80, 50);                /* (1) */
        tmp2 = adbcdw(80, 50) + (a + 1);         /* (2) */
        if ((tmp3 = adbcdw(80, 50) == *p ) …     /* (3) */
                ...
}
```

[Workaround]

Prepare a function used only for calling `adbcdw` and `sbbcdw` and call the function. Use the same parameters and return values as those of `adbcdw` and `sbbcdw`.

Example:

```
[.c]
unsigned int adbcdw_new(unsigned int a, unsigned int b)
{
        return adbcdw(a, b);
}
void func()
{
unsigned int   tmp1[3];
unsigned int   tmp2, tmp3;
unsigned int   a = 10, i = 0, *p;

tmp1[i] = adbcdw_new(80, 50);                /* (1) */
tmp2 = adbcdw_new(80, 50) + (a + 1);         /* (2) */
if ((tmp3 = adbcdw_new(80, 50) == *p ) …     /* (3) */
                ...
}
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 70 An error occurs if the `-ng` option is specified and a branch instruction is used in a function that includes the `asm` statement.

[Description]

If the `-ng` option is specified and a branch instruction is used after the `asm` statement in a function, an error might occur. This error might occur if both of the following conditions are satisfied:

(1) The `asm` statement is used in a function.

(2) A statement that causes processing to branch (such as `if`, `for`, or `while`) is used in the same function.

However, if the above conditions are satisfied, the error occurs during assembly and the object module file is not generated. If no error occurs, this restriction does not apply.

Example:

```
[.c]
unsigned int i;
void func()
{
     do {
         __asm("\t DB  (1000)");
         i++;
     } while ( i < 10) ;
}
```

[Workaround]

Change to the `-g` option.

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 71 The line number is not output for a statement that immediately follows a nested `if` statement and is outside that statement's conditional block.

[Description]

If all of the conditions below are satisfied, the line number might not be output for a statement that immediately follows a nested `if` statement and is outside that statement's conditional block. However, the output code is correct. A break point cannot be specified for a statement for which the line number is not output.

(1) There are at least three levels of nested `if` statements.

(2) An `else` statement in a higher nested level has a larger line number than a nested `if` statement.

(3) At least one statement immediately follows an `if` statement in a higher nested level.

Example:

```
[.c]
int  f0, f1, f2, f3;
int  g0, g1, g2;
void func(void)
{
    if (f0){
        if (f1){           /* if statement in nested level 1 */
           g2 = 5;
        }
        else if (f2){      /* if statement in nested level 2 */
            g2 = 4;
        }
        else if (f3){      /* if statement in nested level 3 (condition (1)) */
           g2 = 3;
        }
        else {
           g2 = 2;
        }
        g0 = 0x1234;       /* A statement immediately follows the if statement
                              in nested level 1 (condition (3)).*/
        g1 = 0x5678;
    }
    else {            /* This else statement has a larger line number than */
        g2 = 1;       /* the if statement in nested level 3 (condition (2)).*/
        }
  }
```

[Workaround]

Insert several blank lines before the statement that immediately follows the nested `if` statement and is outside that statement's conditional block.

For the above example, insert the lines before `g0 = 0x1234;`.

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

No. 72 Invalid code is output when a value is assigned to a `long` variable in an interrupt function.

[Description]

When the assignment of a value to a `long` variable is specified in an interrupt function, the `-ql3` option or higher is specified, the compiler generates code that calls the runtime library function `@@dels03` or `@@hlls03`, the BC register is not used for other processing in the interrupt function, and no function is called from the interrupt function, the BC register contents are corrupted.

Example:

```
[.c]
unsigned long l;
unsigned int i;
__interrupt void func()
{
     l = (unsigned long)i;
}
```

[Workaround]

Do either of the following:

(1) After the `return` statement at the end of the function (which must be added if not there), add code that increments the `long` variable.

```
[.c]
unsigned long l;
unsigned int i;
__interrupt void func()
{
           l = (unsigned long)i;
           return;
           l++;      /* Not executed */
}
```

(2) Create a dummy function that does not perform any processing, and call it from the interrupt function.

```
[.c]
unsigned long l;
unsigned int i;
void dummy { }
__interrupt void func()
```

```
        {
                    l = (unsigned long)i;
                    dummy();
        }
```

(3) Specify `-ql1` or `-ql2` as the `-ql` optimization option level.

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.
A tool used to check whether this restriction applies is available.
For details, contact a Renesas Electronics distributor.

No. 73 Bank function calling code might not be output.

[Description]

If one of the conditions below is satisfied, bank function calling code might not be output for functions allocated to the bank area by using the function information file.
In addition, `C0101: Internal error` might be output if condition *(3)* is satisfied. If this message is not output, there is no problem and the program code is not affected.

(1) A function is called by type-casting it to a function pointer.
(2) A function declared using a `typedef` type is called.
(3) The `-mf` option is specified, and a function is called using a function pointer.

Example:

```
[.c]
typedef void F(void);
typedef void (*FP)(void);
void func1(void);
F func2;
void func()
{
        ((FP)func1)();
        func2();
}
```

[Workaround]

Do not call a bank function by type-casting it to a function pointer.
Do not use a `typedef` type to declare a bank function.

```
[.c]
void func1(void);
void func2(void);
void func()
{
        func1();
        func2();
}
```

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

For *(3)*, no check is performed.

No. 74 Invalid code might be output if using a 1-byte parameter or `auto` variable for a `norec` function.

[Description]

Invalid code might be output if using a 1-byte parameter or `auto` variable for a `norec` function, and a `long` variable is dereferenced within the function.

Example:

```
[.c]
long buf[8]
norec void func(void)
{
     unsigned char c = 7;
     long *s = &buf[c-1], *d = &buf[c];
     *d = *s;
}
```

[Workaround]

Change the `norec` function to a normal function, or change the 1-byte variable to a 2-byte variable.

[Action]

This issue has been corrected in CA78K0 Ver. 1.10.

A tool used to check whether this restriction applies is available.

For details, contact a Renesas Electronics distributor.

No. 75 Macro expansion using the `##` operator results in an error

[Description]

The E0803 error might occur if the `##` operator is not followed by a function-like macro parameter, capital or small letter, or underscore (_).

In addition, using the `##` operator might cause errors other than E0803, such as E0711 or E0301.

Example 1

```
[*.c]
#define m1(x) (x ## .c1 + 23)
#define m2(x) (x ## .c1 + 122)
struct t1 {
     unsigned char c1;
} st1;
unsigned char x1, x2;
void func1()
{
```

```
            x1 = m1(st1) + 100;    /* E0803 error (NG) */
            x2 = m2(st1) + 1;      /* No error   (OK) */
}
```

Example 2

```
[*.c]
#define m3(x) (x ## 1)
unsigned char x3, uc1;
void func2()
{
            x3 = m3(uc);          /* E0711, E0301 error (NG) */
}
```

[Workaround]

Do either of the following:

(1) Do not use the ## operator.

```
#define m1(x)      (x ## .c1 + 23)
#define m2(x)      (x ## .c1 + 122)
     ↓
#define m1(x)      ((x).c1 + 23)
#define m2(x)      ((x).c1 + 122)
```

(2) Place a function-like macro parameter immediately after the ## operator.

```
#define m3(x)      (x ## 1)
unsigned char x3, uc1;
void func2()
{
     x3 = m3(uc);
}
     ↓
#define m3(x, y)     (x ## y)
unsigned char x3, uc1;
void func2()
{
     x3 = m3(uc, 1);
}
```

[Action]

This issue will be corrected in CA78K0 Ver. 1.20.

No. 76 Symbol information for an interrupt function is not output to the assembler source

[Description]

The E3405 error occurs during linking if all the following conditions are satisfied:

(1) `#pragma interrupt` is used to specify the generation of a vector table for an interrupt function.

(2) The interrupt function is not defined in the same source.

(3) The `-no` option, assembler source module file output option (`-a` or `-sa`), and debugging information output option (`-g`) are enabled.

Example

```
[*.c]
#pragma interrupt INTP0 inter
/*    Definition of this interrupt function is not subject to compilation
__interrupt void inter()
{
            ...
}
*/
```

[Workaround]

Define the interrupt function or output the object module file.

[Action]

This issue will be corrected in CA78K0 Ver. 1.20.

(3) File missing from the installation package

No. 1 [Set Variables Relocation] tab error

[Description]

On the [Set Variables Relocation] tab, if "Yes" is selected for [Output variables information file] or [Output ROM/RAM usage], the following error is displayed:

```
Launching program xxx (C:\Program Files\NEC Electronics
CubeSuite\CubeSuite\CA78K0\V1.10\Bin\vf78k0.exe) failed. (E0200002)
```

[Workaround]

There is no workaround.

[Action]

This issue has been corrected in CA78K0 Ver. 1.11.

CA850 Usage Restrictions

1. Product History

No.	Usage Restrictions	CubeSuite Package				
		V1.00 V1.10	V1.12	V1.20		V1.40
		CA850				
		V3.31	V3.41	V3.43	V3.45	V3.47
4	Restriction on precision during floating-point constant operation	×	×	×	×	×
6	Restriction on structure type conditional operator for argument of function	×	×	×	×	×
7	Restriction on indirect calling of function	×	×	×	×	×
8	Restriction on meaningless function definition	×	×	×	×	×
12	Restriction on extra ( ) in function declaration	×	×	×	×	×
21	Restriction on section allocation	×	×	×	×	×
22	Restriction on section file with variable entity in assembly source	×	×	×	×	×
23	Restriction on section file with tentative definition of external variables of same name in multiple files	×	×	×	×	×
29	Restriction on specifying optimization option	×	×	×	×	×
30	Restriction on object size at optimization	×	×	×	×	×
31	Restrictions on debugging at optimization	×	×	×	×	×
33	Restriction on address of structure member	×	×	×	×	×
34	Restriction on bit field	×	×	×	×	×
40	Restriction on referring to specific symbol and reserved symbol in C source	×	×	×	×	×
95	Restriction on floating point constants and integral type	×	×	×	×	×
96	Restriction on input conversion for I/O function in standard library	×	×	×	×	×
104	Incorrect number of loop executions	×*	○	○	○	○
105	Incorrect string literals	×*	○	○	○	○
106	Restriction involving format specifiers for the `sscanf`, `fscanf`, and `scanf` functions	×	○	○	○	○
107	Restriction involving the character string parameter for the `atoi`, `atol`, `strtol`, and `strtoul` functions	×	○	○	○	○
108	Restriction involving initialization of a structure that includes a bit field among its members	×*	○	○	○	○
109	Restriction involving nested conditionally assembled pseudo instructions	×	○	○	○	○
110	Restriction involving assignment within `switch` and `if` statements	×*	○	○	○	○
111	Restriction on incorrect compare operation optimization while casting	×*	×*	×*	×*	○
112	Restriction on incorrect move optimization of assignment sentence	×*	×*	×*	×*	○

×: Applicable, ○: Not applicable, –: Not relevant, *: Check tool available

Remark The item numbers are the same as those for the CA850 (CA703000) and are not sequential.

2. Restriction Details

No. 4 Restriction on precision during floating-point constant operation

[Description]

If a floating-point operation that may be inadvertently executed is described for compilation that involves casting to an integer, the precision drops very slightly, and the value may become illegal as a result of casting to an integer. No problem occurs if a floating point is handled as is.

Example:

```
(long)(1.12 * 100);
```

[Workaround]

Change the above statement as follows.

```
 float f = 1.12;
 (long)(f * 100);
```

Or,

```
 float f;
 (long)(f = 1.12 * 100);
```

[Action]

Correction is not planned. Regard this item as a specification.

No. 6 Restriction on structure type conditional operator for argument of function

[Description]

The correct branch code is not generated if a structure type conditional operator exists in an argument.

Example:

```
 typedef struct {int i;}S;
 S ss1, ss2;
 int j;
 void func(){
     func_call((j>10)?ss1:ss2);
 }
```

[Workaround]

Change the above statement to the *if* statement as follows.

```
 if (j > 10){
     func_call(ss1);
 }else {
     func_call(ss2);
 }
```

[Action]

Correction is not planned. Regard this item as a specification.

No. 7 Restriction on indirect calling of function

[Description]

If an expression that indirectly calls a function requires an offset, an internal compiler error occurs.

Message:

C2000: internal: gen_binary(): OP_CALL : left-child's operator is wrong

C5211: syntax error at line <num> in intermediate file

Example:

```
struct S {;
    int dummy;
    int func_body[0x100];
} sobj;
void f() {
    ((void(*)())sobj.func_body)();
fp();
}
```

[Workaround]

Separate an offset calculation expression from a calling expression as follows.

```
void f(){
    void (*fp)() = (void(*)())&sobj.func_body;
    fp();
}
```

[Action]

Correction is not planned. Regard this item as a specification.

No. 8 Restriction on meaningless function definition

[Description]

An error is not output for a meaningless function definition.

Example:

```
typedef int INTFN();
INTFN f{return(0);}
```

[Workaround]

Avoid the coding that corresponds to this restriction.

[Action]

Correction is not planned. Regard this item as a specification.

No. 12 Restriction on extra ( ) in function declaration

[Description]

A syntax error is output for an extra ( ) in a function declaration.

Example:

```
typedef int Int;
void f1((Int));
```

[Workaround]

Modify the code as follows.

```
typedef int Int;
void f1(Int);
```

[Action]

Correction is not planned. Regard this item as a specification.

No. 21 Restriction on section allocation

[Description]

In the tidata section allocation specification by the *#pragma* section directive or in “*char* type array of a structure” or “access to a *char* type member” specified by sf850 to be located in the *tidata* section, an linker error occurs if the displacement value of the *sst* instruction or *sld* instruction used to access the array or member is exceeded.

[Workaround]

Implement any of the following workarounds.

(1) Do not use the char member or *char* array of a structure that causes the error in the linker.

(2) Do not assign the char member or *char* array of a structure that causes the error in the linker to the *tidata* section.

[Action]

Correction is not planned. Regard this item as a specification.

No. 22 Restriction on section file with variable entity in assembly source

[Description]

In an application where the entity of a variable is in the assembly source and if that variable is referred to on the C source, an error occurs during linking if the section file is generated by sf850.

[Workaround]

Delete the variable in the assembly source from the section file.

[Action]

Correction is not planned. Regard this item as a specification.

No. 23 Restriction on section file with tentative definition of external variables of same name in multiple files

[Description]

If a section file is generated by sf850 when the tentative definition of external variables of the same name are in two or more files, symbols may be defined in duplicate during linking.

Message:

ld850: fatal error: symbol “_xxxx” multiply defined.

[Workaround]

If two or more tentative definitions of external variables of the same name exist, be sure to declare extern in the file that references the external variables.

[Action]

Correction is not planned. Regard this item as a specification.

No. 29 Restriction on specifying optimization option

[Description]

If the optimization level of the optimization option specified during compilation is increased, the phases that are executed during compilation (such as the optimization function and compilation function) increase. If the -Ot option is specified, the intermediate file created between these phases increases in size, causing a fatal error in some cases.

[Workaround]

Decrease the optimization level (by using -Os) and execute compilation.

[Action]

Correction is not planned. Regard this item as a specification.

No. 30 Restriction on object size at optimization

[Description]

When the optimization option is specified, the size of an object file including debug information may significantly increase.

[Workaround]

Either lower the optimization level or use the -g option, which outputs the debug information only to the file to be debugged.

[Action]

Correction is not planned. Regard this item as a specification.

No. 31 Restrictions on debugging at optimization

[Description]

When the optimization option is specified for debugging the source, the following restrictions apply.

(1) When the value of a variable is referenced, a temporary value in the middle of calculation, not the correct value, may be obtained.

(2) If part of an array, element of a structure, or user-defined pointer variable is assigned to a register, variables may be illegally displayed or modified in the variable window of the debugger.

(3) If part of an array of an automatic variable or an element of structure is not used, the area may be deleted. In this case, variables may be illegally displayed or modified in the variable window of the debugger. The stack may be destroyed when a variable is modified.

[Workaround]

There is no workaround.

[Action]

Correction is not planned. Regard this item as a specification.

No. 33 Restriction on address of structure member

[Description]

If any of the conditions below apply when structure packing is performed, data access follows the data alignment of the device and the accessed address is masked. As a result, data will be missing or discarded by accessing the address of the structure member.

Conditions:

(1) The device used does not support misalign access

(2) The device used supports misalign access but misalign access is disabled

Example:

```
struct test {
    char c;        /* offset 0 */
    int i;         /* offset 1-4 */
} test;
int *ip, i;
void func(){
    i = *ip;       /* Accessed from a masked address */
}
void func2(){
  ip = &(test.i);
}
```

[Workaround]

There is no workaround.

[Action]

Correction is not planned. Regard this item as a specification.

No. 34 Restriction on bit field

[Description]

If the width of a bit field is less than the type of a member when the bit field is accessed during structure packing, the bit field is read by the type of the member. Consequently, an area outside the object (area where there is no data) is also accessed. This access is usually executed correctly but it may be illegal if I/O is mapped.

Example:

```
struct S {
       int x:21;
} sobj; /* 3 bytes */
sobj.x = 1;
```

[Workaround]

There is no workaround.

[Action]

Correction is not planned. Regard this item as a specification.

No. 40 Restriction on referring to specific symbol and reserved symbol in C source

[Description]

Target-specific symbols such as _gp_DATA and reserved symbols such as _stext cannot be referred to in the C source.

[Workaround]

Do not use a target-specific symbol and reserved symbol in the C source.

[Action]

Correction is not planned. Regard this item as a specification.

No. 95 Restriction on floating point constants and integral type

[Description]

When a floating point type value is converted into a value of the integral type, the range that can be expressed with integer values is the signed *int* or *signed long* type area, but if the value is converted into the *unsigned int* or *unsigned long* type while the range is exceeded, a compile error may result.

`E2519: invalid has occurred at compile time.`

Example:

```
unsigned int ui = 2147483647.0;           /* OK */
unsigned int ui = 2147483648.0;           /* Error */
```

[Workaround]

Use the integer type.

Example:

```
unsigned int ui = 2147483648;
```

[Action]

Correction is not planned. Regard this item as a specification.

No. 96 Restriction on input conversion for I/O function in standard library

[Description]

When input conversion processing is performed for *printf, sprintf, vprintf*, or *vsprintf*, which are I/O functions in the standard library, the precision specified for the conversion specifier “g,G” is incremented by 1.

Example:

```
printf("%.2g", 12.3456789);
/* The result should be 12, but 12.3 is output*/
```

[Workaround]

There is no workaround.

[Action]

Correction is not planned. Regard this item as a specification.

No. 104 Incorrect number of loop executions

[Description]

If all of the following conditions are satisfied, a loop might execute the incorrect number of times:

Conditions:

(1) `-Og`, `-O`, `-Os`, or `-Ot` is specified as the optimization option.

(2) The loop counter is not declared as `volatile`.

(3) The loop termination condition involves comparing the loop counter to a constant.

(4) A constant is added to or subtracted from the loop counter within the loop.

(5) The loop counter is used within the loop as described in (A) or (B).

(A) Conditions *a* to *e* are satisfied (example 1):

a. The loop counter is used to index an array.

b. The elements of the array in *a* are structures, unions, or arrays.

c. The size of the elements of the array in *a* is not a power of 2.

d. The size of the elements of the array in *a* is within 65,534 bytes.

e. The product of the constant used for the loop termination condition in (3) and the size of an element of the array in *a* is too large to store in a 32-bit integer.

(B) Conditions *f* to *i* are satisfied (example 2):

f. The loop counter is used as one multiplier in an expression with a constant.

g. The constant in *f* is not a power of 2.

h. The constant in *f* is in the range from –65,534 to 65,534.

i. The product of the constant used for the loop termination condition in (3) and the constant in *f* is too large to store in a 32-bit integer.

* Loop counter: This controls when a loop ends.

Example 1:

```
struct {
    int s1;
    int s2;
    int s3;           // The elements of the array "ary" are structures.
} ary[100];           // The size (12 bytes) is not a power of 2 and
                      // is within 65,534 bytes.

int i;                     // The loop counter i is not declared as volatile.
for ( i = 0; i < INT_MAX; i ++ ){  // The loop termination condition involves
                                   // comparing the loop counter to a constant.
                                   // A constant is added to the loop counter i.

    ary[i].s1 = 1;         // The loop counter i is used to index the array.
    …
}
```

`INT_MAX*12` (which equals 0x5FFFFFFF4) is too large to store in a 32-bit integer, so this code might apply to this restriction.

Example 2:

```
volatile int vi;

int i = 0;               // The loop counter i is not declared as volatile.
while ( i < INT_MAX ){  // The loop termination condition involves
                         // comparing the loop counter to a constant.
  vi = i * 100;          // The loop counter i is used as one multiplier
                        // in an expression with a constant.
  …                     // The constant is in the range from -65,534 to 65,534.

  i++ ;                 // A constant is added to the loop counter i.
}
```

`INT_MAX*100` (which equals 0x31FFFFFF9C) is too large to store in a 32-bit integer, so this code might apply to this restriction.

[Workaround]

Do one of the following:

(1) Change the constant in the loop termination condition as follows:

- The product of this constant and the size of an element in an array indexed using the loop counter is not too large to store in a 32-bit integer.
- The product of this constant and a constant multiplied with the loop counter is not too large to store in a 32-bit integer.

(2) Declare the loop counter as `volatile`.

Modify with `volatile` the automatic variable whose address is acquired.

Before modification:

```
int i;
for ( i = 0; i < INT_MAX; i ++ ){

     ary[i].s1 = 1;
     …
}
```

After modification:

```
volatile int i;
for ( i = 0; i < INT_MAX; i ++ ){

    ary[i].s1 = 1;
    …
}
```

(3) Specify `-Od` or `-Ob` as the optimization option.

[Correction]

This issue has been corrected in CA850 Ver. 3.41.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

No. 105 Incorrect string literals

[Description]

If the following conditions are satisfied, the contents of the string literal become incorrect:

Conditions:

(1) ASCII code 0x00 is used in a string literal.

(2) An ASCII code from 0x30 through 0x37 immediately follows 0x00 in (1).

Example: When the ASCII code following the ASCII code 0x00 is 0x37

```
char string1[] = "\x00\x37";
char string2[] = "\000\067";     // (37)16 = (67)8
char string3[] = "\x00" "7";     // (37)16 = ’7 ’
```

The correct output is 0x00, 0x37, 0x00, respectively, but 0x07 and 0x00 are output.

[Workaround]

Do one of the following:

(1) Initialize strings without using string literals.

```
char string4[] = {'\x00', '\x37', '\0'};
```

(2) Specify the ASCII code for a character other than 0x30 to 0x37 after 0x00, and then dynamically replace that character.

```
char string5[] = "\x00*";
string5[1] = '\x37';
```

[Correction]

This issue has been corrected in CA850 Ver. 3.41.
A tool used to check whether this restriction applies is available.
Contact a Renesas Electronics sales representative or distributor for details.

No. 106 Restriction involving format specifiers for the `sscanf`, `fscanf`, and `scanf` functions

[Description]

If conditions (1) to (3) are satisfied, the parameter for the format specifier described in (3) is overwritten:

Conditions:

(1) The `sscanf`, `fscanf`, or `scanf` function is used.
(2) There are less entered fields than format specifiers.
(3) The first extra format specifier character is `s`, `e`, `f`, `g`, `E`, `F`, `G`, or `[ ]` (examples 1 and 2).

If conditions (4) and (5) are satisfied, the parameter for the format specifier described in (5) is overwritten.

Conditions:

(4) The `sscanf`, `fscanf`, or `scanf` function is used.
(5) `[ ]` are used as format specifier characters, and the character pattern enclosed by them is not in the entered fields (example 3).

Example : When first extra format specifier is `f`

```
char ary1[5];
float  f1 = 2.0, f2 = 3.0;

sscanf ("aaaa", "%s %f %f", ary1, &f1, &f2);
                    ↑ First extra format specifier
```

The input field `aaaa` is stored in `ary1` as a character string. However, there is no input field for the format specifiers `%f` and `%f` after `%s`. In this case, the value of the parameter `f1` for the first extra format specifier is overwritten.

```
Expected values: ary1 = "aaaa", f1 = 2.0, f2 = 3.0
Output result:   ary1 = "aaaa", f1 = 0.0, f2 = 3.0
```

Example 2: When first extra format specifier is `s`

```
int  data1, data2;
char ary2[5]="test";

sscanf ("1 2", "%d %d %s", &data1, &data2, ary2);
                      ↑ First extra format specifier
```

The input field 1 is stored in data1 as a decimal integer. The input field 2 is stored in data2 as a decimal integer. However, there is no input field for the format specifiers %d %d following after %s. In this case, the value of the parameter ary2 for the first extra format specifier is overwritten.

Expected values: `data1 = 1, data2 = 2, ary2 = "test"`
Output result: `data1 = 1, data2 = 2, ary2 = "\0"`

Example 3: When the character pattern enclosed by [ ] is not in the entered fields

```
char ary3[5] = "test";
char ary4[5] = "test";
char ary5[5] = "test";

sscanf ("aaaa bbbb cccc", "%s %[a] %s", ary3, ary4, ary5);
                                ↑ Format specifier that does not match the entered field
```

The input field aaaa is stored in ary2 as a string literal. Next, the code searches for a in the input field bbbb and attempts to store the result in ary3, but the character is not found. In this case, the value of ary3 is overwritten.

Expected values: `ary3 = "aaaa", ary4 = "test", ary5 = "test"`
Output result: `ary3 = "aaaa", ary4 = "\0", ary5 = "test"`

[Workaround]
There is no workaround.
[Correction]
This issue has been corrected in CA850 Ver. 3.41.

No. 107 Restriction involving the character string parameter for the atoi, atol, strtol, and strtoul functions

[Description]
If conditions (1) to (5) below are satisfied, the return value might be invalid. For the strtol and strtoul functions, the global variable errno is not set to ERANGE:
Conditions:

(1) The atoi, atol, strtol, or strtoul function is used.
(2) For the atoi or atol function, the character string parameter exceeds 32 bits when expressed as a decimal number.
For the strtol or strtoul function, the first character string parameter exceeds 32 bits when expressed using the base specified as the third parameter.
(3) When the portion of the character string parameter in (2) from the first character to a given character is converted to a number and compared to the absolute value of the lower 32 bits of the value that results when the portion of the same character string from the first character to the character following the given character above is converted to a number, the latter value is greater than the former.
(4) The comparison in (3) includes all characters in the string.

(5) For the `atoi`, `atol`, or `strtol` function, the lower 32 bits of the number to which the character string was converted in (2) are within the range from `LONG_MIN` to `LONG_MAX` after adding a sign.

Example 1: When the `strtoul` function is used

```
char *p;
unsigned long ul;

ul = strtoul("123456789", &p, 16);
```

The hexadecimal value 0x123456789 exceeds 32 bits, and the comparison in (3) includes all characters in the string.

Example: If the character string 12345678 is converted to hexadecimal (0x12345678) and the value is compared to the lower 32 bits of the value that results when the string including the next number (9) is converted to hexadecimal (0x123456789), the latter is greater than the former.
In other words, 0x12345678 < 0x23456789.

Expected values: `ul = ULONG_MAX   errno = ERANGE`
Output result: `ul = 0x23456789`

Example 2: When the `strtol` function is used

```
char *p;
signed long l;

l = strtol("-123456789", &p, 16);
```

The hexadecimal value 0x-123456789 exceeds 32 bits, and the comparison in (3) includes all characters in the string.

Expected values: `l = LONG_MIN   errno = ERANGE`
Output result: `l = DCBA9877(-0x23456789)`

Example 3: When the `atoi` function is used

```
signed int i;

i = atoi("5368709120");
```

The decimal value 5368709120 (which equals 0x140000000) exceeds 32 bits, and the comparison in (3) includes all characters in the string.

Example: If the character string "536870912" is converted to decimal (536870912, which equals 0x20000000) and the value is compared to the lower 32 bits of the value that results when the string including the next number ("5368709120") is converted to decimal (5368709129, which equals 0x140000000), the latter is greater than the former.
In other words, 0x20000000 < 0x40000000.

Expected values: `i = LONG_MAX`

Output result: `i = 1073741824(0x40000000)`

[Workaround]

There is no workaround.

[Correction]

This issue has been corrected in CA850 Ver. 3.41.

No. 108 Restriction involving initialization of a structure that includes a bit field among its members

[Description]

If conditions (1) and (2) are satisfied, the bit field is not correctly initialized (example 1):

Conditions:

(1) A structure is used that includes a bit field immediately followed by another structure (or union) as members.

(2) Both of the members included in the structure in 1 are initialized using initial values.

If conditions (3) to (5) are satisfied, the bit field might not be correctly initialized (example 2).

(3) An automatic array of structures is used.

(4) The structure in 3 includes a bit field and an element that is at least 126 bytes among its members.

(5) The initializer for this element is omitted, and the element is implicitly initialized to 0.

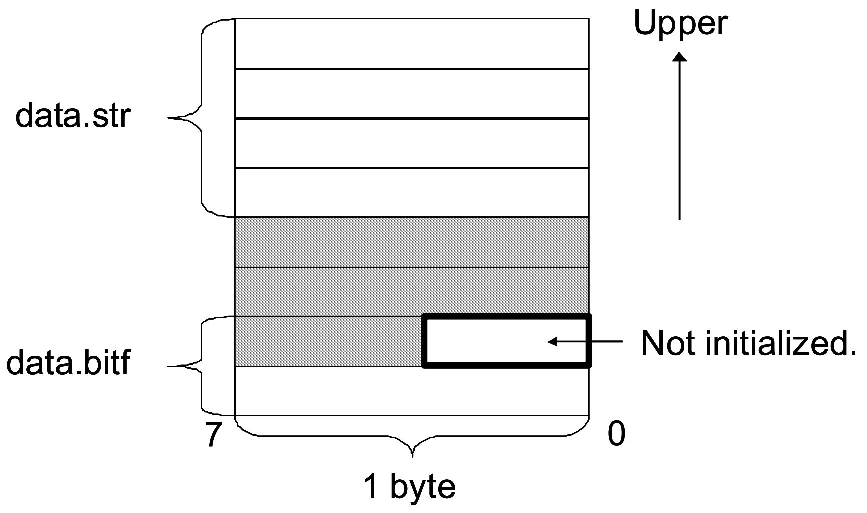
Example 1:

```
struct {
        int bitf : 12 ;      // A bit field is used.
        struct {            // A structure follows the bit field.
                int s ;
        } str ;
} data = { 0xFFF, { 2 } } ;  // Both members of the structure that includes the
                            // bit field are initialized.
```

The higher 4 bits of `data.bitf` are not initialized.

Expected values: `data.bitf  = 0xFFF`

Output result: `data.bitf  = 0xFF`



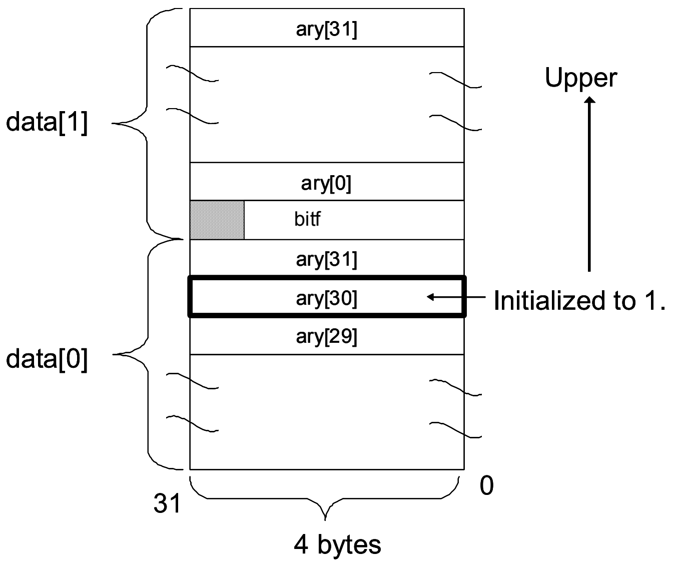
Example 2:

```
void func(void)
{
  struct {
      int bitf : 25;           // A bit field is used.
      int ary[32];             // The following element is at least 126 bytes.
  } data[2] = { { 1 }, { 1 } }; // The automatic array of structures data is used.
  …                            // ary is implicitly initialized to 0.
}
```

The offset of the code used to set `data[1].bitf` to 1 ends up too low. `data[0].ary[30]` is initialized to 1. `data[1].bitf` ends up undefined.

Expected values: `data[0].ary[30] = 0`
`data[1].bitf = 1`

Output result: `data[0].ary[30] = 1`
`data[1].bitf` is undefined.



[Workaround]

Do one of the following:

(1) Initialize the structure by assigning values to its members.

For example 1:

```
struct {
   int bitf : 12 ;
   struct {
        int s ;
   } str ;
} data ;                  // Not initialized
…
data.bitf = 0xFFF ;       // Changed to assignment
data.str.s = 2 ;
```

(2) Declare the bit field in a separate structure within the main structure.

For example 1:

```
struct {
   struct {
          int bitf : 12 ;
   } str2 ;              // Structure within the structure
   struct {
          int s ;
   } str ;
} data = { { 0xFFF }, { 2 } } ;
```

[Correction]

This issue has been corrected in CA850 Ver. 3.41.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

No. 109 Restriction involving nested conditionally assembled pseudo instructions

[Description]

If the following conditions are satisfied, the error message F3510 is output:

Conditions:

(1) An `.elseif` or `.elseifn` pseudo instruction is used.

(2) The condition in (1) evaluates to true.

(3) There is an `.elseif` or `.else` pseudo instruction that corresponds to the pseudo instruction in (1).

(4) There is a pseudo instruction that is conditionally assembled nested within the block that corresponds to the pseudo instruction in (1).

Example:

```
.set    FLAG1, 0
.set    FLAG2, 1
.set    FLAG3, 1
.set    FLAG4, 0

.if FLAG1 == 1
  .set   TEMP, 1
.elseif FLAG2 == 1  -- Satisfies conditions (1) and (2)
   .if FLAG3 == 1    -- Satisfies condition (4)   ┐
     .set   TEMP, 2                               ├ Nested
   .endif                                         │
.elseif FLAG4 == 1  -- Satisfies condition (3)    ┘
  .set   TEMP, 3
.endif
```

[Workaround]

When nesting a conditionally assembled pseudo instruction, use either block 1 or 2, below.

(1) `.if` pseudo instruction block

```
.if FLAG1 == 1
  .set   TEMP, 1
.else
  .if FLAG2 == 1
    .if FLAG3 == 1     ┐
      .set   TEMP, 2   ├ Nested
    .endif             │
  .elseif FLAG4 == 1   ┘
    .set   TEMP, 3
  .endif
.endif
```

(2) `.elseif` pseudo instruction block that ends with `.endif`

```
.if FLAG1 == 1
  .set   TEMP, 1
.elseif FLAG4 == 1
  .set   TEMP, 3
.elseif FLAG2 == 1
   .if FLAG3 == 1
     .set   TEMP, 2
 .endif
.endif
```

Nested

[Correction]

This issue has been corrected in CA850 Ver. 3.41.

No. 110 Restriction involving assignment within `switch` and `if` statements

[Description]

If the following conditions are satisfied, the processing within `switch` or `if` statements might become incorrect as a result of optimization by the ca850:

Conditions:

(1) The C source code satisfies conditions (A) to (C) (example 1).

(A) For a version earlier than 2.50, the compiler optimization option `-Os` or `-Ot` is specified in addition to `-Ol`, but `-Wi,-O4` is not specified.
For Ver. 2.50 or later, the compiler optimization option `-Os` or `-Ot` is specified, but `-Wi,-O4` is not specified.

(B) Parallel processing is performed to assign a value to the same variable in a `switch` or `if` statement.

(C) After the assignment in (B), execution jumps back to the same position.

Example 1:

```
int func(int val) {
  int res;
  switch (val) {
  case 0xA: res = 0x1; break;
  case 0xB: res = 0x2; break;
  case 0xC: res = 0x3; break;
  case 0xD: res = 0x3; break;
  case 0xE: res = 0x3; break;
  default:  res = 0x3; break;
  }
  return res;
}
```

Parallel processing to assign a value to the same variable (`res`).

After a value is assigned to `res`, execution jumps back to the same position.

(2) The assembly language output according to the C source code in (1) satisfies all of the following conditions (example 2):

(a) The `mov` or `ld` directive is used to transfer data to the same register at the end of multiple basic blocks (*Block A*).

(b) *Block A* combines everything into one block (*Block B*).

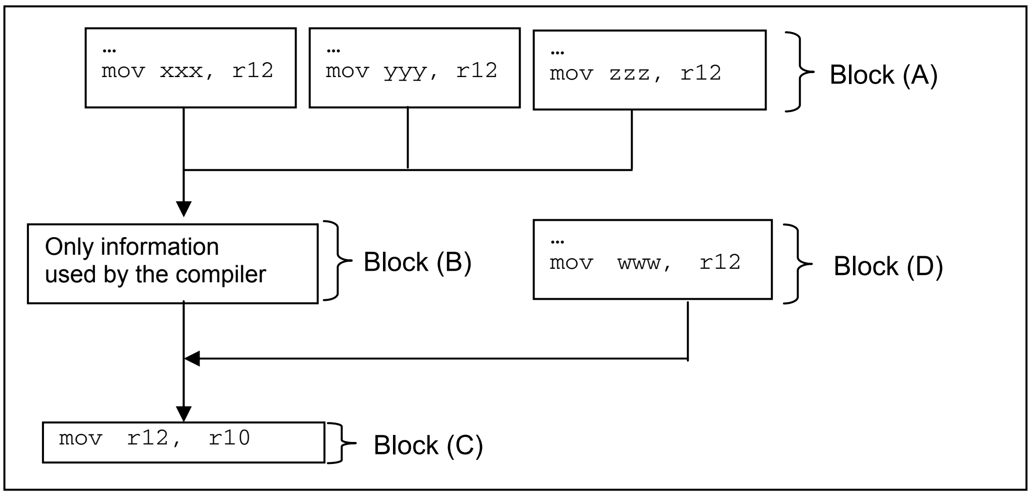
(c) There are no directives (or output code) in *Block B*, which contains information used by the compiler.
At least one information item is included.

(d) In *Block C* (the block following *Block B*), the data is transferred to a register other than the one used in *Block A*.

(e) Among the blocks directly combined into *Block C*, there is a block (*Block D*) that performs the same transfer as that performed in *Block A*.

Remark A basic block is group of directives that are processed in order and ends with a directive that causes execution to jump.

Example 2:



[Workaround]

Do one of the following:

(1) Insert `__asm("\n");`. (The check tool outputs the position where this insertion is required.)

```
int func(int val) {
  int res;
  switch (val) {
  case 0xA: res = 0x1; break;
  case 0xB: res = 0x2; break;
  case 0xC: res = 0x3; break;
  case 0xD: res = 0x3; break;
  case 0xE: res = 0x3; break;
  default:  res = 0x3; break;
  }
  __asm("\n");  ←——————— Insert __asm("\n");.
  return res;
}
```

(2) Specify `-O` or lower as the optimization option.

(3) Specify the `-Wi,-O4` option.

[Correction]

This issue has been corrected in CA850 Ver. 3.41.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

No. 111 Restriction on incorrect compare operation optimization while casting

[Description]

No.111 occurs when the following conditions are satisfied:

Conditions:

(1) Compare operation (<, <=, >, >=, ==, !=) between integer operands.

(2) Either operand is casted as shown below:

• signed char	→ unsigned char
• signed char	→ unsigned short
• unsigned char	→ signed char
• signed short	→ unsigned short
• unsigned short	→ signed short
• unsigned short	→ signed char

(3) After casting the operand, its type is the same as the type of the other operand.

(4) Other operand is treated as constant by way of compiler optimization.

(5) Comparison used in (1) occurs between constant in (4) and before casting of (2). This will always be true or always be false).

Example:

```
short s; unsigned short us;
s = -1;
…
if (s == (short)us)
```

In this example, even if the result of comparison operation depends on its operands, CA850 incorrectly interprets the comparison result as always false. This results in eliminating both the if-sentence and if-true-block.

[Workaround]

Insert '__asm("\n");' before the line where incorrect comparison operation optimization occurs.

Checktool will identify the line number. When a function that has incorrect comparison operation optimization is inlined, checktool will identify the function name and its line number.

```
short s; unsigned short us;
s = -1;
…
__asm("\n");
if (s == (short)us)
```

[Correction]

This issue has been corrected in Ver. 3.47.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

No. 112 Restriction on incorrect move optimization of assignment sentence

[Description]

No.112 occurs when the following conditions are satisfied, resulting in moving the statement assigning variable X in Basic Block 1 to the back of the sentence assigning variable X in Basic Block 2 incorrectly:

Conditions:

(1) Optimization level is –Og or higher (-Og, -O, -Os, -Ot, and associated code optimizations)

(2) Control flow of function consists of Basic Blocks like (a) to (e).

(a) Either non-volatile automatic variable X or non-volatile argument variable X is assigned in both Basic Block 1 and 2, then referenced in Basic Block 3.

(b) The value assigned to variable X in Basic Block 2 is either volatile variable expression or peripheral I/O register expression.

(c) The path from Basic Block 1 is the only one merging into Basic Block 2.

(d) After Basic Block 2, both path-merging-into Basic Block 3 and path-not exist.

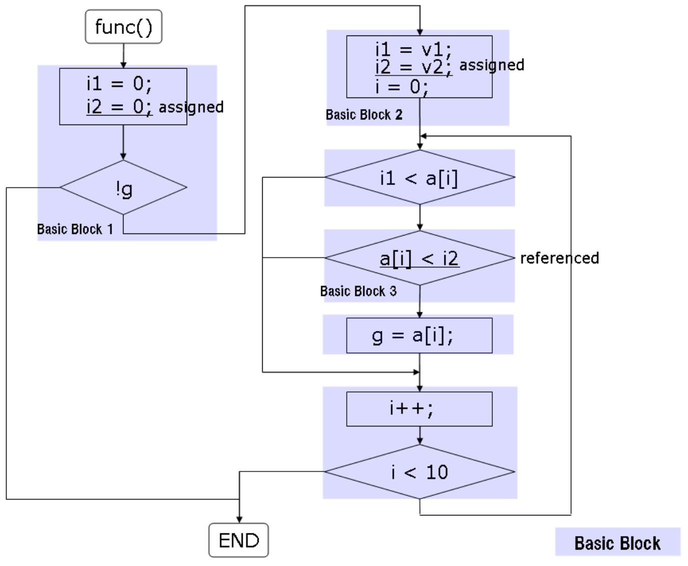
(e) No indirect access to variable X between Basic Block 1 and 3.

Note: A basic block is code that has one entry point, one exit point, and no jump instructions contained within it.

Example:

```
int g, a[10];
volatile int v1, v2;
void func(void) {
  int i1 = 0, i2 = 0;
  if(!g){
    int i;
    i1 = v1;
    i2 = v2;
    for(i = 0; i < 10; i++)
      if(i1 < a[i] && a[i] < i2)
        g = a[i];
  }
}
```

Control flow of the example code is as follows.



[Workaround]

Insert '__asm("\n");' before the line which the checktool output.

```
if(!g){
  int i;
  i1 = v1;
  i2 = v2;
  __asm("\n");
  for(i = 0; i < 10; i++)
    if(i1 < a[i] && a[i] < i2)
      g = a[i];
}
```

[Correction]

This issue has been corrected in Ver. 3.47.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

CX Usage Restrictions

1. Product History

No.	Usage Restrictions	CubeSuite Package		
		V1.20		V1.40
		CX		
		V1.00	V1.01	V1.10
1	Specifying the "-Xno_romize" and "-Xtwo_pass_link" options simultaneously causes an error	×	×	×
2	Input file names containing non-ASCII characters	×	×	×
3	Specifying a file on a different drive using a relative path causes an error	×	×	○
4	"-Xr" option	×	×	○
5	Interrupt functions with static modifier and smart correction functions	×	○	○
6	Restriction on arrays declared in sections that have the `data` or `bss` attribute	×	○	○
7	Error caused by a variable allocated in a `data` or `bss` section	×	×	○
8	A variable referenced after being changed in a function call becomes an invalid value.	○	○	×
9	A variable referenced after indirect assignment becomes an invalid value.	○	○	×

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 Specifying the "-Xno_romize" and "-Xtwo_pass_link" options simultaneously causes an error

[Description]

When the "-Xno_romize" and "-Xtwo_pass_link" options are specified simultaneously, then an F0562003 error will occur if a file with the same name as the output load module file already exists.

```
F0562003:"file" is not ELF file.
```

[Workaround]

Delete any file with the same name as the output load module file before running.

[Fix]

Correction is not planned. Regard this item as a specification.

No. 2 Input file names containing non-ASCII characters

[Description]

If the file name (including the path) contains no-ASCII characters, one of 1 or 2 below will occur if "-Xpass_source" is specified.

1. C source lines output as comments in the assembler source file will be invalid
2. An E0592018 error will occur

```
E0592018:Failed to open an list file "file".
```

[Workaround]

Change the file name (including the path) to one that does not contain non-ASCII characters.

[Fix]

Correction is not planned. Regard this item as a specification.

No. 3 Specifying a file on a different drive using a relative path causes an error

[Description]

Specifying a file located on a different drive as an input or output file, or as an option parameter, using a relative path, will cause an E0511102 error.

```
E0511102: The file "file" specified by the "character string" option is not found.
```

[Example]

Specifying a parameter to the "-Xstartup" option

```
>cx.exe -Cf3507 -Xstartup=c:..\cstart.obj main.c
```

The following E0511102 error will occur.

```
E0511102: The file " c:..\cstart.obj " specified by the "-Xstartup " option is not found.
```

[Workaround]

Specify files on different drives using the absolute path.

[Fix]

This issue has been corrected in CX V1.10.

No. 4 "-Xr" option

[Description]

Using multiple "-Xr" options to specify the same external variable for different registers will not cause an error. It will be allocated to the register which is specified first by -Xr option.

[Example]

The external variable "A" is assigned to registers r15, r16, and r17

```
>cx.exe -Cf3507 -Xreg_mode=22 -Xr15=A -Xr16=A -Xr17=A main.c
```

In this case, external variable A will be allocated to register r15.

[Workaround]

When using the "-Xr" option, do not specify the same external variable for different registers.

[Fix]

This issue has been corrected in CX V1.10.

No. 5 Interrupt functions with static modifier and smart correction functions

[Description]

If an interrupt function as a static modifier, then one of the 1 or 2 below will occur.

1. If the option "-Odelete_static_func=off" is specified, CX will crash.
2. If the option "-Odelete_static_func=off" is not specified, the interrupt function with static modifier will be deleted

If a function corrected via the "smart correction" feature is given a static modifier, it will cause an F0550508 error.

```
F0550508: identifier undefined.
```

[Workaround]

Do not add a "static" modifier to interrupt functions or functions corrected via the smart correction feature.

[Fix]

This issue has been corrected in CX V1.01.

No. 6 Restriction on arrays declared in sections that have the `data` or `bss` attribute

[Description]

If the following four conditions are satisfied, the access or address acquisition in *3* might be performed incorrectly:

1. The optimization option `-O`, `-Osize`, or `-Ospeed` is specified.
2. An array is declared in a section that has the `data` or `bss` attribute.
3. A variable is used as an index to access an element of the array in *2* other than the first element or to acquire the address of such an element.
4. The index in *3* is handled as a constant due to optimization.

[Example]

```
#pragma section data  /* The array ary is declared in a .bss section. */
int ary[7];

void f(void)
{
    int i;
    i = 1;
    ary[i] = 0;        /* The variable i is handled as the constant 1 due
                        to optimization. */
}                      /* The array element ary[1], which is not the first
                        element, is accessed.*/
```

[Workaround]

Declare the variable used as the array index as `volatile`.

```
#pragma section data
int ary[7];

void f(void)
{
    volatile int i;
    i = 1;
    ary[i] = 0;
}
```

[Fix]

This issue has been corrected in CX V1.01.

No. 7 Error caused by a variable allocated in a data or bss section

[Description]

When a variable defined in a separate file that satisfies both of the following conditions is accessed from the C source, error F0560419 occurs:

1. The variable is allocated in a `data` or `bss` section.
2. The data is allocated outside the range of ±32 KB from the GP symbol.

[Workaround]

Insert the following inline-assembler instruction at the beginning of the first function using any variable that causes error F0560419:

```
__asm("$data _variable-name");
```

Example: When the `val` variable causes the error

```
__asm("$data _val");
```

Note the following when inserting the above instruction:

- Insert the instruction at the beginning of only the first function that references the variable in a given file.
- Do not insert the instruction into the file in which the variable is defined.

[Fix]

This issue has been corrected in CX V1.10.

No. 8 A variable referenced after being changed in a function call becomes an invalid value.

[Description]

When all the following requirements are met, the value of variable "`V`" that is changed within the called function "`F2`" will become invalid.

<Conditions>

1. One of the optimization options is chosen: `-O`, `-Osize`, or `-Ospeed`.
2. The variable "`V`" which is global or static (declared in file-scope) is not declared as volatile and the address of "`V`" is not taken in function "`F1`".
3. The function "`F2`" that rewrites the value of variable "`V`" is called in function "`F1`" and the variable "`V`" is read before and after the call to function "`F2`".
4. Variable "`V`" is read in basic block "`B2`" after function "`F2`" is called. There must be only one execution path back to basic block "`B1`".
5. In the execution path after basic block "`B2`", there are no function calls or there are function calls before which "`B2`" is always executed.
6. None of the built-in functions are contained in function "`F1`": `asm/nop/halt/set_il/ldsr/stsr/ldgr/stgr`.

Note: A basic block is a portion of the code within a program with certain desirable properties that make it highly amenable to analysis.

[Example]

```
int F2(void);      // Rewrite variable V in this function block
int V;

 void F1(void)
 {
   if (V){        // Read variable V
     if (F2()){   // F2 rewrites variable V
     }
     else {
       V++;      // Read variable V
   }
  }
```

[Workaround]

Do either of the following:

1. Specify an optimization option other than `-O`, `-Osize`, and `-Ospeed`.
2. Insert `__asm("\n");` in any line in function `F1`.
3. Declare variable `V` as `volatile`.

[Fix]

This issue will be corrected in CX V1.11.

No. 9 A variable referenced after indirect assignment becomes an invalid value.

[Description]

When all the following requirements are met, the value of variable "`V`" after indirect assignment to "`A`" becomes invalid.

<Conditions>

1. One of the optimization options is chosen: `-O`, `-Osize`, or –Ospeed.
2. The pointer is not declared as volatile when specifying initial address of variable "`V`" or the address of variable "`V`" is taken in another source file.
3. The value of variable "`V`" is accessed by pointer `*P` in function "`F1`". It is rewritten between two different accesses of variable "`V`".
4. There is one execution path to basic block "`B2`" from "`B1`" where variable "`V`" is read in each block, and no other execution path input to "`B2`".
5. None of the built-in functions are contained in function "`F1`": `asm/nop/halt/set_il/ldsr/stsr/ldgr/stgr`.

Note: A basic block is a portion of the code within a program with certain desirable properties that make it highly amenable to analysis.

[Example]

```
int  V;
int  j;
int* P = &V; // Initialize the pointer to the address of variable V.

void F1(void)
{
  if (V) {   // Read variable V
    *P = 0;  // Rewrite variable V using indirect assignment.
    if (V)   // Read variable V
      j = 0;
```

[Workaround]

Do either of the following:

1. Specify an optimization option other than `-O`, `-Osize`, and `-Ospeed`.
2. Insert `__asm("\n");` in any line in function `F1`.
3. Declare variable `V` as `volatile`.

[Fix]

This issue will be corrected in CX V1.11.

CX Utility Usage Restrictions

1. Product History

No.	Usage Restrictions	CubeSuite Package		
		V1.20		V1.40
		CX		
		V1.00	V1.01	V1.10
1	CX outputs an information file for CubeSuite when a structure or union is defined.	×	○	○

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 CX outputs an information file for CubeSuite when a structure or union is defined.

[Description]

If the following conditions 1 to 3 are satisfied, CX outputs an invalid information file for CubeSuite. However, compiler processing is performed correctly.

Conditions

1. The `-Xcube_suite_info` option is specified.
2. A structure or union (including a bit field) is defined.
3. `typedef` is used to declare the data type of a variable in condition 2, or a type qualifier (`const`, `volatile`, or both) is specified for a variable in condition 2.

Due to the information file for CubeSuite being invalid, the following CubeSuite features cannot be used:

- Function jump feature
- Program analysis feature

[Workaround]

There is no workaround.

[Fix]

This issue has been corrected in CubeSuite Ver. 1.30.

Stack Usage Tracer Usage Restrictions

1. Product History

No.	Usage Restrictions	CubeSuite Package						
		V1.00	V1.10		V1.12		V1.20	
		Stack Usage Tracer						
		V4.00					V4.01	
1	Restriction on output folder for assembly file	×					○	

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 Restriction on output folder for assembly file

[Description]

If conditions 1 and 2, below, are both met, then the Stack Usage Tracer cannot operate correctly on files for which the individual compile option is not set.

[Conditions]

1. "Project for CA850" or "Library project for CA850" is selected as the kind of project
2. In the compile option, the "Output folder for assembly file" was changed from the default of *%BuildModeName%*

[Workaround]

Do either of the following:

1. Re-specify the default *%BuildModeName%* for the “Output folder for assembly file” option on the “Compile Options” tab.
2. Specify the “Output folder for assembly file” option on the “Individual Compile Options” tab.

[Action]

This issue has been corrected in stack usage tracer Ver. 4.01.

Usage Restrictions on Simulator for 78K0R/Kx3

1. Product History

No.	Usage Restrictions	CubeSuite Package					
		V1.00	V1.10		V1.12		V1.20
		Simulator for 78K0R/Kx3					
		V2.42	V2.43				
1	Restrictions for disconnection of debug tool	×	○				

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 Restrictions for disconnection of debug tool

[Description]

When a limited user (a user who is not an Administrator) is logged into Windows and uses the 78K0R/Kx3 Simulator, the following error message may appear when disconnecting from the debug tool.

"Failed to exit debugger" (Error Number: E0210001)

Once this error occurs, no matter how many times the user attempts to disconnect the debug tool, the same error message will appear, and it will not be possible to exit CubeSuite.

[Workaround]

There is no workaround.

If this issue has occurred, forcibly terminate CubeSuite.

[Action]

This issue has been corrected in 78K0R/Kx3 Simulator Ver. 2.43 (CubeSuite Ver. 1.10).

CA78K0R Usage Restrictions

1. Product History

List of restrictions for compiler part

No.	Usage Restrictions	CubeSuite Package		
		V1.00, V1.10	V1.12, V1.20	V1.40
		CA78K0R		
		V1.00	V1.01	V1.02
27	An error occurs when the *-ng* option is specified and a branch instruction is used in a function that includes the *asm* statement.	×	○	○
28	The line number is not output for a statement that immediately follows a nested *if* statement and is outside that statement's conditional block.	×	○	○
29	Restriction due to an instruction (`mov`, `movw`) that uses `word[BC]` as an operand accessing the incorrect address	×	×*	○
30	The C0101 error occurs when the `-qj` option is specified	×	×	×
31	Restriction on type-casting addresses in the `far` area to a `long` or `unsigned long`	×*	×*	×*
32	Macro expansion using the `##` operator results in an error	×	×	×
33	Symbol information for an interrupt function is not output to the assembler source	×	×	×
34	Code that specifies the ES register settings might not be output	×*	×*	×*

×: Applicable, ○: Not applicable, –: Not relevant, *: Check tool available

Remark The item numbers are the same as those for the CC78K0R and are not sequential.

2. Restriction Details

No. 27 An error occurs when the *-ng* option is specified and a branch instruction is used in a function that includes the *asm* statement.

[Description]

If the *-ng* option is specified and a branch instruction is used after the *asm* statement in a function, an error might occur. This error might occur if both of the following conditions are satisfied:

(1) The *asm* statement is used in a function.

(2) A statement that causes processing to branch (such as *if*, *for*, or *while*) is used in the same function.

However, if the above conditions are satisfied, the error occurs during assembly and the object module file is not generated. If no error occurs, this restriction does not apply.

Example:

```
[.c]
unsigned int i;
void func()
{
    do {
    __asm("\t DB    (1000)");
    i++;
    } while ( i < 10) ;
}
```

[Workaround]

Change to the *-g* option.

[Action]

This issue has been corrected in CA78K0R Ver. 1.01.

No. 28 The line number is not output for a statement that immediately follows a nested *if* statement and is outside that statement's conditional block.

[Description]

If all of the conditions below are satisfied, the line number might not be output for a statement that immediately follows a nested *if* statement and is outside that statement's conditional block. However, the output code is correct. A break point cannot be specified for a statement for which the line number is not output.

(1) There are at least three levels of nested *if* statements.

(2) An else statement in a higher nested level has a larger line number than a nested *if* statement.

(3) At least one statement immediately follows an *if* statement in a higher nested level.

Example:

```
[.c]
int   f0, f1, f2, f3;
int   g0, g1, g2;
void func(void)
{
        if (f0){
            if (f1){            /* if statement in nested level 1 */
                g2 = 5;
            }
            else if (f2){       /* if statement in nested level 2 */
                 g2 = 4;
            }
            else if (f3){       * if statement in nested level 3 (condition (1)) */
                g2 = 3;
            }
            else {
                g2 = 2;
            }
            g0 = 0x1234;        /* A statement immediately follows the if statement */
                                /* in nested level 1 (condition (3)).*/
            g1 = 0x5678;
        }
        else {                  /* This else statement has a larger line number */
            g2 = 1;             /* than the if statement in nested level 3 (condition (2)).*/
        }
  }
```

[Workaround]

Insert several blank lines before the statement that immediately follows the nested if statement and is outside that statement's conditional block.

For the above example, insert the lines before *g = 0x1234;*.

[Action]

This issue has been corrected in CA78K0R Ver. 1.01.

No. 29 Restriction due to an instruction (`mov`, `movw`) that uses `word[BC]` as an operand accessing the incorrect address

[Description]

In either of the cases below, the output code is invalid.

For the `word[BC]` operand, if the `word` + `BC` address exceeds 10000H, an unintended address is accessed.

(1) Indirect reference is performed by subtracting a constant from a pointer.

However, this does not apply in the following cases:

(a) When the large model is used and the pointer is not pointing to a `near` area

(b) When the pointer is pointing to a `far` area

(c) When the pointer is pointing to one byte of data and the constant 1 is subtracted

(d) When the pointer is pointing to one byte of data, the constant 2 is subtracted, and optimization is not being performed with speed specified as having priority[Note].

(e) When the pointer is pointing to two bytes of data, the constant 1 is subtracted, and optimization is not being performed with speed specified as having priority[Note].

Note When `-qx1` is specified, or `l` is not specified for the `-q` option

(2) When referencing an array element, the variable used as the index contains a negative value, or indirect reference is performed by adding an offset (including a variable that contains a negative value) to the address indicated by an aggregate[Note].

Note An aggregate is a structure or array.

Condition (1) example

```
-------*.c-------------
unsigned char x[5], *ucp1;
unsigned short y[5], *usp1;
signed short si1;
void func1()
{
/*********When the pointer accesses one byte of data**********/
x[0] = *(ucp1 - 1);   /* No problem */
x[1] = *(ucp1 - 2);   /* A problem occurs only if speed is prioritized for optimization. */
x[2] = *(ucp1 - 3);   /* A problem occurs. */
x[3] = ucp1[-4];     /* A problem occurs. The code has the same meaning as *(ucp1 -4). */
x[4] = *(ucp1 + si1 - 5); /* A problem occurs.*/
/*********When the pointer accesses two bytes of data**********/
y[0] = *(usp1 - 1);    /* A problem occurs only if speed is prioritized for optimization. */
y[1] = *(usp1 - 2);    /* A problem occurs. */
y[2] = *(usp1 - 3);    /* A problem occurs. */
y[3] = usp1[-4];       /* A problem occurs. The code has the same meaning as *(usp1 -4). */
y[4] = *(usp1 + si1 - 5); /* A problem occurs. */
}
```

```
--------*.asm---------  (Excerpt from the assembly code for the above)
; line    25 :   x[2] = *(ucp1 - 3);  /* A problem occurs. */
$DGL   0,9
      movw  bc,!_ucp1         ;[INF] 3, 1
      mov   a,65533[bc]       ;[INF] 3, 1 ← An invalid address might be accessed.
      mov   !_x+2,a           ;[INF] 3, 1
---------------------
```

Condition (2) example

```
unsigned char x[4], *ucp1, uca1[10];
signed short sidx;
signed char cidx;
void func2()
{
 x[0] = uca1[cidx];          /* A problem only occurs if cidx is a negative non-zero value. */
 x[1] = *(&uca1[3] + cidx);  /* A problem only occurs if cidx is a negative non-zero value. */
 x[2] = uca1[sidx];          /* A problem only occurs if sidx is a negative non-zero value. */
 x[3] = *(&uca1[4] + sidx);  /* A problem only occurs if sidx is a negative non-zero value. */
}
```

```
--------*.asm---------  (Excerpt from the assembly code for the above)
; line    36 :     x[0] = uca1[cidx];/* A problem only occurs if cidx is a negative non-zero
                                     value. */
$DGL  0,20
      mov    a,!_cidx        ;[INF] 3, 1
      sarw   ax,8            ;[INF] 2, 1
      movw   bc,ax           ;[INF] 1, 1
      mov    a,_uca1[bc]     ;[INF] 3, 1 ← An invalid address might be accessed.
      mov    !_x,a           ;[INF] 3, 1
---------------------
```

[Workaround]

- Workaround for condition (1)

  If indirect reference is performed when subtracting a constant from a pointer, assign the result of the subtraction to a different pointer before accessing an address.

Example 1:

Before applying the workaround	After applying the workaround

`unsigned char x, *ucp1;`
`void func1()`
`{`
    `x = *(ucp1 - 3);`
 	
`}`
	`unsigned char x, *ucp1, *ucp2;`
	`void func1()`
	`{`
	    `ucp2 = ucp1 - 3;`
	    `x = *ucp2;`
	`}`

Example 2:

Before applying the workaround	After applying the workaround

`unsigned char x, *ucp1;`
`void func2()`
`{`
    `x = ucp1[-4];`
 	
`}`
	`unsigned char x, *ucp1, *ucp2;`
	`void func2()`
	`{`
	    `ucp2 = ucp1 - 4;`
	    `x = *ucp2;`
	`}`

- Workaround for condition (2)

  If a variable that has a negative value is used in a signed index expression to reference an array element, or if indirect reference is performed by adding an offset (a variable that has a negative value) to an address that indicates an aggregate, assign the result of calculating the expression to a pointer before accessing an address.

Example 1:

Before applying the workaround	After applying the workaround

`unsigned char x, uca1[10];`
`signed char cidx;`
`void func1()`
`{`
    `x = uca1[cidx];`
 	
`}`
	`unsigned char x, uca1[10], *ucp1;`
	`signed char cidx;`
	`void func1()`
	`{`
	    `ucp1 = uca1 + cidx;`
	    `x = *ucp1;`
	`}`

Example 2:

Before applying the workaround	After applying the workaround

`unsigned char x, uca1[10];`
`signed char cidx;`
`void func2()`
`{`
    `x = *(&uca1[3] + cidx);`
 	
`}`
	`unsigned char x, uca1[10], *ucp1;`
	`signed char cidx;`
	`void func2()`
	`{`
	    `ucp1 = &uca1[3] + cidx;`
	    `x = *ucp1;`
	`}`

[Action]

This issue has been corrected in CA78K0R Ver. 1.02.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

No. 30 The C0101 error occurs when the `-qj` option is specified

[Description]

The C0101 error occurs if both of the following conditions are satisfied:

(1) The jump optimization option (`-qj`) is enabled.

(2) Any of the following code is included in processing that is not executed, such as an `if(0)` or `if(1) else` statement:

- Initialization for arrays, structures, or unions
- `switch` statements accompanying table jumps

Example:

```
[*.c]
struct t1 {
unsigned char uc1;
};
char c;
void func1()
{
  struct t1 st0 = {0x07};
    if(0) {
            struct t1 st0 = {0x08};    /* Processing that is not executed */
            c++;
    }
}
```

[Workaround]

Do either of the following:

(1) Disable the`-qj` option.

(2) Delete the code that is not executed or embed the code between `#if 0` and `#endif` so as not to make it subject to compiling.

[Action]

This issue has been corrected in CA78K0R Ver. 1.10.

No. 31 Restriction on type-casting addresses in the `far` area to a `long` or `unsigned long`

[Description]

If an address in the `far` area is type-cast to a `long` or `unsigned long` and the result is used as the initial value of the address, the highest byte of the address is fixed to 0FH.

Example:

```
[*.c]
__far int const fi1 = 5;
__far unsigned long const ula1[] = { &fi1 };
__far unsigned long const ula2[] = { (unsigned long)&fi1 };


[*.asm]
@@CNSTL  CSEG  PAGE64KP
_fi1:    DW    05H    ; 5
_ula1:   DG    _fi1
_ula2:   DW    loww (_fi1)
         DW    0FH    ; 15
```

[Workaround]

Do not type-cast an address to a `long` or `unsigned long` if the initial value is used for the address.

[Action]

This issue will be corrected in CA78K0R Ver. 1.10.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

No. 32 Macro expansion using the `##` operator results in an error

[Description]

The E0803 error might occur if the `##` operator is not followed by a function-like macro parameter, capital or small letter, or underscore (_).

In addition, using the `##` operator might cause errors other than E0803, such as E0711 or E0301.

Example 1

```
[*.c]
#define m1(x) (x ## .c1 + 23)
#define m2(x) (x ## .c1 + 122)
struct t1 {
    unsigned char c1;
} st1;
unsigned char x1, x2;
void func1()
{
            x1 = m1(st1) + 100;    /* E0803 error (NG) */
            x2 = m2(st1) + 1;      /* No error   (OK) */
}
```

Example 2

```
[*.c]
#define m3(x) (x ## 1)
unsigned char x3, uc1;
void func2()
{
            x3 = m3(uc);          /* E0711, E0301 error (NG) */
}
```

[Workaround]

Do either of the following:

(1) Do not use the `##` operator.

```
#define m1(x)      (x ## .c1 + 23)
#define m2(x)      (x ## .c1 + 122)
     ↓
#define m1(x)      ((x).c1 + 23)
#define m2(x)      ((x).c1 + 122)
```

(2) Place a function-like macro parameter immediately after the `##` operator.

```
#define m3(x)      (x ## 1)
unsigned char x3, uc1;
void func2()
{
     x3 = m3(uc);
}
     ↓
#define m3(x, y)     (x ## y)
unsigned char x3, uc1;
void func2()
{
     x3 = m3(uc, 1);
}
```

[Action]

This issue will be corrected in CA78K0 Ver. 1.20.

No. 33 Symbol information for an interrupt function is not output to the assembler source

[Description]

The E3405 error occurs during linking if all the following conditions are satisfied:

(1) `#pragma interrupt` is used to specify the generation of a vector table for an interrupt function.

(2) The interrupt function is not defined in the same source.

(3) The `-no` option, assembler source module file output option (`-a` or `-sa`), and debugging information output option (`-g`) are enabled.

Example

```
[*.c]
#pragma interrupt INTP0 inter
/*    Definition of this interrupt function is not subject to compilation
__interrupt void inter()
{
            ...
}
*/
```

[Workaround]

Define the interrupt function or output the object module file.

[Action]

This issue will be corrected in CA78K0 Ver. 1.20.

No. 34 Code that specifies the ES register settings might not be output

[Description]

Code that specifies the ES register setting might not be output if a far variable is referenced directly after a floating point variable is incremented or decremented.

Example 1

```
[*.c]
__far char c1;
float f1;
void func()
{
      ++f1;
      c1 = 5;
}
```

[Workaround]

Immediately after incrementing or decrementing a floating point variable, insert a dummy code that references the variable by using a pointer.

```
[*.c]
__far char c1;
float f1;
void func()
{
      char *cp1;         /* Dummy variable */
      ++f1;
      *cp1;             /* Dummy dereference */
      c1 = 5;
}
```

[Action]

This issue will be corrected in CA78K0R Ver. 1.20.

A tool used to check whether this restriction applies is available.

Contact a Renesas Electronics sales representative or distributor for details.

Restrictions on Using CubeSuite with Real-Time OS

1. Product History

No.	Affected Real-Time OS	Usage Restrictions	CubeSuite Package										
			V1.00	V1.10		V1.12				V1.20			
			CubeSuite										
			V1.00	V1.10	V1.11	V1.12	V1.13	V1.14	V1.15	V1.20	V1.21	V1.30	V1.31
1	RX78K0R, RX850V4, RX850 Pro	Invalid strings are added at the end of object IDs displayed in system performance analyzer windows	–	×	○	○	○	○	○	○	○	○	○

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

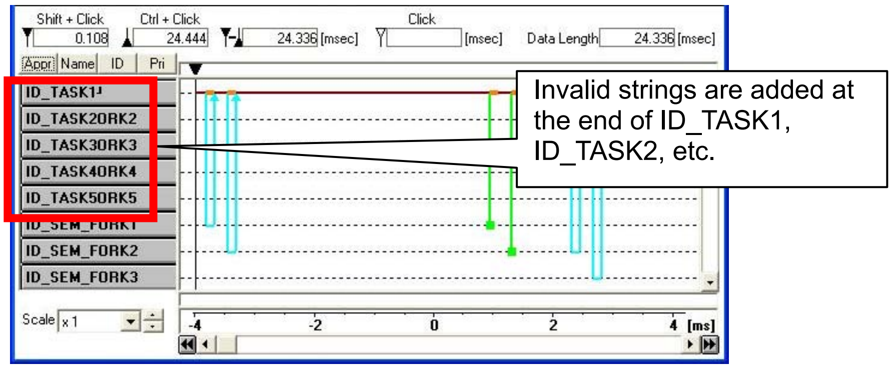
No. 1 Invalid strings are added at the end of object IDs displayed in system performance analyzer windows

[Applies to] RX78K0R, RX850V4, RX850 Pro

[Description]

Invalid strings are added at the end of object IDs (such as task IDs or semaphore IDs) that are displayed in system performance analyzer windows.

Example: Task IDs displayed in the Analyze window



[Workaround]

There is no workaround.

[Action]

This issue has been corrected in Ver. 1.11.

Building Tool Usage Restrictions

1. Product History

No.	Affected Tool	Usage Restrictions	CubeSuite Package									
			V1.00	V1.10		V1.12				V1.20		
			CubeSuite									
			V1.00	V1.10	V1.11	V1.12	V1.13	V1.14	V1.15	V1.20	V1.21	V1.30/ 1.31
1	CA78K0	Restriction whereby the `-qc` option is not saved in project files	×	×	×	○	○	○	○	○	○	○
2	All	Restriction on building files for which individual options are specified	×	×	×	○	○	○	○	○	○	○

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 Restriction whereby the `-qc` option is not saved in project files

[Description]

On the **Individual Compile Options** tab of the CA78K0, if **Optimization**, **Perform optimization**, and then **Yes(Detail setting)** is clicked to open **Optimization(Details)**, and **Not use sign extended calculation for char** is selected and then the project file is reloaded, **Yes** is selected for **Not use sign extended calculation for char**.

[Workaround]

Select **No** for **Perform optimization** and specify the necessary optimization options for **Other additional options**.

[Action]

This issue has been corrected in CubeSuite Ver. 1.12.

No. 2 Restriction on building files for which individual options are specified

[Description]

On the **Individual Compile Options** or **Individual Assemble Options** tab of a build tool, if **Yes** is selected for **Use whole include paths specified for build tool**, the source file for which individual options are specified is always compiled during build processing.

[Workaround]

Select **No** for **Use whole include paths specified for build tool**, and specify the include paths specified using the general option for the individual options.

[Action]

This issue has been corrected in CubeSuite Ver. 1.12.

Programming Tool Usage Restrictions

1. Product History

No.	Affected Tool	Usage Restrictions	CubeSuite Package							
			V1.00	V1.10		V1.12		V1.20		V1.40
			CubeSuite							
			V1.00	V1.10	V1.11	V1.12	V1.13/ 1.14/ 1.15	V1.20	V1.21/ 1.30/ 1.31	V1.40
1	QB-Programmer	Restriction on aborting CubeSuite when a specific HEX file is loaded	×	×	×	×	○	○	○	○

×: Applicable, ○: Not applicable, –: Not relevant

2. Restriction Details

No. 1 Restriction on aborting CubeSuite when a specific HEX file is loaded

[Description]

When the HEX file which meets following condition is read by a property of QB-Programmer, CubeSuite aborts.

[Condition]

The HEX file by which the value which has added 32 bytes to the file size of the HEX file will be 4096 bytes of multiple

[Addition]

If there is only one HEX file which can be chosen by a property of QB-Programmer, when opening a property of QB-Programmer, HEX file is loaded. If multiple HEX files can be chosen by a property of QB-Programmer, when choosing the file name on a property, HEX file is loaded.

[Workaround]

There is no workaround.

[Action]

This issue has been corrected in CubeSuite Ver. 1.13.