NEC

Customer Notification

78K0/78K0S™ Series

CCIAR-CDR-78K0/K0S

Operating Precautions

Embedded Workbench 78K0/K0S
Integrated Development Environment

Document No. U18067EE2V9IF00
Date Published: January 2008

DISCLAIMER

The related documents in this customer notification may include preliminary versions. However, preliminary versions may not have been marked as such.

The information in this customer notification is current as of its date of publication. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC's data sheets or data books, etc., for the most up-to-date specifications of NEC PRODUCT(S). Not all PRODUCT(S) and/or types are available in every country. Please check with an NEC sales representative for availability and additional information.

No part of this customer notification may be copied or reproduced in any form or by any means without prior written consent of NEC. NEC assumes no responsibility for any errors that may appear in this customer notification. NEC does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC PRODUCT(S) listed in this customer notification or any other liability arising from the use of such PRODUCT(S).

No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC or others. Descriptions of circuits, software and other related information in this customer notification are provided for illustrative purposes of PRODUCT(S) operation and/or application examples only. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

While wherever feasible, NEC endeavors to enhance the quality, reliability and safe operation of PRODUCT(S) the customer agree and acknowledge that the possibility of defects and/or erroneous thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects and/or errors in PRODUCT(S) the customer must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

The customer agrees to indemnify NEC against and hold NEC harmless from any and all consequences of any and all claims, suits, actions or demands asserted against NEC made by a third party for damages caused by one or more of the items listed in the enclosed table of content of this customer notification for PRODUCT(S) supplied after the date of publication.

Applicable Law:

The law of the Federal Republic of Germany applies to all information provided by NEC to the Customer under this Operating Precaution document without the possibility of recourse to the Conflicts Law or the law of 5th July 1989 relating to the UN Convention on Contracts for the International Sale of Goods (the Vienna CISG agreement).

Düsseldorf is the court of jurisdiction for all legal disputes arising directly or indirectly from this information. NEC is also entitled to make a claim against the Customer at his general court of jurisdiction.

If the supplied goods/information are subject to German, European and/or North American export controls, the Customer shall comply with the relevant export control regulations in the event that the goods are exported and/or re-exported. If deliveries are exported without payment of duty at the request of the Customer, the Customer accepts liability for any subsequent customs administration claims with respect to NEC.

Notes: (1) "**NEC**" as used in this statement means NEC Corporation and also includes its direct or indirect owned or controlled subsidiaries.

(2) "**PRODUCT(S)**" means 'NEC semiconductor products' *(NEC semiconductor products* means any semiconductor product developed or manufactured by or for NEC) and/or 'TOOLS' *(TOOLS* means 'hardware and/or software development tools' for NEC semiconductor products' developed, manufactured and supplied by 'NEC' and/or 'hardware and/or software development tools' supplied by NEC but developed and/or manufactured by independent 3rd Party vendors worldwide as their own product or on contract from NEC)

Table of Contents

(A) Table of Operating Precautions for IDE (EW78000)

No.	Outline	EW78000				
	Version	2.31e				
A1	**Temporary File isn't deleted**	×				

×: Applicable
✓: Not applicable
- : Not checked

(B) Table of Operating Precautions for Assembler (A78000)

No.	Outline	A78000				
	Version	3.34A	3.34B			
B1	**Backslash as last character in a comment**	×	×			
B2	**Project using only absolute segments**	×	×			
B3	**Assembler Error Message isn't displayed**	×	✓			

×: Applicable
✓: Not applicable
- : Not checked

(C) Table of Operating Precautions for C-Compiler (ICC78000)

No.	Outline	ICC78000				
	Version	3.34A	V3.34B			
C1	**Reverse keyword order**	×	×			
C2	**Missing external variable entry, if a global variable is only used in Inline-Assembler**	×	×			
C3	**Comparison of Bit-Variable and Bitfield - Elements**	×	×			
C4	**Wrong parameter calculation**	×	✓			
C5	**Switch-case limitation of V-switch library function**	×	✓			
C6	**Wrong SFRP access (16bit special function registers)**	×	✓			
C7	**Internal Compiler Error**	×	✓			
C8	**Option -d works incorrect**	×	✓			
C9	**Parameter overwritten**	×	✓			
C10	**Wrong comparison of two static local variables**	×	×			
C11	**Unsaved register in ISR**	×	×			
C12	**Register overridden in library function Longjmp**	×	×			
C13	**Problem with redundant parenthesis in expressions that contain "||" or "&&"**	×	×			
C14	**Wrong negation during cast from unsigned int to unsigned long**	×	×			
C15	**Optimized away volatile declaration of a short variable on an absolute address**	×	×			
C16	**Wrong load from stack to register HL if speed optimization is used**	×	×			
C17	**Usage of callt in an interrupt function doesn't cause a compiler warning**	×	×			
C18	**Wrong Comparison of Bitfield-Element and Constant**	×	×			
C19	**Tool Internal Error in case of using conditional Expression inside a Switch Statement**	×	×			
C20	**Faulty change of execution order in case of using single bit-test instructions**	×	×			
C21	**Enum constant(s) of datatype long are not supported**	×	×			

×: Applicable
✓: Not applicable
- : Not checked

(D) Table of Operating Precautions for Linker (XLINK)

No.	Outline	XLINK				
	Version	4.53l	4.55l	4.58h	4.59f	
D1	**Unused Segments in multi-segment-definition line causes unnecessary error message**	×	×	×	×	
D2	**Unused entries in the IRQ-table are filled with 0x0000**	×	×	✓	✓	
D3	**Corrupted assembler file debug information**	✓	✓	×	✓	

×: Applicable
✓: Not applicable
- : Not checked

(E) Description of Operating Precautions for IDE (EW78000)

No. A1	Temporary File isn't deleted
	Details The Embedded Workbench generated under special conditions a temporary file (0001.tmp) in the project directory, which isn't deleted after exiting the Workbench. The problem occurs only if the used operating system is Windows 98 and at the first generation of a new project. *Workaround(s)* 1) Update the operating system to Windows 98 SE 2) Delete the temporary file manually.

(F) Description of Operating Precautions for Assembler (A78000)

No. B1	Backslash as last character in a comment

Details

If the last character in a comment is a backslash the pre-processor looks upon the next source line as a comment too. So the assembler code in the following line is not used.

Workaround

Don’t use the backslash as the last character in a comment

No. B2	Project using only absolute segments

Details

If a project uses only absolute segments, the linker generates a correct output file in XCOFF78K-format. But a NEC debugger (or simulator) before version 2.30 can't read the file (Error message; c0003(A): Failed in reading file).

Workaround

Use relative segments and do the segment definition in the XCL file.

No. B3	Assembler Error Message isn't displayed

Details
If the total character size of filename, path and message exceeds a certain limit and an assembler list file is generated an assembler error message may be not displayed. As the size of the message depends on the line number kind of error etc. the maximum length of path and file name is not easily predictable. Keeping the path and file name within 80 characters should work.

Workaround(s)

Don't enable the generation of an assembler list file.

(G) Description of Operating Precautions for C-Compiler (ICC78000)

No. C1	Reverse keyword order

Details

CALLT functions returning bit cannot be defined as given in the example below. The order of keywords shall be given as described in the workaround.

Example:

```
callt bit foo(void)
{
      ...
}
```

Workaround

Use the reversed keyword order:

```
bit callt foo(void)
{
      ...
}
```

No. C2	Missing external variable entry, if a global variable is only used in Inline-Assembler

Details

The global variable "var2" does not have an external entry! The result will be an unresolved external error during linking.

Example:

```
extern char var1;
extern char var2;


void func1(void)
{
   var1 = 2;

    _ASM("MOV R0,#var1");
    _ASM("MOV R0,#var2");
}
```

Workaround

Use the global variable in a C statement, too

No. C3	Comparison of bit-variable and Bitfield-elements

Details

If a comparison of a bit-variable and a bitfield element is done, the sign of the bitfield element must be correct. A bit variable is always of the unsigned-type, but the type of a bitfield element depends on the type of the basis-data-type used for definition. The type of a bitfield-element is always the as the type of the used basis-data-type. An unsigned bit and a signed bit-field element are only equal, if both are '0'. If both have the value '1', they are not equal!

Example:

```
struct {
   int b0:1;
   …
}bField1;

struct {
   unsigned int b0:1;
   …
}bField2;
```

The type of `bField1.b0` is signed and the type of `bField2.b0` is unsigned

Workaround

To avoid problems it is recommended to use only unsigned bitfield-elements.

No. C4	Wrong parameter calculation

Details

The compiler generates wrong in following example, if speed-optimization is used. Registers R6 & R7 are used even if RP3 is used as a pointer to stack. This results to an incorrect calculation of the expression 'loc1+p3'.

Example:

```
extern unsigned char f1 (unsigned int  p1,const unsigned char* p2,
                         unsigned char p3);


void test  (unsigned int p1, unsigned char p2, unsigned char p3,
            const void* p4, unsigned char p5)
{
   unsigned int loc1;
   loc1 = ((0) * (p2) + (p1) + 4);
   f1(loc1 + p3, p4, p5);
}
```

Workarounds

There two workarounds to solve the problem:

1) Use size-optimization (compiler option -z0 … -z9) instead of speed optimization.
2) Do the calculation of the first parameter in a separate instruction before the function call:

Example:

```
void test  (unsigned int p1, unsigned char p2, unsigned char p3,
            const void* p4, unsigned char p5)
{
   unsigned int loc1;
   loc1 = ((0) * (p2) + (p1) + 4);
   loc1 += p3;
   f1(loc1, p4, p5);
}
```

No. C5	Switch-case limitation of V-switch library function

Details

There is a limitation in the library function for V-switches. The maximum number of cases that can be handled is 127.

Workarounds

There two workarounds:

1) Use the compiler-option **-Es** to avoid the V-switch library function and generate inline-code for the switch case.

2) Split the switch case to less than 127 cases.

No. C6	Wrong SFRP access (16bit special function registers)

Details

If the compiler option -u (=enable byte-alignment) is used, the compiler would generate two 8 bit accesses instead of one 16bit access to 16bit special function registers. If only a 16bit access is allowed, the generated code is wrong.

Workarounds

There two workarounds:

1) Don't use the option -u (-> word-alignment).

2) Use the inline-assembler to access 16bit special function register correctly.

No. C7	Internal Compiler Error

Details

If the compiler option -v1 (= Target "78K/0, 7801X and above, and 780XXX") and the compiler option -u (=enable byte-alignment) is used, the following sample code (?-mark expression with a return value containing an arithmetic operation) generates an internal compiler error:

```
* * *   I N T E R N A L   E R R O R   * * *

In function: label_trees_C03
Diagnostic: Illegal state
Line: 5   P0: 0    P1: 0
```

Example:

```
unsigned int number;
unsigned int test(void)
{
   return number > 100 ? number / 10 : 0;
}
```

Workarounds

Use an if-else statement to replace the conditional expression.

```
unsigned int workaround(void)
{
      if (foo > 100)
            return(foo/10);
      else
            return 0;
}
```

No. C8	Option -d works incorrect

Details

If more than one local variable per function is defined and the compiler option -d (use all local variables as static) is used, the compiler generates wrong code to access the local variables. Only the access to the first local variable is correct.

Example:

```
void int test(void)
{
   char l1,l2,l3;
   …
}
```

Workarounds

Use the keyword '`static`' to define static local variables instead of using the compiler option -d.

```
char workaround(void)
{
   static char l1,l2,l3;
   …
}
```

No. C9	Parameter overwritten

Details

If the first parameter of a function with an undefined number of parameters is of type char, this parameter is overwritten within the function.

Example:

```
#include "stdio.h"

void test(unsigned char ucPara1, const char *pszPara2, ...)
{
  va_list ArgPtr;
  …
}
```

Workarounds

Don’t use a parameter of type char as the first one (e.g. change the parameter order).

```
#include "stdio.h"

void test(const char *pszPara2, unsigned char ucPara1, ...)
{
  va_list ArgPtr;
  …
}
```

No. C10	**Wrong comparison of two static local variables**

Details

The code generated to compare two static local variables may be incorrect on optimization levels greater than 5. The problem occurs when either of the variables has been updated in certain ways, e.g. by register indirect assignments.

Example:

```
void function x( void)
{
  static unsigned char y;
  x  = 0;
  if (P5 & ((unsigned char) 0x02))    {
      x |= (unsigned char) 0x01 ;
   }
  if (P5 & ((unsigned char) 0x04))    {
      x |= (unsigned char) 0x02;
  if (x != y)  {
     // …
  }
}
```

Workarounds

Reduce the optimization level with #pragma statements.

```
void function x( void)
{
  static unsigned char x,y;
  #pragma optimize=z(5)
  x  = 0;
  if (P5 & ((unsigned char) 0x02))    {
      x |= (unsigned char) 0x01 ;
   }
  if (P5 & ((unsigned char) 0x04))    {
      x |= (unsigned char) 0x02;
  if (x != y)  {
     // …
   }
  #pragma optimize=z(9)
}
```

No. C11	**Unsaved register in ISR**

Details

If an optimization level greater 4 is used the registrerpair RP0 is not preserved in an ISR, if only a local variable is modified within the ISR.

Example:

```
static void interrupt [0x0E] test (void)
{
    unsigned char dummy = RXB0;
}
```

Workarounds

Reduce the optimization level within the ISR with #pragma statements.

```
static void interrupt [0x0E] workaround (void)
{
    #pragma optimize=z(4)
    unsigned char errorClearer = RXB0;
    #pragma optimize=z(9)
}
```

No. C12	**Register overridden in library function longjmp**

Details

A call to longjmp will, as the C standard requires, return via setjmp. However, the values of registers BC, DE and HL (only BC for banked systems) are not correctly restored to the values they had before the original call to setjmp, and the execution of code following setjmp may fail.

If the library functions longjmp and stjmp are used in an application can be checked in the linker list file (*.map).

Workarounds

The library functions for setjmp and longjmp have been rewritten. To use the corrected library versions, do the following:

- Include the file l08.s26 in your project. If you are using the banked memory model, you must define BANKED before assembling the file. If you do not define BANKED, the unbanked version will be assembled instead.
- Replace the file sysmac.h in your product with the new file. The new file will redefine the size of the jump buffer to 10 bytes instead of 6 bytes which was the old size.
- Rebuild your project. Specifically, recompile all files making "#include <setjmp.h>"

Note! The setjmp and longjmp library functions are not compatible with code using the -W option to store local variables in the short address work area.

The required files (archive: CCIAR-CDR-78K0_K0S-V334B-PAT1.zip) can be downloaded from the NEC EE internet page. **www.ee.nec.de/update**

No. C13	**Problem with redundant parenthesis in expressions that contain "||" or "&&"**

Details

Redundant parenthesis in expressions that contain "||" or "&&" lead to wrong evaluation of the expression.

Workarounds

Avoid / remove redundant parenthesis.

For details how to check existing code for redundant parenthesis please contact the NEC Electronics (Europe) 78K support team at K0_support@ee.nec.de .

No. C14	**Wrong negation during cast from unsigned int to unsigned long**

Details

An integer expression of the type "long or unsigned long - unsigned short" is in many situations evaluated by negating the second operand and then adding the operands. As the first expression is a long integer, the second operand is implicitly converted to long as well. However, in this case, the second operand is negated before being cast to long. As the second operand is unsigned, this will lead to an error.

Please also see the release note as written in the icc78000.htm readmefile (`CFE0016/EW10052`).

Example:

```
unsigned char expr[8];
unsigned int array[8]={0,0,0,0,0,0,0,32000};
unsigned char i=7;

void test (void)
{
    expr[0] = ((33000 - array[i]) > 2000);    // should be '0'
}
```

Workarounds

Use explicit casting to unsigned :

```
expr[0]= ((33000u - Window[i]) > 2000);
```

No. C15	**Optimized away volatile declaration of a short variable on an absolute address**

Details

A volatile declared load of a short variable on an absolute address might be optimized away if no further instruction is executed between two accesses to the variable. In the following sample keeps the compiler 'FLAGS' in register RP0, although it is defined as volatile and does the bit-test without the required reload. The behavior is independent of the used optimization.
The problem occurs only if there is no further instruction between the assignment of the value and the IF condition.

Example:

```
#define Flags (*((volatile unsigned short int *)(0xF8CC)))

void test(void)
 {
  Flags = 0x0000;
  if ( Flags & 0x0040 )
    Flags = 0x00;
 }
```

Workaround

Insert a ‘nop’ instruction between the value assign to Flags and the if statement:

```
  …
  Flags = 0x0000;
  _NOP();
  if ( Flags & 0x0040 )
  …
```

No. C16	**Wrong load from stack to register HL if speed optimization is used.**

Details

In case of using the speed optimization (any level) a 16 or 32bit load from the stack (except from the top of the stack) to register HL may result in a wrong value. This happens more likely with complex logical expressions using arrays, but it is hard to specify exactly when it happens. A way to detect when this happens, is to look in the list file for instructions on the form

```
MOV     Rx, [RP3+0x1839B95]
```

where the offset, in this case 0x1839B95, is very large. The value is a random one, so there is no guarantee that a small value indicates correct code, but a large value (larger than can fit in one byte) certainly indicates that the bug is present.

Example:

```
…
if (Structure.el1[index] + 10 >= MAX1
    && Structure1.el1[index] <= MAX1 + func1(index)
    && Structure1.el2[index] + 10 >= MIN1
    && Structure1.el2[index] <= MIN1 + 10)
{
    return (TRUE);
}
else
{
    return (FALSE);
}
…
```

Workarounds

a) Use size optimization
b) Use temporary local variables to simplify the expression (this also results in faster code)

```
…
temp1 = Structure.el1[index];
temp2 = Structure.el2[index];
if (temp1 + 10 >= MAX1
    && temp1 <= MAX1 + func1 (index)
    && temp2 + 10 >= MIN1
    && temp2 <= MIN1 + 10)
{
    return (TRUE);
}
else
{
    return (FALSE);
}
…
```

No. C17	**Usage of callt in an interrupt function doesn't cause a compiler warning.**

Details

The wrong usage of the keyword 'callt' in the definition of an interrupt function doesn't cause a compiler warning. The keyword 'call' is simply ignored, because a function cannot be of type interrupt and of the type 'callt' simultaneously. The generated code is correct.

Example:

```
callt interrupt [0x0010] void isr1(void)
{

}
```

Workarounds

Don't use the keyword callt in the definitions of interrupt functions.

No. C18	**Wrong Comparison of Bitfield-Element and Constant**

Details

In case of comparison of a constant and a bitfield-element, the compare generates code for a 1bit comparison instead of doing a 16bit-comparison according to the ANSI C integer promotion rules.

Example:

```
volatile struct   {
        unsigned char t0:1;
        unsigned int  t1:1;
   }t;

volatile unsigned int r1;

void test(void )
{

   if (t.t1 == 6){
      r1 = 101;
   }
   else{
      r1= 100;

   }
```

Workaround

Use explicit casting to force the correct comparison:

```
   if ((unsigned int) t.t0 == 6){
      r1 = 101;
   }
```

If the base type of the bitfield-element is ‘unsigned int’ a double typecast is necessary:

```
   if ((unsigned int)(unsigned char) t.t0 == 6){
      r1 = 101;
   }
```

No. C19	Tool Internal Error in case of using conditional Expression inside a Switch Statement

Details

In case of using maximum optimization level (-s9 or –z9) a tool internal error occurs, if a conditional expression of datatype int is used inside a switch statement.

```
In function: gp_handler - M00
Diagnostic: general protection fault
Line: 14   P0: 1    P1: 0
```

Example:

```
int a, b, c, d, tab[4];

void test(unsigned char i)
{
      switch(a == 1 || b == 2 ? c : d){
            case 1:
                  tab[i] = 1;
                  break;
            case 2:
                  tab[i] = 2;
                  break;
      }
}
```

Workarounds

- Reduce the optimization level:

```
void test(unsigned char i)
{
      #pragma optimize=s(8)
      switch(a == 1 || b == 2 ? c : d){
      #pragma optimize=default
         …
      }
}
```

- Use a temporary variable:

```
void test(unsigned char i)
{
      int temp;
      temp = a == 1 || b == 2 ? c : d;
      switch(temp){
         …
      }
}
```

No. C20	Faulty change of execution order in case of using single bit-test instructions

Details

In case of using maximum optimization level (-s9 or –z9) the compiler changes the access order in the following sample and executes the code occurring in the if- as well in the if else-path of an if-statement before test the condition.

The problem will only occur if there is an if-statement that is reduced to a BF or BT instruction and that both the if and the else block begins with code that is exactly the same and also modifies the variables that the BF/BT instruction test.

Example:

```
saddr unsigned char u8_array[2];
saddr unsigned char c8_Var1;

void test (void)
{

   if (0x80 & u8_array[0])
   {
      u8_array[0] = u8_array[0] & c8_Var1 & 0x0F;
      // … further code
   }
   else
   {
      u8_array[0] = u8_array[0] & c8_Var1 & 0x0F;
      // … further code
   }
}
```

Workarounds

Reduce the optimization level or define the array as volatile

No. C21	**Enum constant(s) of datatype long are not supported**

Details

Against the description in the manual, enums using the integer type long are not supported. The compiler generates the following error message:

```
Error[83]: "enum" constant(s) outside "int" or "unsigned int" range
```

Example:

```
enum
{
  e0,
  e1,
  e2 = 0x12345678UL
};
```

Workarounds

None.

(H) Description of Operating Precautions for Linker (XLINK)

No. D1	Unused Segments in multi-segment-definition line causes unnecessary error message

Details

If an unused segment is defined in a multi-segment-definition-line and there are no more bytes available the linker generates an unnecessary error message telling that the unused segment can not be located:

Error[e16]: Segment NO_INIT (size: 0 align: 0) is too long for segment definition.
At least 0 more bytes needed.

Example:

```
-Z(DATA)CSTACK+20,NO_INIT,IDATA2,UDATA2,ECSTR,TEMP=FE00-FE1F
```

The segment CSTACK uses the complete address area, but even if the following segments are not used the linker generates the error message and stops the link process.

Workaround

Change the segment order: Unused segments should be defined before the last used segment:

```
-Z(DATA)NO_INIT,IDATA2,UDATA2,ECSTR,TEMP,CSTACK+20=FE00-FE1F
```

No. D2	Unused entries in the IRQ-table are filled with 0x0000

Details

Unused entries in the IRQ-table are filled by the linker with 0x0000. In case of using the MINCUBE emulator this causes an error and the application cannot be downladed into the emulator.

Workaround

Update the linker to version V4.58h or later

No. D3	**Corrupted assembler file debug information**

Details

When generating an output file in the xcoff78k format, XLINK failed to generate debug information for an assembler module if it was the first module encountered on the command line. In case of using the Embedded Workbench the files are linked in alphabetical order.

Workarounds

a) Update the linker to version V4.59f
b) In case of using the command line interface:
   Change the link order of the project files in the command line so that the first file is a C source file.
c) In case of using the Embedded Workbench:
   Rename the assembler file, so that the first file of your project is a C source file.

(I) Valid Specification

Item	Date published	Document No.	Document Title
1	November 1997	ICC78000-4	78000 C Compiler Programming Guide
2	November 1997	A78000-6	78000 Assembler, Linker and Librarian Programming Guide
3	November 1997	EW78000-1	78000 Windows Workbench Interface Guide
4	November 1997	CL78000-1	78000 Command Line Interface Guide
5	October 2001	XLINK-453E	XLINK Linker™ and XLIB Librarian Reference Guide
6	September 2000	IARCLIB-1	C Library Functions Reference Guide
7	December 2001	ILG-2	Installation and Licensing Guide for Embedded Workbench™

(J) Revision History

Item	Date published	Document No.	Comment
1	06-06-2002	DTOP0006V10	First release.
2	11-07-2002	DTOP0006V11	Item C4 added
3	12-08-2002	DTOP0006V12	Item C5 added
4	05-12-2002	DTOP0006V13	Item A2 added
5	16-12-2002	DTOP0006V14	Item C6 added
6	10-01-2003	DTOP0006V15	Item C7 added
7	19-03-2003	DTOP0006V16	Item C8 added
8	11-04-2003	DTOP0006V17	Item C9 added
9	08-08-2003	DTOP0006V18	Information for tool package V3.34B added
10	24-11-2003	DTOP0006V19	Item C10, C11 and C12 added
11	07-05-2004	DTOP0006V20	Item C13 added
11	02-06-2004	DTOP0006V21	Item C14 added
12	05-07-2004	DTOP0006V22	Item C15, D2 added
13	20-10-2004	DTOP0006V23	Item D3 added
14	15-11-2004	DTOP0006V24	Item C16 added
15	12-01-2005	DTOP0006V25	Item C17 added
16	22-09-2005	DTOP0006V26	Item C18 added
17	04-10-2005	DTOP0006V27	Item C19 added
18	20-11-2006	U18067EE2V8IF00	Item C20 added, new NEC Electronics global document number
19	15-01-2008	U18067EE2V9IF00	Item C21 added