National Semiconductor®

CompactRISC™

CR16 Development Toolset
Version 2.1
Release Letter

Part Number: 433521772-001

August 1998

PREFACE

This release letter describes the CompactRISC Development Toolset, Version 2.1, for the CompactRISC CR16 CPU cores (CR16A and CR16B). This is a product quality release.

This release letter includes: major component description, product enhancement list, list of incompatibilities with the previous release (CompactRISC Toolset Version 2.0), installation procedure, known software errors, and fixed software errors.

The *CompactRISC Toolset - Introduction* provides an overview of the toolset, and includes guidelines for its correct use.

The *CR16A Programmer's Reference Manual* and *CR16B Programmer's Reference Manual* provide full descriptions of the CR16A and CR16B CPU architectures.

To get the most out of the CompactRISC Toolset, you should familiarize yourself with these manuals.

For technical support and additional information, visit our Web site at:

http://www.CompactRISC.com

or contact the National Semiconductor support center in your area, as listed on the back of this document.

CompactRISC is a trademark of National Semiconductor Corporation.
National Semiconductor is a registered trademark of National Semiconductor Corporation.
Dinkum and Dinkumware are registered trademarks of Dinkumware, Ltd.

CONTENTS

1.0 RELEASE PACKAGE CONTENTS

Part/Lit Number	Description
440521772-001	One CD-ROM, containing the following: • CompactRISC CR16 Development Toolset Version 2.1 and on-line documentation (in PDF format). • Debugger Communication Interface Version 0.80 software package.
424521772-001	CompactRISC Toolset - Introduction.
410521426-001	CR16A Quick Reference Card.
633151-001	CR16B Quick Reference Card.
633150-001	CR16B Programmer's Reference Manual.
433521771-001	DbgCom Version 0.80 Release Letter.
433521772-001	This release letter.

2.0 PRODUCT DESCRIPTION

This package contains cross-development software tools for the CompactRISC CR16 CPU cores (CR16A and CR16B), which are members of National Semiconductor's CompactRISC microprocessor family.

The package is intended for IBM-PC compatible computers, with a 486, or Pentium, CPU, running Microsoft Windows 95 or Microsoft Windows NT 4.0.

The following table shows the major software components of the package:

Component	Description
crcc	ANSI-C optimizing compiler. Generates CR16 assembly files, object files, or executable object files.
crasm	Macro Assembler for the CR16 assembly language. Generates relocatable CR16 object files.
crlink	Links CR16 object files and library files. Generates CR16 executable object files.
crdb	Source-level graphic debugger for CR16 programs.
crview	CR16 object file viewer. Displays all parts of CR16 object files in a formatted manner.
crprom	PROM file generator. Converts executable object files into standard PROM programmer formats, such as Intel-hex.
crlib	Librarian (archiver). Creates and maintains libraries of CR16 object files.
libc	C run-time library based on Dinkum C library by Dinkumware Ltd. Includes ANSI-C standard functions, and CR16-specific functions.
libstart	Development board support start-up library. Includes the default start-up routine, some initialization routines, and Target Monitor (TMON) interface routines. Use this library in conjunction with any new TMON (e.g., TMON Version 2.1.0 or higher).
libadb	Development board support start-up library. Includes a start-up routine, some initialization routines, and Target Monitor (TMON) interface routines. Use this library in conjunction with any old TMON (e.g., TMON Version 2.0.x or lower).
libhfp	Floating-point emulation library. Provides floating-point emulation routines which are automatically called by the C compiler when floating-point operations are involved.
libd	Dummy floating-point emulation library. To be used by programs that do not require floating-point support.
include	Common C header files. Located in the **include** subdirectory under the CompactRISC toolset directory. The header files include both ANSI-C standard header files, and CR16 specific header files.

Component	Description
src	Source files of **libstart**, **libadb** and Target Monitors (TMONs) for commonly used development boards. Located in the **src** directory, under the CompactRISC toolset directory. They can be used for customization.

The following table shows the on-line documentation components of the package:

Component	Description
Documentation Icon in the CR Tools Group (Documentation Roadmap)	This on-line document provides links to all the other documentation, and on-line Acrobat help, and a detailed explanation for first-time Acrobat Search users. (Available only if you choose to install the on-line documentation.)
CR16A Programmer's Reference Manual	Describes the CR16A CPU architecture, including the programming model, detailed description of the instruction set, and other related topics.
CR16B Programmer's Reference Manual	Describes the CR16B CPU architecture, including the programming model, detailed description of the instruction set, and other related topics.
Introduction	Introduces both the 1.2 and 2.0 versions of the CompactRISC Toolset. It provide an overview of the tools, a short tutorial, and Embedded Application implementations for the CompactRISC architecture.
C Compiler Reference Manual	Describes the CompactRISC C compiler, and its usage. In addition, it describes the CompactRISC C library.
Assembler Reference Manual	Describes the CompactRISC assembly language, and the CompactRISC assembler.
Object Tools Reference Manual	Describes the CompactRISC linker, and additional object utilities.
Debugger Reference Manual	Describes the CompactRISC Debugger, and its usage.
DbgCom User Guide	Describes the installation procedure for the Debugger Communication Interface, which is required for this release.
CompactRISC Toolset Release Letter	Release letter for the CompactRISC Development Toolset version 2.1.
DbgCom Release Letter	Release letter for the Debugger Communication Interface, which is required for this release.
Application Notes	Application notes on subjects of interest to CompactRISC Toolset users.

3.0 PRODUCT ENHANCEMENTS

- The CompactRISC ANSI-C library is based on the Dinkum C library, by Dinkumware Ltd. This package provides an upgrade to version 2.01 of the Dinkum C library.
- Support for code Bank Switching (**-BS**) is added (only for CR16A-based chips with Bank Switching hardware support).
- The **printf** function is extended to support printing of **far** strings. You can now use the **%S** format to print **far** strings.
- The assembler enables double slash (//) comment style.
- The debugger supports verbose mode for displaying all DbgCom calls and Target Monitor (TMON) protocol.
- The CR16B simulator supports a hardware breakpoint.
- Speed is improved for both the functional and performance simulators.
- Initialized variables can be placed in a user-defined section.

4.0 INCOMPATIBILITY

- The compiler CPP predefined symbol list has been modified. The compiler now predefines only symbols with leading and trailing underscores. Symbols that did not include leading and trailing underscores are not no longer predefined. For the full list, see the *CompactRISC - C Compiler Reference Manual*.

5.0 INSTALLATION PROCEDURES

5.1 INSTALLATION OPTIONS

You have a choice of three installations:

1. Tools only.
   Install only the CompactRISC Toolset.

1. On-line documents only.
   Install the CompactRISC reference manuals, and, optionally, the Adobe Acrobat Reader.

2. Full installation (Customizable).
   Install both the CompactRISC Toolset and the reference manuals, and, optionally, the Adobe Acrobat Reader. This type of installation is customizable and you can select/deselect components to be installed.

The reference manuals are in Adobe PDF format. The Adobe Acrobat Reader, supplied on the CD-ROM, includes full Search facilities. If you do not have an Acrobat Reader with the Search Plug-In, you should install the Reader supplied.

5.2 BEFORE YOU INSTALL

To install this version you must have a software access key. This can be found on the back of your CD-ROM package.

The package requires approximately 8 Mbytes of disk space for the Toolset, and a further 18 Mbytes for the documentation and Adobe Acrobat Reader.

To use the CompactRISC Debugger (CRDB) you *must* first install the CompactRISC Debugger Communication Interface (DbgCom) version 0.80, or higher. For more details see the *CompactRISC Debugger Communication Interface (DbgCom) User Guide*.

If you have already installed a previous version of the Toolset, replace the previous version by uninstalling it, using the uninstall icon of the old version, and then install the new version.

We recommend closing any running application, before you start the installation.

5.3 WINDOWS 95 / NT 4.0 INSTALLATION

Carry out the following steps:

1. Insert the CompactRISC Toolset CD-ROM into your CD-ROM drive.

2. From the **Start** menu, select **Run**. In the Run dialog box, type the following command:
   e:\setup.exe (where **e:** is your CD-ROM drive)

3. Select the CR16 Version 2.1 installation in the CompactRISC main installation program.

4. Follow the instructions in the installation program.

5. The installation program creates a new folder under the Program sub-menu of the Start Menu. This folder contains the following:
   - CompactRISC Toolset Debugger.
   - The Debugger on-line help.
   - A command prompt window (i.e., DOS box) configured for the CompactRISC Toolset environment.
   - A registration card.
   - An Uninstallation program.
   - Documentation files in Adobe Acrobat PDF format (optional).

5.4 UNINSTALL

The CompactRISC folder (located under the Programs sub-menu of the Start Menu), includes an Uninstall program. This program removes all the files and directories belonging to the CompactRISC Toolset package, and deletes all the modifications that were applied to your environment during installation. You can also invoke the Uninstall program from the 'Add/Remove programs' control, located in the Control Panel.

5.5 GENERAL

- After the installation program has terminated, reboot your PC.
- If you choose to install the Adobe Acrobat Reader, the installation program launches the Acrobat Reader installation program. After the Toolset installation has been completed, look for the Acrobat installation window, or icon, to complete the Acrobat Reader installation.
- During the installation procedure, when prompted for a directory, make sure that you specify an absolute path (e.g., **`c:\temp`** and not **`c:temp`**).
- The value of the **`path`** variable is limited in size. The installation procedure may cause the **`path`** variable to exceed this limitation. If this happens, Windows truncates the value of the **`path`** variable, and the Toolset can not function.

5.6 LIMITATIONS

- Under Windows NT, you can not install the CompactRISC Toolset in a directory that has blank characters in its name, other than the standard **`Program Files`** directory.
- You can not define the temporary directory of the CompactRISC Tool set (i.e., **`TMPDIR`**) in a directory which has blank characters in its name, even though this is allowed under Windows 95 and Windows NT (e.g., **`Temp Files`** is not a valid name).

6.0 LIMITATIONS

- There is limited debugging support for the code overlay allocation feature of the linker (**-0**). You can only debug overlayed functions in disassembly mode, and without any symbolic information.
- There is no debugging support for the code Bank Switching mechanism.
- The pipe status information in the performance log file, and the wait state mechanism, were only partially tested.

7.0 KNOWN SOFTWARE ERRORS

7.1 INSTALLATION

Issue 1297

Problem The Registration Card may not open when you select its icon in the CompactRISC folder.

Workaround Open the Windows Write application first, then open the Registration Card using the File Open menu.

7.2 COMPILER

Issue 697

Problem The header file **sys\stat.h** requires the header file **sys\types.h**, but does not include it.

Workaround Include the file **sys\types.h** before you include **sys\stat.h**.

Issue 806

Problem A false overflow warning is issued by the compiler in the following case:

```
unsigned int i = ~0;
```

Workaround Cast the expression to unsigned int:

```
unsigned int i = (unsigned int)~0;
```

Issue 846

Problem An illegal operation is detected during compilation if the program contains the following floating-point constant expression: **`inf/inf`**. The compiler attempts to calculate this expression at compile time, however this kind of calculation causes a trap on the 486/Pentium processor.

Issue 924

Problem The compiler does not generate symbolic information for a structure member referring to another structure with an incomplete definition.

Example

```
struct t1 *p1;

struct t2 {
  int i;
  struct t1 *p2;
};

struct t1 {
     int a,b;
     struct t1 *p3;
};

struct t2 s;
```

The type of structure member **`p2`** is unknown at debug time, because the full definition of **`struct t1`**, on which it is based, appears later.

Note that **`p1`**, which is not a structure member, is not affected by this problem.

Symbolic information is generated for **`p3`**, which is a structure member that refers to the structure to which it belongs.

Workaround Place a structure declaration before any reference to this structure.

Issue 1008

Problem The compiler may generate wrong code for switch statements if the size of the switch's jump table exceeds 64 Kbytes, and the code is not generated with the **`-O`** flag.

Issue 1080

Problem The compiler generates wrong symbolic information for a function argument of type **unsigned char** that is declared in traditional C (pre ANSI-C) form, and resides on the stack. The debugger shows the wrong type for this argument. In the following example, incorrect type information is generated for the argument **e**.

Example

```
foo(a, b, c, d, e)
int a, b, c, d;
unsigned char e;
{
   ...
}
```

Workaround Declare the function in ANSI-C form:

```
foo(int a, int b, int c, int d, unsigned char e)
{
   ...
}
```

Issue 1351 (CR16B Large model only)

Problem It is not possible to assign an address greater than 128K to a function pointer.

Example

```
int (*p)();

main(){

    p = (int(*)())(120000);

}
```

This results in an error.

Workaround Use a constant variable to perform this assignment.

Example

```
int (*p)();

const unsigned long addr = 120000;

main(){

    p = (int(*)())addr;

}
```

Issue 1603 (CR16A only)

Problem When optimizing for speed, the save emulation optimization option is also used. As save-emul is a space optimization, this may slightly reduce the speed.

Workaround Use **`crcc -Wc,-O2`** instead of **`crcc -O`**.

Issue 1710: (CR16A only)

A false warning is given when casting a function pointer to unsigned short.

Example

```
unsigned short stk;
void (*ptr)(unsigned char arg);
void foo(void)
{
 stk = (unsigned short) ptr;
}
```

Issue 1711

Problem A false warning is given when accessing array elements with offset greater than 32767.

Example

```
void foo(void)
{
 int array[20000];
 array[19999]==19999;
}
```

7.3 ASSEMBLER

Issue 1427 (CR16B only)

Problem The **movd** instruction does not accept negative offsets.

Example The following results in an error.

```
movd $a-10,(r1,r0)
```

7.4 LINKER

Issue 646

Problem The linker first processes any object files and libraries that are specified in the invocation line, and then processes any input object files specified in the linker directives file. As a result, references to libraries that are specified in the invocation line may not be resolved.

Workaround Either specify all the object files and libraries in the invocation line, or include all of them in a file named *filename*, and invoke the linker with the following command:

crlink @*filename* -d linker.def

Issue 694

Problem The linker does not issue a warning when a user request to direct a specific input section to an output section is ignored. Consider the following example:

```
sections {
   .text into(mem1) : { *(.text) }
   .text into(mem2) : {x.o(.text) }
}
```

You want to separate the **.text** input section from the object file **x.o** from the reset of the **.text** input sections, and direct it to a **.text** output section which is located in **mem2**. However, all the **.text** input sections were already directed to the first **.text** output section which resides in **mem1**. The user request is ignored, and there is no warning.

Workaround The correct method is as follows:

```
sections {
   .text into(mem2) : { x.o(.text) }
   .text into(mem1) : {*(.text) }
}
```

First, the **.text** input section from **x.o** is directed to the **.text** output section that resides in **mem2**. Then, the **.text** sections from all other files are directed to the other **.text** output section (***(.text)** refers to **.text** input sections, from all input files, which were not yet associated with an output section).

Issue 931

Problem A pointer to code, or data, which resides at address 0 might be considered as a **NULL** pointer (**NULL** is defined as **(void *)0**), although it points to a real entity.

Workaround This problem is very rare. If it does affect your program, avoid allocating address 0 to data, or code, which may be pointed to by a pointer. You can use your own linker directives file to control the memory allocation.

7.5 LIBRARIES

Issue 1604

Problem The **far_strtok()** library function does not work properly.

Issue 1631

Problem There is no prototype for the **setjmp()** library function in the **setjmp.h** header file.

Workaround Include this prototype in your header file:

```
int setjmp(jmp_buf env)
```

where **jmp_buf** is a typedef that is defined in **setjmp.h**.

7.6 DEBUGGER

Issue 881

Problem The debugger modifies its current working directory after it opens, using a file browser, a COFF file, or a log file, which is not in the current working directory.

Issue 925

Problem If the debugger default working directory is a drive root directory e.g., **`c:\`**, it is impossible to perform the **`Save Setup`** operation. The debugger incorrectly adds another backslash to the name. For example, if the current working directory is **`c:\`** the debugger attempts to save the file **`c:\\crdb.env`**, which is an invalid path, and therefore fails. This problem also applies to directory names specified in the environment variable **`CRDBENV`**.

Workaround In the Windows properties of the debugger program, specify a non-root default directory, e.g., **`c:\myproj`**.

Issue 1101

Problem The debugger does not prevent you from executing the open log file as a command file, i.e., as the argument of the debugger's **`input`** command. If you attempt to perform such an operation the debugger stops responding.

Workaround Close the log file (by quitting the debugger) before you use it as an input command file.

Issue 1250

Problem The debugger does not handle correctly an undefined instruction (UND) trap. The result of an UND trap is unpredictable.

Issue 1263

Problem If the default font of your Windows operating system is not the standard font, the status bar of the debugger may be empty (i.e., the status bar shows no status information).

Workaround Configure your Windows to use the standard font as default.

Issue 1455

Problem If your Windows operating system is configured for high-resolution color (16 bit), the output window of the debugger may be corrupted.

7.7 SIMULATOR

Issue 1256

Problem The trace exception is not written properly in the performance simulator log file, i.e., **`excp ???`** is written instead of **`excp trc`**.

Issue 1688

Problem The **`Reset`** command overrides the performance simulator log file, even if the log file is off (i.e., disabled).

8.0 FIXED SOFTWARE ERRORS

8.1 COMPILER

Issue 962

Problem The compiler did not issue a warning when performing an **`sbrel`** optimization on a far variable. Note that the linker did issue a warning if a far variable resided in the **`sbrel`** mechanism.

Issue 979

The library function **`printf`** could not print far strings (**`%s`**).

Issue 1320

A false warning was generated when passing a function argument of type char.

Issue 1391

Problem The compiler generated wrong code when accessing array elements with offset greater than 4095.

Example The following statement resulted in the wrong code:

```
int ar[20000];
if (ar[19999]==19999);
```

The assembly code generated for the above statement:

```
    movw   $-962,r0 // instead of 39998
    loadw  _ar(r0),r0
    cmpw   $19999,r0
```

8.2 ASSEMBLER

Issue 1150 (CR16B only)

Problem The assembler did not correctly truncate the five MSBs of a 21-bit constant, using the **$hi** assembler macro, if the constant was greater than 256K.

For example:

```
movw   $hi(l1),r0    ; where l1 is greater than 256K
```

8.3 LINKER

Issue 1304

Problem The linker sometimes failed to reopen an existing error file if you used both the **-z** and **-o** flags.

8.4 CRVIEW

Issue 556

Problem When invoked with the **-T** option, **crview** disassembled code only from sections named **.text**, and not from any section of type **STYP_TEXT**. This was because the section type **STYP_TEXT** was also used for sections that contained data (e.g., the section **.rdata_2**, which contained read-only data, was generated with type **STYP_TEXT**).

8.5 DEBUGGER

Issue 1052

Problem If you changed **Core** or **Radix** in the debugger **Config** menu, while the focus was not in the main debugger menu, the check mark was not updated but remained in its previous position.

For example, if, under the above circumstances, you changed the default radix from 10 to 16, the check mark remained in the old position (10), although the actual radix was changed to 16.

Issue 1057

Problem The debugger **call** command does not work correctly after execution of the debugged program is completed.

Workaround Reset your application, and then issue the **call** command.

Issue 1084

Problem Sometimes, the debugger did not accept a breakpoint, using the breakpoint edit window, if the path of its source file started with "**..**" (e.g., **../temp/a.c**).

8.6 SIMULATOR

Issue 1262

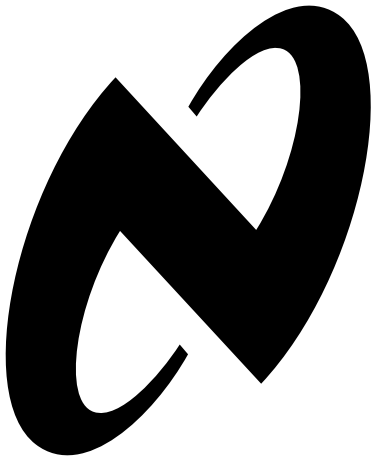
Problem The simulator did not simulate correctly a jump to one of the debugging monitor's trap handler (e.g., the **bpt** handler). This occurred only if you jumped to an address taken from the dispatch table after it was updated with debugging trap handlers (using SVC trap).

Issue 1338

Problem The performance simulator did not function correctly.

National Semiconductor supplies a comprehensive set of service and support capabilities. Complete product information and design support is available from National's customer support centers.

To receive sales literature and technical assistance, contact the National support center in your area.



Americas		**Fax:**	**1-800-737-7018**
		Email:	**support@nsc.com**
		Tel:	**1-800-272-9959**
Europe		**Fax:**	**+49 (0)1 80 5 30 85 86**
		Email:	**europe.support@nsc.com**
	Deutsch	**Tel:**	**+49 (0)1 80 5 30 85 85**
	English	**Tel:**	**+49 (0)1 80 5 32 78 32**
Japan		**Fax:**	**81-3-5620-6179**
		Tel:	**81-3-5620-6175**
Asia Pacific		**Fax:**	**65-250-4466**
		Email:	**sea.support@nsc.com**
		Tel:	**65-254-4466**

Visit us on the Worldwide Web at

http://www.national.com

or

http://www.CompactRISC.com

© 1998 National Semiconductor Corporation. All rights reserved.