To our customers,

## Old Company Name in Catalogs and Other Documents

On April 1$^{st}$, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1$^{st}$, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

RENESAS

# Debugging Interface

User's Manual HS6400DIIW4SE

# IMPORTANT INFORMATION

## READ FIRST

**READ this user's manual before using the Hitachi Debugging Interface (hereafter, called HDI).**

- **KEEP the user's manual handy for future reference.**

**Do not attempt to use the system until you fully understand its mechanism.**

## Target User of the System:

This system should only be used by those who have carefully read and thoroughly understood the information and restrictions contained in the user's manual. Do not attempt to use the system until you fully understand its mechanism.

It is highly recommended that first-time users be instructed by users that are well versed in the operation of the system.

## Purpose of HDI:

This system is a software and hardware development tool for systems employing the Hitachi microcomputer. This system must only be used for the above purpose.

## Improvement Policy:

Hitachi, Ltd. (including its subsidiaries, hereafter collectively referred to as Hitachi) pursues a policy of continuing improvement in design, performance, and safety of the system. Hitachi reserves the right to change, wholly or partially, the specifications, design, user's manual, and other documentation at any time without notice.

## Figures:

Some figures in this user's manual may show items different from your actual system.

## Other Important Things to Keep in Mind:

1. Examples described herein are meant merely to indicate the characteristics and performance of Hitachi's semiconductor products. Hitachi assumes no responsibility for any intellectual property claims or other problems that may result from applications based on the examples described herein.

2. No license is granted by implication or otherwise under any patents or other rights of any third party or Hitachi.

## All Rights Reserved:

## Trademarks:

Microsoft®, Windows® and WindowsNT® are registered trademarks of Microsoft Corporation in the United States and/or other countries.

IBM PC is the name of a computer administered by International Business Machines Corporation.

ELF/DWARF is the name of an object format developed by the Tool Interface Standards Committee.

All products or brand names used in the manual are trademarks or registered trademarks of their respective companies.

# Preface

## *About this Manual*

This manual explains the use of the Hitachi Debugging Interface (HDI) for Hitachi microcomputer development tools. The following section will provide a brief *Introduction* to the debugging interface and list its key features.

The following sections *Preparing to Debug, Looking at Your Program, Working with Memory, Executing Your Program, Stopping Your Program, Looking at Variables, Overlay Function, Selecting Functions,* and *Configuring the User Interface,* provide a "how to" guide to using HDI for debugging.

The next two sections *Menus* and *Windows* give in depth reference information about the operation and facilities available from these respective areas.

This manual assumes that the HDI is used on the English version of Microsoft® Windows®95 operating system running on the IBM PC.

The separate *Debugging Platform User's Manual* will typically provide:

A *Setting up* section that informs you about installing the debugging platform's hardware and software on your PC and verifying that all the components have been correctly installed.

A *Tutorial* section that takes you through the available features using some sample code.

A *Reference* section that describes the user interface that is specific to that debugging platform; for example, editing breakpoints, configuring the trace acquisition, etc.

## *Assumptions*

It is assumed that the reader has a competent knowledge of the C/C++ programming language, assembly-language mnemonics for the processor being debugged and is experienced in using Microsoft® Windows® applications on PC compatible computers.

## *Document Conventions*

This manual uses the following typographic conventions:

**Table 1   Typographic Conventions**

| CONVENTION | MEANING |
|---|---|
| **[Menu->Menu Option]** | Bold text with '->' is used to indicate menu options (for example, **[File->Save As...]** ). |
| FILENAME.C | Uppercase names are used to indicate file names. |
| "enter this string" | Used to indicate text that must be entered (excluding the " " quotes). |
| **Key+Key** | Used to indicate required key presses. For example, **Ctrl+N** means press the Ctrl key and then, while holding the Ctrl key down, press the N key. |
| ⊃ (The "how to" symbol) | When this symbol is used, it is always located in the left-hand margin. It indicates that the text to its immediate right is describing "how to" do something. |

# Contents

# Figures

# 1. Introduction

The Hitachi Debugging Interface (HDI) is a Graphical User Interface intended to ease the development and debugging of applications written in C/C++ programming language and assembly-language for Hitachi microcomputers. Its aim is to provide a powerful yet intuitive way of accessing, observing and modifying the debugging platform in which the application is running.

## 1.1 Key Features

- **Windows® GUI for debugging**

- **Intuitive interface**

- **On-line help**

- **Common "Look & Feel"**

**Notes**   1. For detailed information about debugging platform hardware, please refer to the separate *Debugging Platform User's Manual*.

2. The HDI does not run on Windows® version 3.1.

# 2. System Overview

HDI is a modular software system, utilizing self-contained modules for specific tasks. These modules are linked to a general purpose Graphical User Interface, which provides a *common look & feel* independent of the particular modules with which the system is configured.

## 2.1 User Interface

The HDI Graphical User Interface is a Windows® application that presents the debugging platform to you and allows you to set up and modify the system. Refer to a standard Windows® user manual for details on how to operate within a Windows® application.

## 2.2 Data Entry

When entering numbers in any dialog box or field you can always enter an expression instead of a simple number. This expression can contain symbols and can use the operators in the C/C++ programming languages. Use of C/C++ programming language features such as arrays and structures is only available if an object reader that supports C/C++ programming language debugging is in use.

In some dialogs, where there is a control expecting an End address, it is possible to enter a range by prefixing the value with a + sign. This will set the actual End address to be equal to the Start address plus the entered the value.

### 2.2.1 Operators

The C/C++ programming language operators are available:

+, -, *, /, &, |, ^, ~, !, >>, <<, %, (, ), <, >, <=, >=, ==, !=, &&, ||

### 2.2.2 Data Formats

Unprefixed data values will be taken as being in the default radix set by the [Setup->Radix] menu option. The exception is count field which use decimal values by default (independent of the current default system radix).

Symbols may be used by name and ASCII character strings can be entered if surrounded by single quote characters, e.g. 'demo'.

The following prefixes can be used to identify radices:

O'    Octal
B'    Binary
D'    Decimal
H'    Hexadecimal
0x    Hexadecimal

The contents of a register may be used by specifying the register name, prefixed by the # character, e.g.:

#R1, #ER3, #R4L

### 2.2.3 Precision

All mathematics in expression evaluation is done using 32 bits (signed). Any values exceeding 32 bits are truncated.

### 2.2.4 Expression Examples

```
Buffer_start + 0x1000
#R1 | B'10001101
((pointer + (2 * increment_size)) & H'FFFF0000) >> D'15
!(flag ^ #ER4)
```

### 2.2.5 Symbol Format

You can specify and reference symbols in the same format as in C/C++ programming language. Cast operators may be used together with symbols, and you can reference data after its type has been converted. Note the following limitations.

• Pointers can be specified up to four levels.
• Arrays can be specified up to three dimensions.
• No typedef name can be used.

### 2.2.6 Symbol Examples

```
Object.value             //Specifies direct reference of a member (C/C++)
p_Object->value          //Specifies indirect reference of a member (C/C++)
Class::value             //Specifies reference of a member with class (C++)
*value                   //Specifies a pointer (C/C++)
array[0]                 //Specifies an array (C/C++)
Object.*value            //Specifies reference of a pointer to member (C++)
::g_value                //Specifies reference of a global variable (C/C++)
Class::function(short)   //Specifies a member function (C++)
(struct STR) *value      //Specifies cast operation (C/C++)
```

## 2.3 Help

HDI has a standard Windows® context sensitive help system. This provides on-line information about using the debugging system.

Help can be invoked by pressing the **F1** key or *via* the Help menu. Additionally, some windows and dialog boxes have a dedicated help button to launch the help file at the appropriate content.

### 2.3.1 Context Sensitive Help

To get help on a specific item in the HDI help cursor can be used. To enable the help cursor, press **SHIFT+F1** or click the button on tool bar.

Your cursor then changes to include a question mark. You can then click on the item for which you require help and the help system will be opened at the appropriate content.

# 3. Preparing to Debug

This section of the manual describes all the facilities that are available in HDI for setting up the debugging platform ready to start debugging your program. You will learn how to select and configure a debugging platform with which to debug, and how to load your program to be debugged.

## 3.1 Compiling for Debug

In order to be able to debug your program at C/C++ source level, your C/C++ program must be compiled and linked with the debug option enabled. When this option is enabled, the compiler and linkage editor put all the information necessary for debugging your C/C++ code into the absolute file or management information file, which are then usually called *debug object files*.

**Note**  Make sure you have the debug option enabled on your compiler and linker, when you generate an object file for debugging.

If your debug object file does not contain any debugging information, then you can still load it into the debugging platform, but you will only be able to debug at assembly-language level.

## 3.2 Selecting a Debugging Platform

When HDI is launched, it will display a splash screen and then create its main window. The splash screen will clear to display the **Select Session** dialog. Choose the 'Create a new session on' option; select the appropriate debugging platform from the list; and click the [**OK**] button. If you select the wrong platform this dialog can be launched again by choosing the [**File->New Session...**] menu option.

```
 Select Session

  (• Create a new session on:                      OK

     E6000 H8S/2600 Emulator           ▼
                                                    Exit

  C  Previous session file:

     c:\hdi\tutorial\h8s\tutorial.hds   ▼          Browse...
```
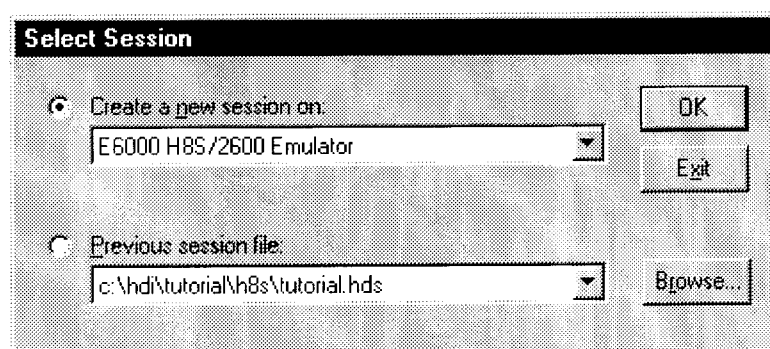
**Figure 3.1    Select Session Dialog Box**

HDI will load the platform's plug-in module and establish communications with the debugging platform. As the module loads, it will initialize any hardware or data structures and provide status messages on the status bar as the initialization progresses. When the debugging platform has been successfully initialized HDI will report "Link up" on the status bar.

## 3.3 Configuring the Debugging Platform

Before you can load a program into your debugging platform you must set it up to match your application's system. The items that must be set-up are typically device type, operating mode, clock speed and the memory map. It is particularly important to set-up the memory map, as you must have memory in the debugging platform to which your user program will be loaded.

### 3.3.1 Setup

To set-up the debugging platform configuration choose the [Setup ->Configure Platform...] menu option. You will be presented with a set-up dialog box specific to the debugging platform that you chose in the **Select Session** dialog box.

**Note**    For a detailed description of the features available in your debugging platform; please refer to the separate *Debugging Platform User's Manual*.

### 3.3.2 Mapping

For the debugger to correctly represent your user system, the memory map must be set up. It needs to know which areas in the device's address space are RAM, ROM, on-chip registers or areas where there is no memory.

When you select the device type and mode in the debugging platform configuration dialog, HDI will automatically set up the map for that device and the mode in which the processor is operating. For example in a device with internal ROM and RAM, the areas where these are located in the device's memory map will be set by default.

If you are using a device that does not have internal memory, or a device with external memory instead of, or in addition to, the internal memory, then you must tell the debugging platform that you have memory there. Also if you are trying to debug code with an emulator and wish to have some memory available in the address map that does not exist either internally in the device or externally in your user system, then you can map some *emulation memory* from the emulator to the address space for your application to use.

To edit the memory map configuration choose the [**Memory->Configure Map...**] menu item, or (for some platforms) *via* a pane on the **Configure Platform** dialog. The dialog box shown will be specific to the debugging platform that you chose in the **Select Session** dialog box. But, for example, with a hardware in-circuit emulator you will see something like:
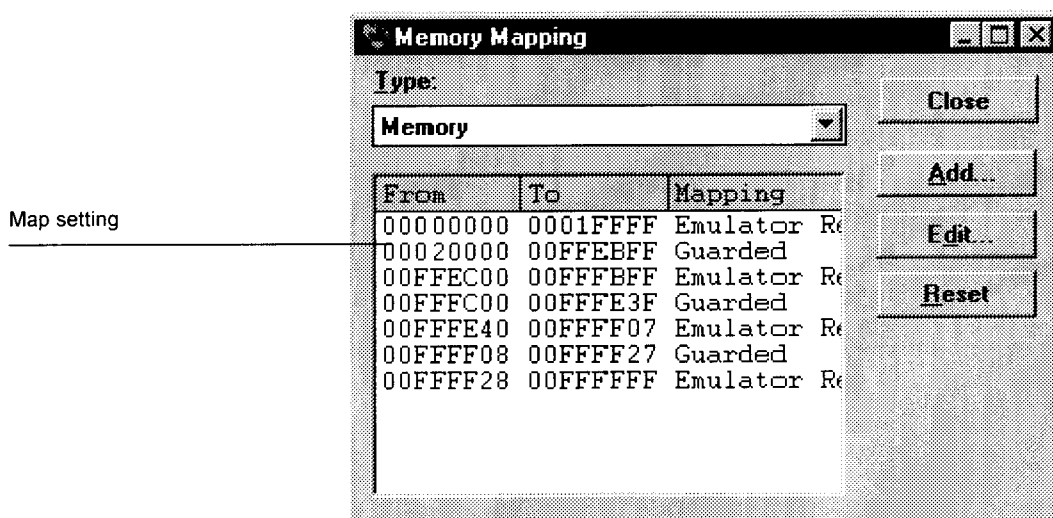
Map setting



**Figure 3.2    Memory Mapping Window**

The *Map Setting* area shows how the address space is currently mapped. It lists all address ranges covering the entire address space and the type of memory to which they are set - internal or external to the emulator and any access restrictions they may have, e.g. read only or guarded (no access). This includes those ranges set automatically by HDI and those you have set or modified.

Additional information about the memory mapping can be viewed in the **System Status** window's Memory pane. The *Device Configuration* area shows how the memory in the device's address space is configured, according to the device type and mode selected in the **Configure Platform** dialog box and any on-chip memory control settings. The *System Resources* area shows the status of mapping resources available to the system. For example in an emulator this will show the address ranges to which emulation memory has been allocated and which are currently available.

Clicking on the **[Reset]** button will set the system map setting back to the default for the current device type and mode.

To modify a map setting, select it and click on the **[Edit]** button or double-click on the map setting line.
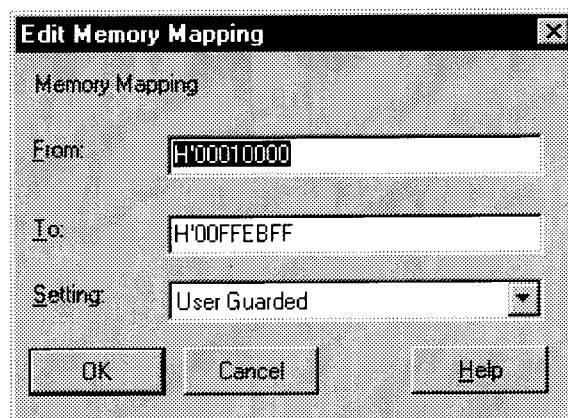


**Figure 3.3    Edit Memory Mapping Dialog Box**

You will then be able to modify the start and end addresses of the map range, and the memory type setting.

To add a new range click on the **[Add]** button, the Add Memory Mapping dialog box will open (it is the same as the **Edit Memory Mapping** dialog box but without any default values). Enter the start and end addresses of the map range, and the memory type setting for the new area. If the new range is in the middle of an existing range, HDI will automatically adjust the new range.

**Note**    Due to page length limitations in some emulators, the range addresses may not exactly match the entered addresses.

### 3.3.3 Status

To check the configuration and status of the debugging platform in
the **System Status** window choose the **[View->Status]** menu.



```
System Status                                               ◇ _ □ ☒

Item                              Status
Target Device Configuration  H8S/2xxx Advanced mode
                             ROM 00000000-000FFFFF
                             RAM 00FF0000-00FFFFFF

System Memory Resources      No information

Loaded Memory Areas          00001412 - 00001490
                             00000000 - 00000003
                             00000064 - 00000067
                             00000120 - 00000123
                             00000148 - 0000014B
                             000004A4 - 000004A5
                             00001000 - 00001411
                             00001492 - 000014A5

Session / Platform \ Memory / Events /
```

**Figure 3.4     System Status Window**

The System Status window is split into four panes:

1. Session - contains information about the current session including
   the connected debugging platform and the names of loaded files.

2. Platform - contains information about the current status of the
   debugging platform, typically including CPU type and mode; run
   status; and timing information.

3. Memory - contains information about the current memory status
   including the memory mapping resources and the areas used by the
   currently loaded object file.

4. Events - contains information about the current event (breakpoint)
   status, including resource information.

To update the status in the window on demand choose the **[Update]**
menu option from the popup menu.

## 3.4 Downloading a Program

Once you have made sure that there is memory in your system in
which to download your program, you can then proceed to download
a program to debug. To select an object file for debugging, choose the

[**File->Load Program...**] menu option to open the **Load Program** dialog box:
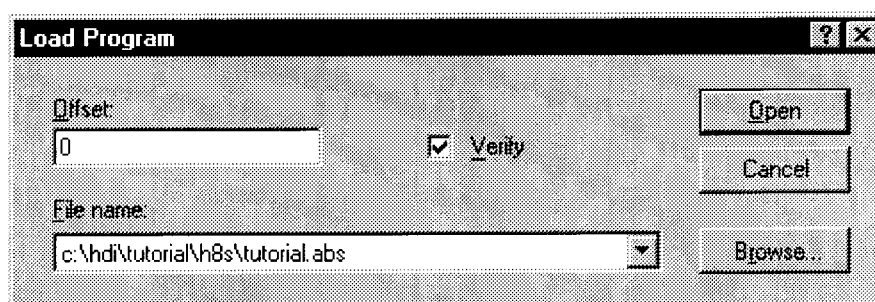


**Figure 3.5    Load Program Dialog Box**

The dialog includes a combo box containing a list of the previous four downloaded files and controls to allow an offset address to be used (suitable only for some object formats, e.g. S-Record) and to enable verification of the load. Verification checks that data downloaded to the platform can be correctly read back - this causes slower download speeds, so it is recommended that you only verify if you suspect a problem with memory or with the link file.

If the combo box does not include the file you require, it is possible to either enter the file name directly into the edit area, or to use the Browse button to search for the file you want to download.
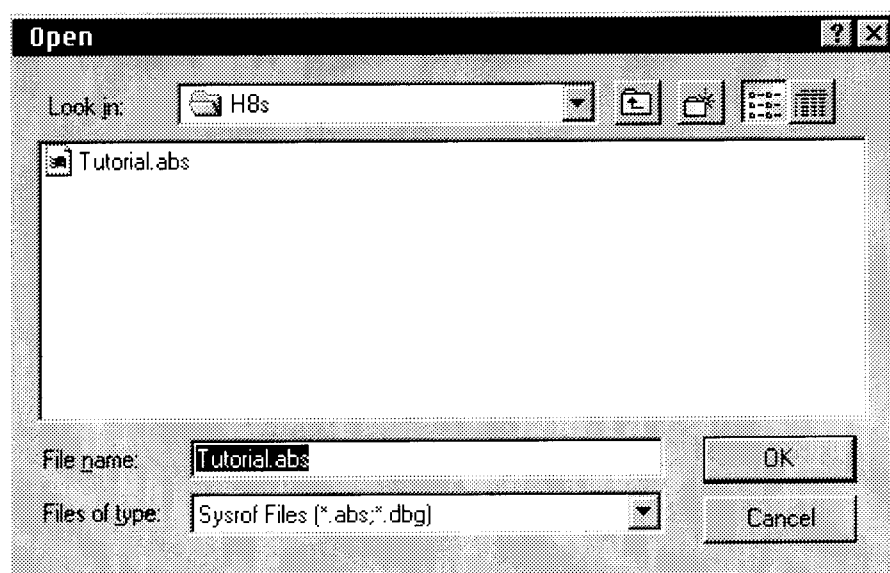


**Figure 3.6    Open Dialog Box**

To select a file to download from the browser dialog, first select the type of file to display in the list area by clicking in the Files of type field (see also *section 11.6 Customizing the File Filters*) and then click on the file type that you require.
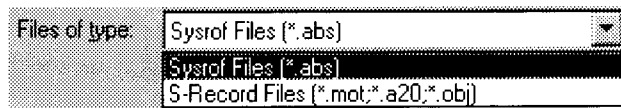
**Figure 3.7    File Type Selection**

The file list will then be updated with the files available, from which your selection can be made. Directory and drive navigation is possible using the standard windows file open dialog box controls, to the right of the file list. Alternatively the file name can be typed into the File name: field directly. The **[OK]** button will return to the Open Program dialog with the File name field set to the path and file name of the program you selected in the browser dialog.

Clicking the **[Open]** button after selecting a file will initiate the downloading. During the download HDI will report progress on the status bar.

# 4. Looking at Your Program

This section describes how to look at your program as source code and assembly language mnemonics. HDI's facilities for dealing with code and symbol information are explained and you will be shown how to look at text files in the user interface.

**Note**    After a break occurs HDI displays the location of the program counter (PC). In some cases, for example if a sysrof based project is moved from its original path, then the source files may not be automatically found. In this case HDI will try a list of paths previously used for this session. If still unable to locate the source file, then you open the source file in the [View->Source] menu dialog box. It allows you to manually locate the file - this path will then be added to the internal source path list for future reference.

## *4.1 Viewing the Code*

### 4.1.1 Viewing Source Code

To look at your program's source, choose the **[View->Source...]** menu option; use the **Ctrl+K** accelerator; or click on the Source Window toolbar button [[📄]].

Select your source file and click **[Open]**, HDI opens a Source window:

| | Tutorial.c | | | | _ □ × |
|---|---|---|---|---|---|
| Header bar | Line | Address | BP | Label | Source |
| | 28 | 00001012 | | _main | void main(void) |
| | 29 | | | | { |
| | 30 | | | | |
| | 31 | 00001018 | | | if(MDCR.BIT.MDS!=0x6 |
| | 32 | | | | /* printf("Sele |
| Breakpoint | 33 | 00001024 ● | | | return; |
| | 34 | | | | } |
| Source code | 35 | 00001026 | | | if(SYSCR.BYTE!=0x01) |
| | 36 | 00001030 | | | SYSCR.BYTE=0x1; |
| | 37 | | | | |
| Address field | 38 | 00001038 | | | BCRL.BIT.EAE = 0; |
| | 39 | | | | |
| | 40 | 00001040 | | | STOP_MODE(); |
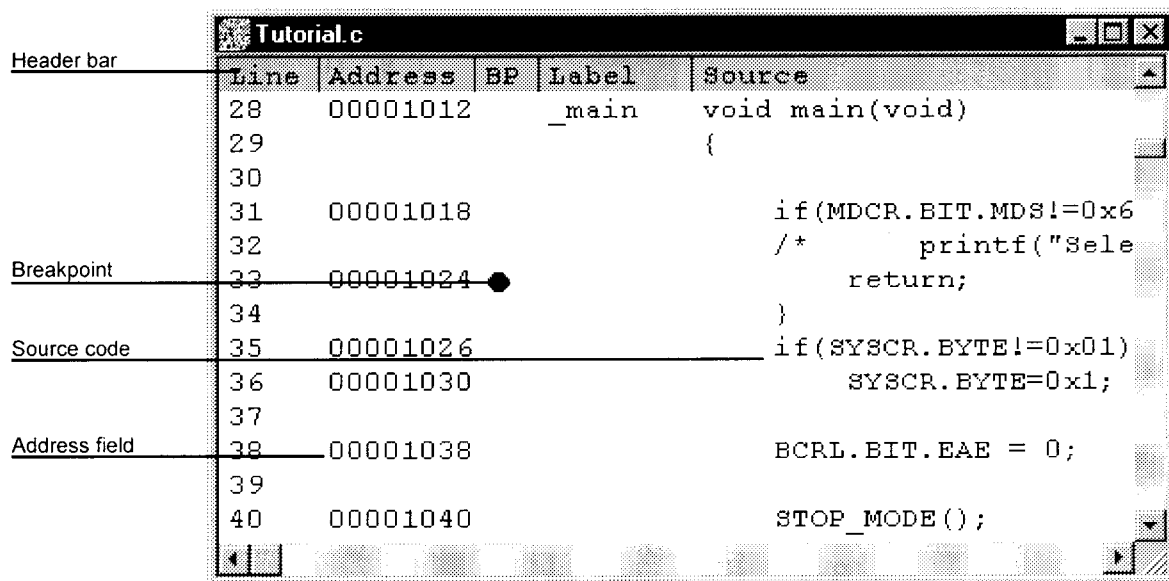
**Figure 4.1    Source Window**
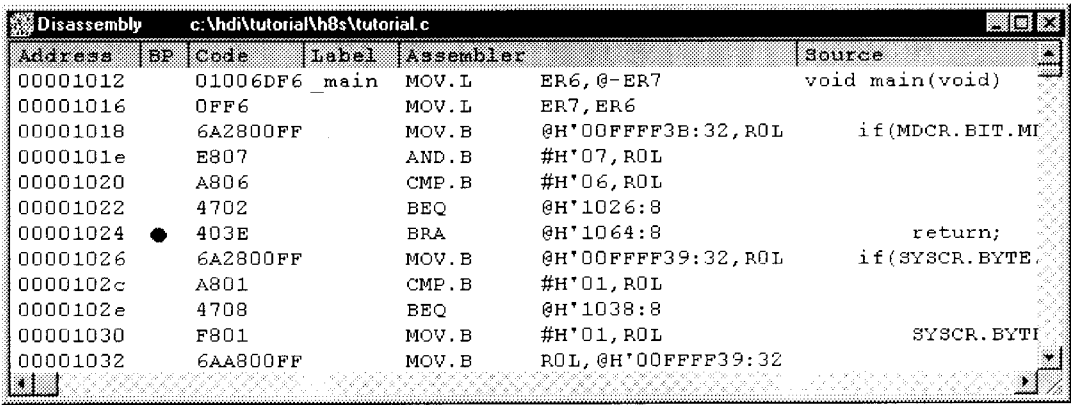
The Source window is divided into two areas; the header bar area and the main window area, and split vertically into five columns; Line, Address, BP(Breakpoint), Label, and Source. The respective width of each column can be adjusted by dragging the dividing line between each column title in the header bar. The cursor will change to +� and a vertical line will be displayed where the dividing line of the columns will be. Release the mouse button when you are satisfied with the column width and the display will be updated with the new column width.

### 4.1.2 Viewing Assembly-Language Code

If you have a source file open, right-click to open the popup menu and select Go to Disassembly to open a Disassembly window at the same address as the current Source window.

If you do not have a source file, but wish to view code at assembly-language level, either choose the [View->Disassembly...] menu option; use the Ctrl+D accelerator; or click on the Disassembly Window toolbar button [▓]. This will open a Set Address dialog in which you can address to start disassembling.

The Disassembly window shows Address, BP(Breakpoint), Code - showing the machine code values, Label and Assembler - showing the disassembled mnemonics (with labels when available). Additionally the final column contains any source line starting at that address, thus providing mixed mode display.

```
Disassembly    c:\hdi\tutorial\h8s\tutorial.c                          _□×
Address   BP  Code        Label   Assembler                    Source
00001012      01006DF6 _main   MOV.L    ER6,@-ER7          void main(void)
00001016      0FF6             MOV.L    ER7,ER6
00001018      6A2800FF         MOV.B    @H'00FFFF3B:32,R0L   if(MDCR.BIT.MI
0000101e      E807             AND.B    #H'07,R0L
00001020      A806             CMP.B    #H'06,R0L
00001022      4702             BEQ      @H'1026:8
00001024  ●   403E             BRA      @H'1064:8                  return;
00001026      6A2800FF         MOV.B    @H'00FFFF39:32,R0L   if(SYSCR.BYTE
0000102c      A801             CMP.B    #H'01,R0L
0000102e      4708             BEQ      @H'1038:8
00001030      F801             MOV.B    #H'01,R0L                SYSCR.BYT
00001032      6AA800FF         MOV.B    R0L,@H'00FFFF39:32
```

**Figure 4.2    Disassembly Window**

### 4.1.3 Modifying Assembly-Language Code

You can modify the assembly-language code by double-clicking on the instruction that you wish to change. The Assembler dialog box will open:
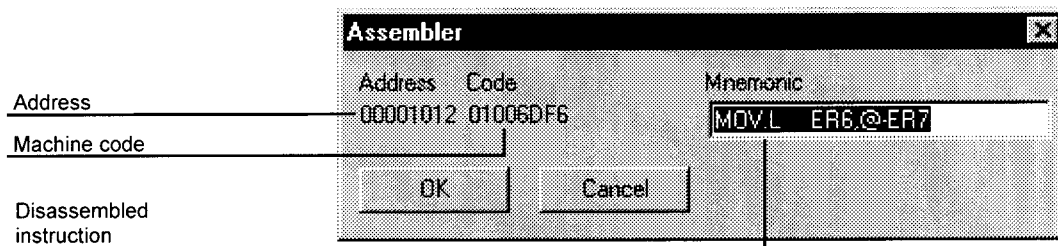


**Figure 4.3     Assembler Dialog Box**

The address, machine code and disassembled instruction are displayed. Type the new instruction or edit the old instruction in the Mnemonic field. Clicking **[OK]** or pressing **ENTER** will assemble the instruction into memory and move on to the next instruction. Clicking **[Cancel]** or pressing **ESC** will close the dialog box.

**Note**   The assembly-language display is disassembled from the actual machine code in the debugging platform's memory. If the memory contents are changed the display will show the corresponding new assembly-language code, but will not match the text shown in the source display.

## 4.2 Looking at Labels
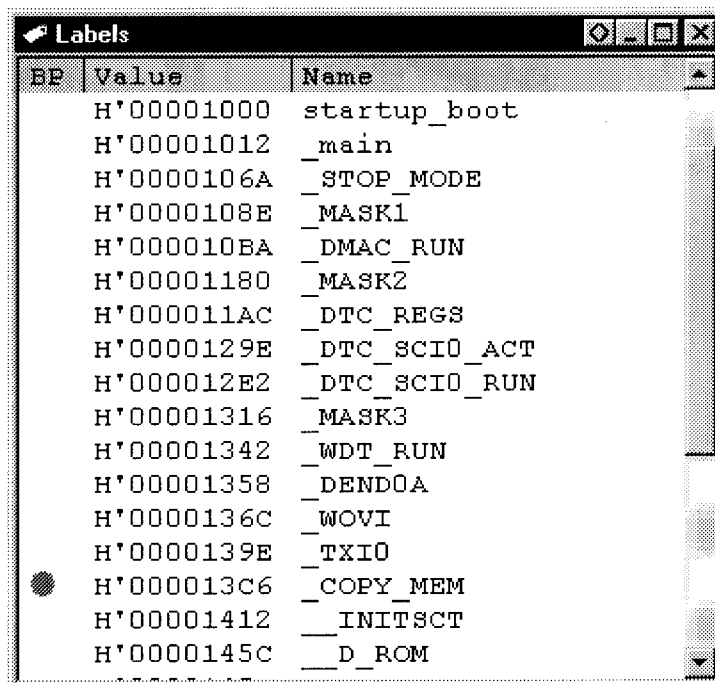
In addition to the debugging information that HDI uses to link your program's source code to the actual code in memory, the *debug object file* also contains symbolic information. This is a table of text names that represent an address in the program and is referred to as labels in HDI. You will see symbols in the Label field on the line of the corresponding address, and in the Assembler field as part of an instruction's operand.

**Note**   An instruction's operand is replaced with a label name if the operand and label value match. If two or more labels have the same value, then the label that comes first alphabetically will be displayed.

**Note**   Wherever you can enter an address or value in an HDI edit control you can use a label instead.

### 4.2.1 Listing Labels

To see a list of all the labels defined in the current session open the Labels window by choosing the [View->Labels] menu option.

```
Labels                                    ◇ _ □ ✕
BP │Value              │Name                        ▲
     H'00001000  startup_boot
     H'00001012  _main
     H'0000106A  _STOP_MODE
     H'0000108E  _MASK1
     H'000010BA  _DMAC_RUN
     H'00001180  _MASK2
     H'000011AC  _DTC_REGS
     H'0000129E  _DTC_SCI0_ACT
     H'000012E2  _DTC_SCI0_RUN
     H'00001316  _MASK3
     H'00001342  _WDT_RUN
     H'00001358  _DENDOA
     H'0000136C  _WOVI
     H'0000139E  _TXI0
  ●  H'000013C6  _COPY_MEM
     H'00001412   _INITSCT
     H'0000145C   _D_ROM                            ▼
```

**Figure 4.4    Labels Window**

You can view symbols sorted either alphabetically (by ASCII code) or by address value by clicking on the respective column heading.

You can quickly set a software break at the entry point of a function by double-clicking (or right-clicking and selecting Break on the BP popup menu) in the BP column.

### 4.2.2 Adding a Label from a Source or Disassembly Window

You can quickly add a label from a Source or Disassembly window, by double-clicking in the Label column at the address for which you want to assign the Label. The Label dialog box opens for you to enter the text.

```
Label                              ✕

│

    OK        Clear      Cancel
```
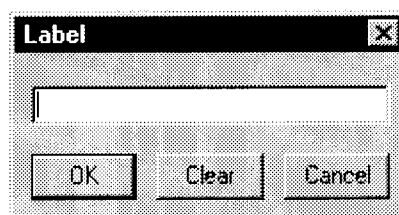
**Figure 4.5    Label Dialog Box**

Enter the label name text and click **[OK]**, the label is added to the label list with the address value contained in the Address column of the corresponding line, and the Source window display is updated to show the label. The **[Clear]** button can be used to remove the label.

This method can also be used for quickly modifying the text of existing labels. When you double-click on the label in the Label column, the text is copied into the edit box of the Label dialog box. You can then edit it and the modified version is saved in the label list. The Source window display is updated to show the new label.

**Note**   To use added or modified labels again in later sessions, save them in a file. For details, see *section 13.5.8, Save As....*

## 4.3 Looking at a Specific Address

When you are looking at your program in a Source window, you may want to look at another area of your program's code. Rather than scrolling through a lot of code in the program, you can go directly to a specific address. Double-click in the Address column, the Set Address dialog box opens:
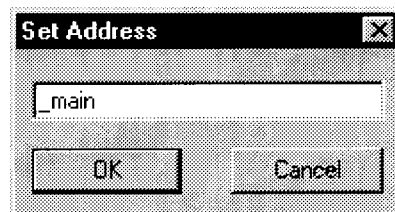


**Figure 4.6     Set Address Dialog Box**

Enter the address or symbol name in the edit box and either click on **[OK]** or press **ENTER**. If the code at that address is in the same source file, the Source window updates to show the code at the new address. When an overloaded function or a class name is entered, the Select Function dialog box opens for you to select a function. For details, refer to *section 10, Selecting Functions.*

If the new address is in a source file that is already being viewed in a Source window, that window is brought to the front and updated to show the code at the new address.

If the new address is in another source file, a new Source window opens to show the code at that address. By default the new window shows source if it is available. If no source is available for the new address, then a Disassembly window shows assembly-language code.

### 4.3.1  Looking at the Current Program Counter Address

Wherever you can enter an address or value into HDI, you can also enter an expression (see *section 2.2, Data Entry*). If you enter a register name prefixed by the "#" character, the contents of that register will be used as the value in the expression. Therefore if you open the Set Address dialog box and enter the expression "#PC", the Source or Disassembly window display will go to the current PC address. It also allows that you can display from an offset of the current PC by entering an expression with the PC register plus an offset, e.g., "#PC+0x100".

## *4.4 Finding Text*

You can search for a particular text string in the Source window using the find option. To do this, choose the [Find...] menu option from the popup menu, or use the Shift+F10 accelerator key.

The Find dialog box is displayed:
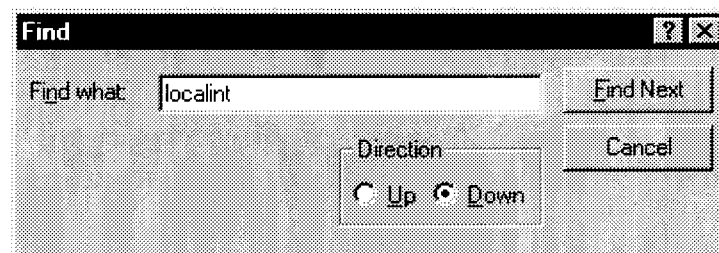


**Figure 4.7    Find Dialog Box**

Enter the text that you wish to find and click [Find Next] or press ENTER. The Source window will display the text (if found) highlighted. To find the next occurrence of the text, click [Find Next] or press ENTER again. To close the Find dialog box, click [Cancel] or press ESC.

# 5. Working with Memory

This section describes how to look at areas of memory in the CPU's address space. It will show you how to look at an area of memory in different formats, fill, move and test a block of memory, and save, load and verify an area of memory with a disk file.

## 5.1 Looking at an Area of Memory

To look at an area of memory, choose the [View->Memory...] menu option; using the Ctrl+M accelerator; or clicking the Memory Window toolbar button [[▣]] to open a Memory window. This will launch an Open Memory Window dialog box:
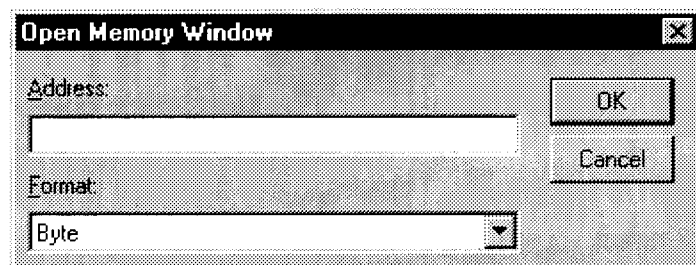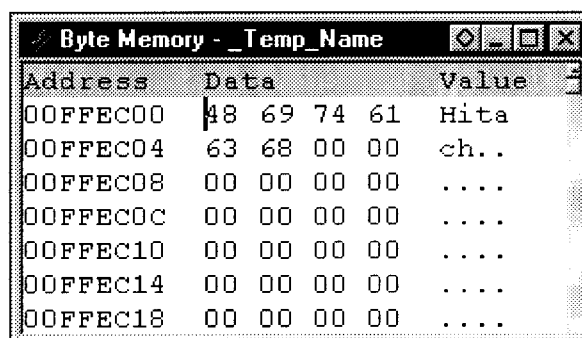


**Figure 5.1    Open Memory Window Dialog Box**

Type in the start address or equivalent symbol for the window display in the Address field and select the required display format from the Format list. Click [OK] or press ENTER, and the dialog box closes and a Memory window opens:



**Figure 5.2    Memory Window (Bytes)**

There are two display columns excluding the address display column:

1. Data - The data read from the debugging platform. Where supported it is read from physical memory at the displayed width. Editing the data is supported.

2. Value - Data displayed in an alternative format. Editing is not supported.

If you want to change the display format from the one you selected when you opened the window, this can be done from the popup menu.

### 5.1.1 Displaying Memory as ASCII

To display and edit memory as ASCII characters, choose the [ASCII] menu option from the popup menu and the display will be updated to show the area of memory as ASCII characters.

### 5.1.2 Displaying Memory as Bytes

To display and edit memory as bytes, choose the [Byte] menu option from the popup menu and the display will be updated to show the area of memory as individual bytes as shown in figure 5.2.

### 5.1.3 Displaying Memory as Words

To display and edit memory as words, choose the [Word] menu option from the popup menu and the display will be updated to show the area of memory as 16 bit words.

### 5.1.4 Displaying Memory as Long words

To display and edit memory as long words, choose the [Long] menu option from the popup menu and the display will be updated to show the area of memory as 32 bit long words.

### 5.1.5 Displaying Memory as Single-Precision Floating Point

To display and edit memory as single-precision floating-point data, choose the [Single float] menu option from the popup menu and the display will be updated to show the area of memory as single-precision floating-point data.

### 5.1.6 Displaying Memory as Double-Precision Floating Point

To display and edit memory as double-precision floating-point data, choose the [Double float] menu option from the popup menu and the display will be updated to show the area of memory as double-precision floating-point data.

### 5.1.7 Looking at a Different Area of Memory

If you want to change the area of memory that the Memory window is displaying you can use the scroll bars. To quickly look at a new address you can use the Set Address dialog box. This can be opened either be choosing the [Set Address] menu option from the popup menu or by double-clicking in the Address column.
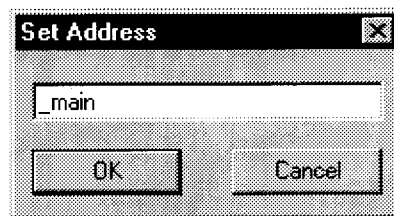


**Figure 5.3     Set Address Dialog Box**

Enter the new address value, and click [OK] or press ENTER. The dialog box closes and the Memory window display is updated with the data at the new address. When an overloaded function or a class name is entered, the Select Function dialog box opens for you to select a function. For details, refer to *section 10, Selecting Functions*.

## 5.2 Modifying Memory Contents

There are two ways that you can change the contents of memory at an address:

1. Quick edit method - allows you to enter values by typing directly into the window, but is limited to ASCII (when displaying ASCII format) or hexadecimal values only (when displaying all other formats).

2. Full edit method - uses a dialog box to enter values as floating point or evaluated expressions.

### 5.2.1 Quick Edit

The quick way to change the contents of memory is to select the digit that you wish to change, by clicking or dragging on it. You will see the selected digit is highlighted. Type the new value for the digit, it must be in the range 0-9, a-f (when displaying not ASCII format) or the new value for ASCII, it must be ASCII (when displaying ASCII format) . The new value is written into the digit and the cursor moves on to the next digit in memory.

### 5.2.2 Full Edit

The full way to change the contents of memory is accessed via the Edit dialog box. Move the cursor on the memory unit (depending on your Memory window display choice) that you wish to change. Either double-click on the memory unit, or press **ENTER**. The Edit dialog box opens:
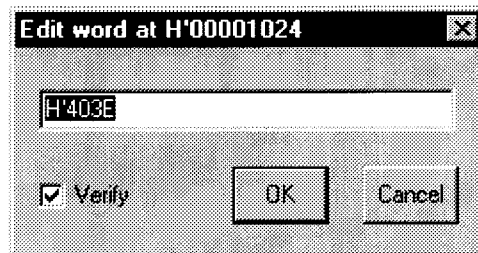


**Figure 5.4    Edit Dialog Box**

Like any other data entry field in HDI, you can enter a formatted number or C/C++ expression (see *section 2.2, Data Entry*). When you have entered the new number or expression, click the **[OK]** button or press **ENTER**, the dialog box closes and the new value is written into memory.

### 5.2.3 Selecting a Memory Range

If the memory address range is in the Memory window, you can select the range by clicking on the first memory unit (depending on your Memory window display choice) and dragging the mouse to the last unit. The selected range is highlighted.

## 5.3 Finding a Value in Memory

To find a value in memory you must open a Memory window, then choose the [Search] menu option from the popup menu. Alternatively, with a Memory window in focus, just press F3.

The Search Memory dialog box opens:
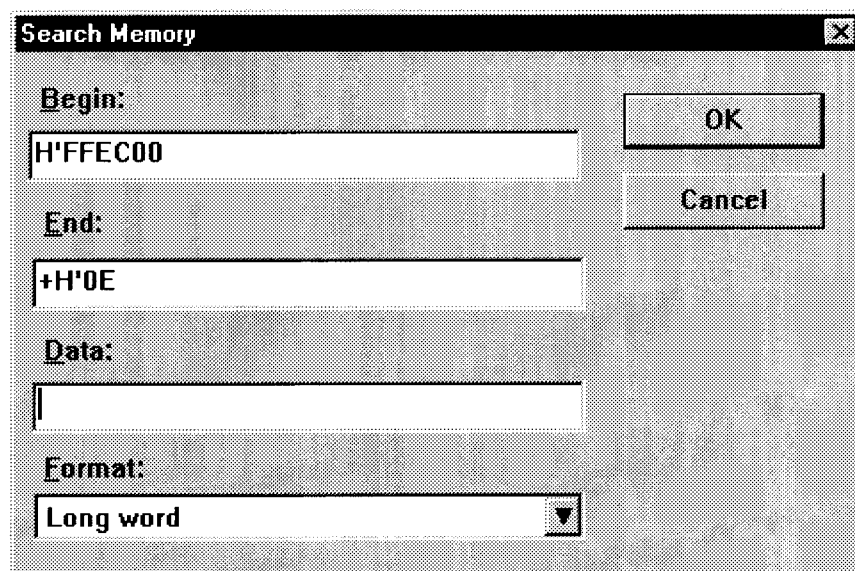


**Figure 5.5    Search Memory Dialog Box**

Enter the begin and end addresses of the range in which to search (if an area of memory was selected in the Memory window then the Begin and End address values will be filled in automatically) and the data value to search for. The end address can also be prefixed by a '+' which will use the entered value as a range.

Select the search format (defaults to data display format) and click [OK] or press ENTER. The dialog box closes and HDI searches the range for the specified data. If the data is found, it will be highlighted in the Memory window. If the data cannot be found, the caret position in the Memory window remains unchanged and a message informing you that the data could not be found is displayed on the message box.

## 5.4 Filling an Area of Memory with a Value

You can set the contents of a range of memory addresses to a value using the memory fill feature.

### 5.4.1  Filling a Range

To fill a range of memory with the same value, choose the [Fill...] menu option on a Memory window's popup menu, or [Memory->Fill...] menu option. The Fill Memory dialog box opens:
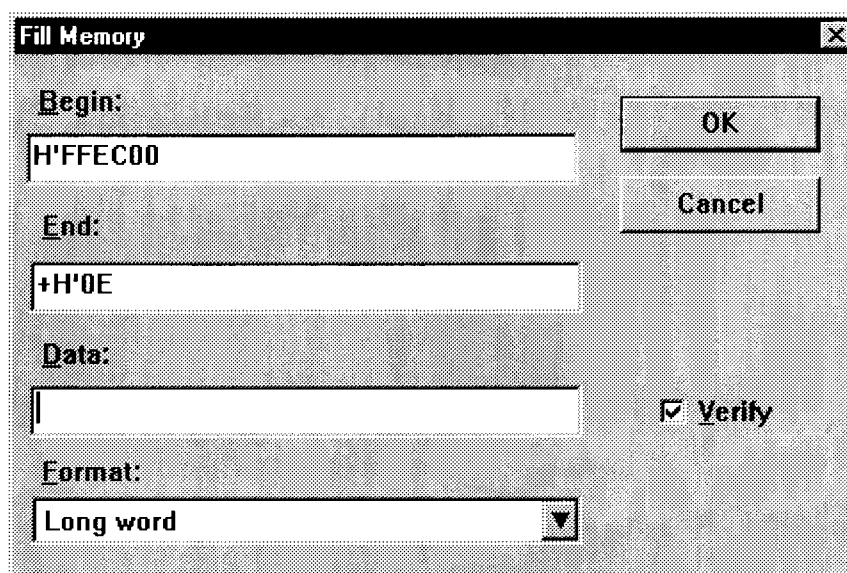


**Figure 5.6    Fill Memory Dialog Box**

If an address range has been selected in the Memory window, the specified begin and end address will be displayed. Select the format from the Format drop list and enter the data value in the Data field. Click the [OK] button or press ENTER, the dialog box closes and the new value are written into the memory range.

## 5.5 Copying an Area of Memory

You can copy an area of memory using the memory copy feature. Select a memory range (see *section 5.2.3, Selecting a Memory Range*), choose the [Copy...] menu option from the popup menu. The Copy Memory dialog box opens:
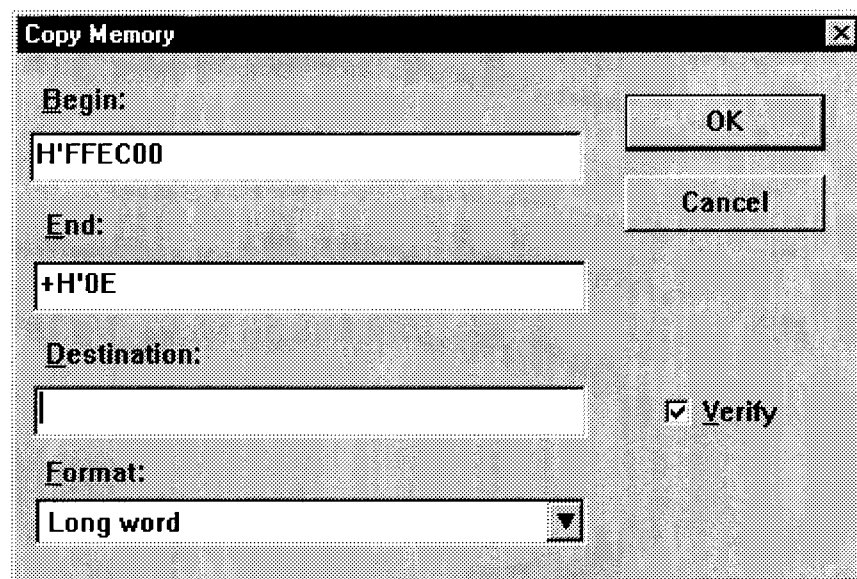
**Figure 5.7     Copy Memory Dialog Box**

The source begin and end address specified in the Memory window will be displayed in the Begin and End fields. Enter the destination start address in the Destination field and click the **[OK]** button or press **ENTER**, the dialog box closes and the memory block will be copied to the new address.

## 5.6 Testing an Area of Memory

**Note**   The exact test is target dependent. However, in all cases the current contents of the memory will be overwritten - YOUR PROGRAM OR DATA WILL BE ERASED.

You can test an area of memory in the address space using the memory test feature. Select a memory range (see *section 5.2.3, Selecting a Memory Range*), choose the **[Test]** menu option from the popup menu. The Test Memory dialog box opens:



**Figure 5.8     Test Memory Dialog Box**

The start address and end address specified in the Memory window will be displayed in the Begin and End fields. Click the **[OK]** button or press **ENTER**, the dialog box closes and HDI will perform a test on the memory range.

## 5.7 Saving an Area of Memory

You can save an area of memory in the address space to a disk file using the save memory feature. Open the Save Memory As dialog box by choosing the **[Memory->Save...]** menu option:
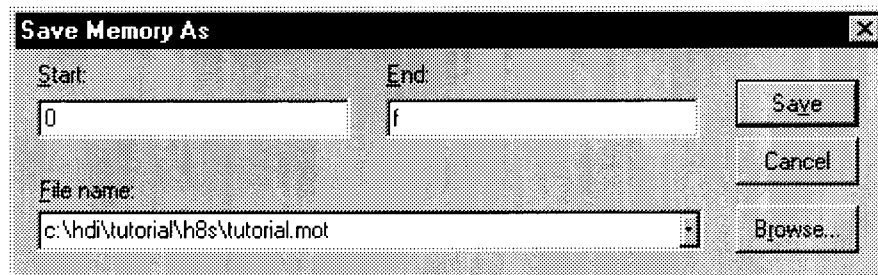


**Figure 5.9    Save Memory As Dialog Box**

Enter the start and end addresses of the memory block that you wish to save and a file name. The File name drop-list contains the previous four file names used for saving memory, or a standard Save As dialog can be launched by clicking the **[Browse...]** button. Click the **[Save]** button or press **ENTER**, the dialog box closes and the memory block will be saved to the disk as a Motorola S-Record format file. When the file save is complete a confirmation message box may be displayed (this can be switched off in the Confirmations tab on the HDI Options dialog).

## 5.8 Loading an Area of Memory

To load an S-Record file to an area of memory without removing the current debug information by using the load memory feature. Open the Load Memory dialog box by choosing the **[Memory->Load...]** menu option:
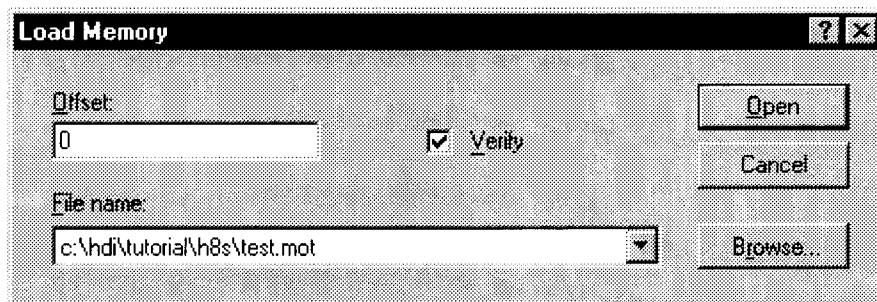
**Figure 5.10   Load Memory Dialog Box**

You can offset the loading address from the address specified in the
S-Record by entering a value (positive or negative) in the Offset field.
Click the [Open] button or press ENTER, the dialog box closes and
the data loads into memory. When the file load is complete a
confirmation message box may be displayed (this can be switched off
in the Confirmations tab on the HDI Options dialog).

## 5.9 Verifying an Area of Memory

You can compare an area of memory against a previously saved block
of memory using the memory verify feature. Open the Verify S-
Record File with Memory dialog box by choosing the [Memory-
>Verify...] menu option:
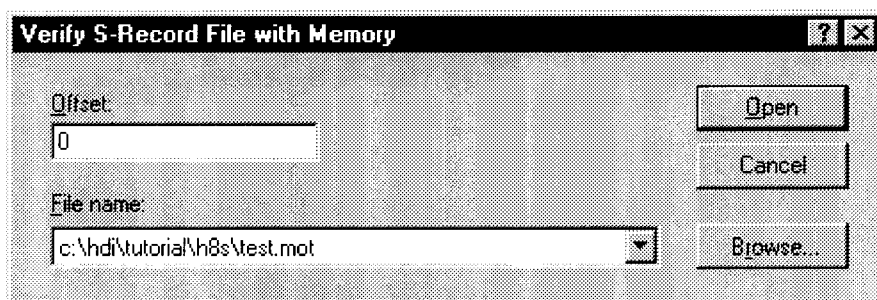


**Figure 5.11   Verify S-Record File with Memory Dialog Box**

You can offset the verification address from the address specified in
the S-Record by entering a value (positive or negative) in the Offset
field. Click the [Open] button or press ENTER, the dialog box closes
and the file are verified. When the file verification is complete a
confirmation message box may be displayed (this can be switched off
in the Confirmations tab on the HDI Options dialog).

# 6. Executing Your Program

This section describes how you can execute your program's code. You will learn how to do this by either running your program continuously or stepping single or multiple instructions at a time.

## 6.1 Running from Reset

To reset your user system and run your program from the *Reset Vector* address choose the [**Run->Reset Go**] menu option, or click the Reset Go toolbar button [▣].

The program will run until it hits a breakpoint or a break condition is met. You can stop the program manually at any time by choosing the [**Run->Halt**] menu option, or by clicking the Halt toolbar button [▣].

**Note**     The program will start running from whatever address is stored in the Reset Vector location. Therefore it is important to make sure that this location contains the address of your startup code.

## 6.2 Continuing Run

When your program is stopped and the debugger is in *break mode*, the HDI will highlight the line in the Source and Disassembly windows that correspond to the CPU's current Program Counter (PC) address value. This will be the next instruction to be executed if you perform a step or continue running.
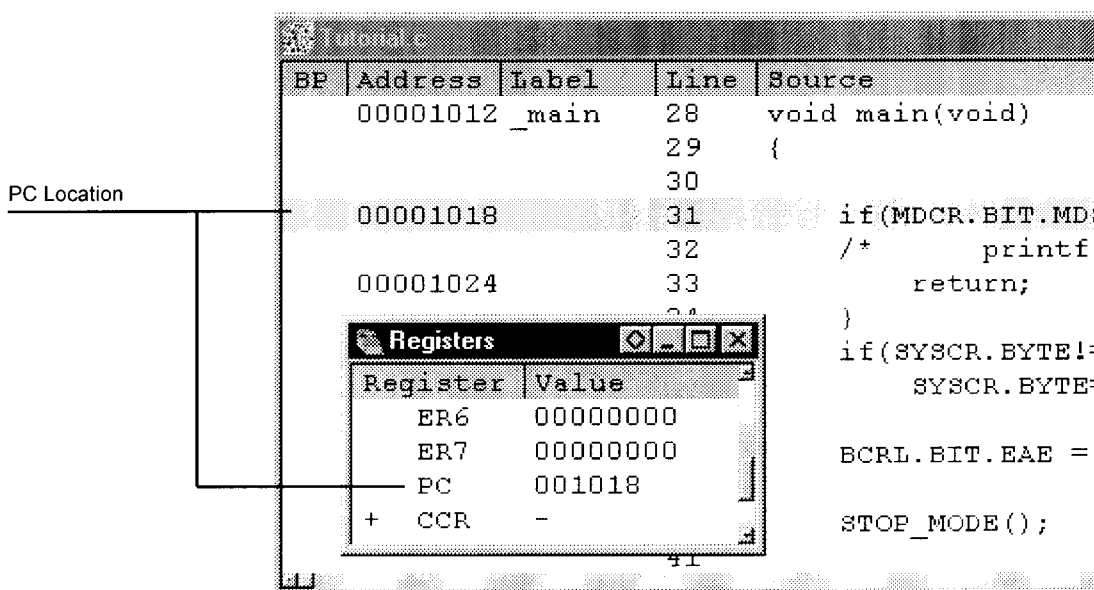


**Figure 6.1      Highlighted Line Corresponding to PC Address**

To continue running from the current PC address click the Go toolbar button [[▦]], or choose the **[Run->Go]** menu option.

## 6.3 Running to the Cursor

The function and it by which only a part of the program is executed provides the Go To Cursor feature to execute to a specific address.

**➲ Using Go To Cursor**

1. Make sure that a Source or Disassembly window is open showing the address at which you wish to stop.

2. Position the text cursor on the address at which you wish to stop by either clicking in the Address field or using the cursor keys.

3. Choose the **[Go To Cursor]** menu option from the popup menu.

The debugging platform will run your program from the current PC value until it reaches the address indicated by the cursor's position.

**Notes 1.** If your program never executes the code at this address, the program will not stop. If this happens, program execution can be stopped by pressing **Esc**, choosing the **[Run->Halt]** menu option, or clicking on the 'Halt' toolbar button [[▦]].

**2.** The Go To Cursor feature requires a temporary breakpoint - if you have already used all those available then the feature will not work, and the menu option will be disabled.

## 6.4 Running to Several Points

When you want to perform something like the Go To Cursor operation but the destination is outside the Source window, or want to stop at several addresses, you can use HDI's temporary breakpoint feature (see *section 7.5, Temporary Breakpoints*).

## 6.5 Single Step

When you are debugging your code it is very useful to be able to step a single line or instruction at a time and examine the effect of that instruction on the system. In the Source window, then a step operation will step a single source line. In the Disassembly window, a step operation will step a single assembly-language instruction. If the instruction calls another function or subroutine, you have the option to either step into or step over the function. If the instruction does not perform a call, then either option will cause the debugger to execute the instruction and stop at the next instruction.

### 6.5.1 Stepping Into a Function

If you choose to step into the function the debugger will execute the call and stop at the first line or instruction of the function. To step into the function either click the Step In toolbar button [[🔲]], or choose the **[Run->Step In]** menu option.

### 6.5.2 Stepping Over a Function Call

If you choose to step over the function the debugger will execute the call and all of the code in the function (and any function calls that that function may make) and stop at the next line or instruction of the calling function. To step over the function either click the Step Over toolbar button [[🔲]], or choose the **[Run->Step Over]** menu option.

## 6.6 Stepping Out of a Function

During debugging, there are occasions when you may have entered a function, finished stepping through the instructions that you want to examine and would like to return to the calling function without tediously stepping through all the remaining code in the function. Or alternatively (and perhaps more usefully) you may have stepped into a function by accident, when you meant to step over it and so want to return to the calling function without stepping all the way through the current function. You can do this with the Step Out feature.

To step out of the current function either click the Step Out toolbar button [[🔲]], or choose the **[Run->Step Out]** menu option.

## 6.7 Multiple Steps

Sometimes you may find it useful to step several instructions at a time. You can do this by using the Step Program dialog box. The dialog box also provides an automated step with a selectable delay between steps. Open it by choosing the [**Run-> Step...**] menu option.

The Step Program dialog box is displayed:
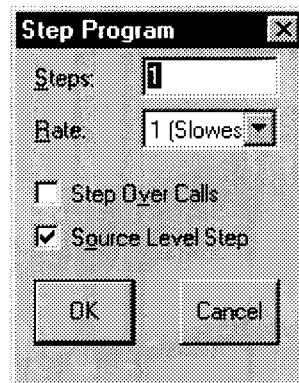
**Figure 6.2   Step Program Dialog Box**

Enter the number of steps in the Steps field , select whether you want to step over function calls by the Step Over Calls check box, and select whether to make one line of the source program correspond to one step by the Source Level Step check box. If you are using the feature for automated stepping, select the step rate from the list in the Rate field. Click [**OK**] or press **ENTER** to start stepping.

# 7. Stopping Your Program

This section describes how you can halt execution of your application's code. This section describes how to do this directly by using the halt command and by setting breakpoints at specific locations in your code.

## 7.1 Halting Execution

When your program is running, the Halt toolbar button is enabled [[🛑]] (a red STOP sign), and when the program has stopped it is disabled [[🛑]] (the STOP sign is grayed out). To stop the program click on the Halt toolbar button, push the ESC key, or choose the [Run->Halt] menu option.

Your program's execution is halted, with the message "Break = Stop" displayed on the status bar. HDI will then update any open windows.

The last break cause can also be viewed in the Platform pane of the System Status window.

## 7.2 Standard Breakpoints (PC Breakpoints)

When you are trying to debug your program you will want to be able to stop the program running when it reaches a specific point or points in your code. You can do this by setting a PC breakpoint on the line or instruction at which to want the execution to stop. The following instructions will show you how to quickly set and clear simple PC breakpoints. More complex breakpoint operation can be done via the Breakpoints window, which is discussed later.

➲ To set a program (PC) breakpoint

1. Make sure that the Source window is open at the place you want to set a program (PC) breakpoint.

2. Double-click in the BP column, or press F9, at the line showing the address at which you want the program to stop.

3. You will see a circle and the word 'Break' appear in the column to indicate that a program (PC) breakpoint has been set.

```
Tutorial.c                                          _ □ ×
Line   Address  BP  Label   Source                      ▲
28     00001012      _main   void main(void)
29                           {
30
31     00001018              if(MDCR.BIT.MDS!=0x6
32                           /*      printf("Sele
33     00001024  ●           return;
34                           }
35     00001026              if(SYSCR.BYTE!=0x01)
36     00001030                  SYSCR.BYTE=0x1;
37
38     00001038              BCRL.BIT.EAE = 0;
39                  .
40     00001040              STOP_MODE();           ▼
◀ ▮                                             ▶
```
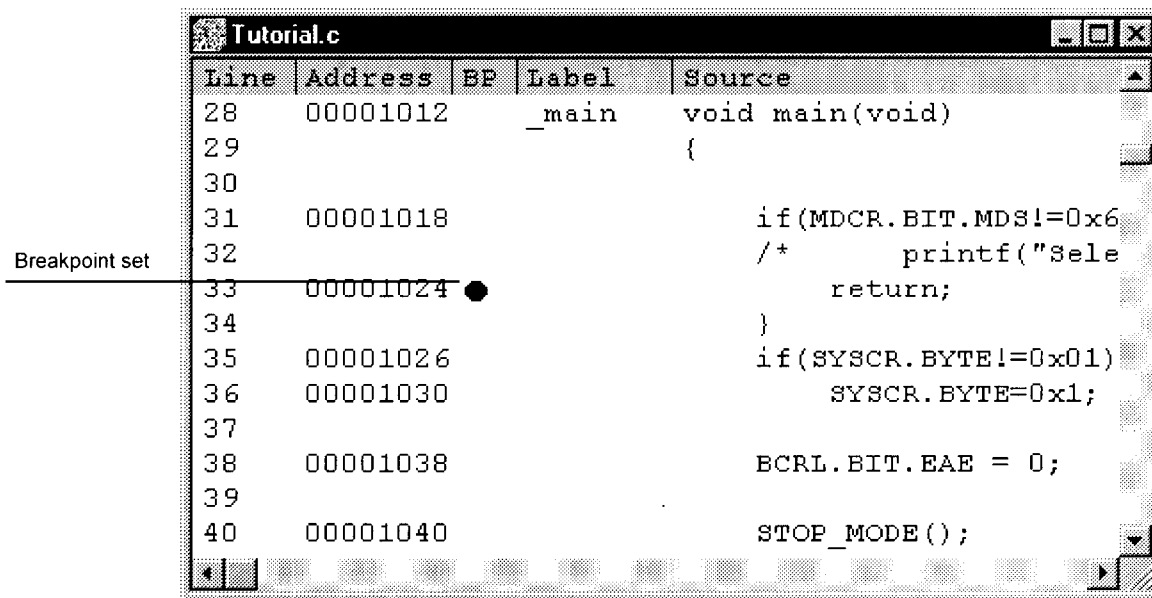
Breakpoint set

**Figure 7.1    Setting a Program Breakpoint**

Now when you run your program and it reaches the address at which you set the program (PC) breakpoint, execution halts with the message "Break = PC Breakpoint" displayed on the status bar, and the Source window display is updated with the program (PC) breakpoint line highlighted.

**Note**   The line or instruction at which you set a program (PC) breakpoint is not actually executed; the program stops just before it is about to execute it. If you choose to Go or Step after stopping at the program (PC) breakpoint, then the highlighted line will be the next instruction to be executed.

## 7.2.1 Cycling Through Standard Breakpoints

By default the standard breakpoint will support the PC breakpoint type. However, depending on the selected platform, more than one type of standard breakpoint may be provided. It is possible to cycle through these by either double-clicking in the BP column of the line at which the program (PC) breakpoint is set or placing the text cursor on the line and using the **F9** key. The display will cycle through the available standard breakpoint types - a color-coded circle and a descriptive word will be shown in the BP column.

### 7.2.2  Clearing Standard Breakpoints

Right-click in the BP column to display a special Break popup menu that lists all the standard breakpoint types for the current platform. The currently selected break type will be shown with a check mark to clear the breakpoint simply click the 'None' option.

Alternatively when all the standard breakpoints have been cycled through, then the breakpoint is cleared.

## 7.3  The Breakpoints Window

The Breakpoints window allows you to access complex breakpoints (if your debugging platform supports them) and gives you more control over setting/clearing and enabling/disabling breakpoints. To open the Breakpoints window choose the [View->Breakpoints] menu option or click the Breakpoint Window toolbar button [⊞], if it visible.

A Breakpoints window opens.
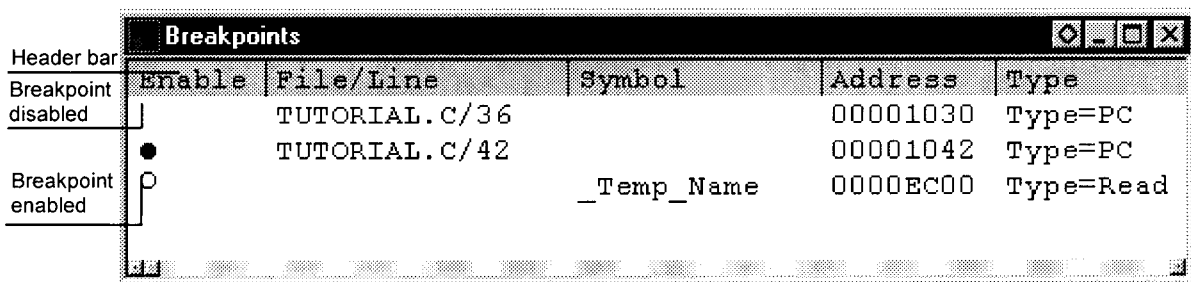


Figure 7.2    Breakpoints Window

The window displays a list of the breakpoints set in the system. The breakpoint list is divided horizontally into five columns; Enable, File/Line, Symbol, Address, and Type. The respective widths of each of the columns can be adjusted by clicking and dragging on the dividing line between each column title in the header bar. The cursor will change to ↔ and a vertical line will be displayed at the dividing line of the columns. Release the mouse button when you are satisfied with the column width and the display will be updated with the new column width.

### 7.3.1  Adding a Breakpoint

You can add a new breakpoint in the Breakpoints window by choosing the [Add...] menu option from the popup menu.

The Breakpoint/Event Properties dialog box will open in which you can enter the type and parameters of the new breakpoint.

**Note**   The Breakpoint/Event Properties dialog box is specific to the debugging platform you have selected. Its appearance and operation depend on the breakpoint features available in the debugging platform. For details on debugging platform specific breakpoints, see the separate *Debugging Platform User's Manual*.

### 7.3.2  Modifying a Breakpoint

To edit an existing breakpoint in the Breakpoints window, select the breakpoint in the list by double-clicking, or by clicking on the line corresponding to it and choose [**Edit...**] menu option from the popup menu.

The Breakpoint/Event Properties dialog box will open in which you can change the type and parameters of the selected breakpoint.

**Note**   Breakpoint/Event Properties dialog box is specific to the debugging platform you have selected. Its appearance and operation depend on the breakpoint features available in the debugging platform. For details on debugging platform specific breakpoints, see the separate *Debugging Platform User's Manual*.

### 7.3.3  Deleting a Breakpoint

To delete an existing breakpoint in the Breakpoints window, select the breakpoint in the list by clicking on the line corresponding to it and choose the [**Delete**] menu option from the popup menu.

The breakpoint is deleted and the window is updated.

### 7.3.4  Deleting All Breakpoints

To delete all of the breakpoints listed in the Breakpoints window choose the [**Delete All**] menu option from the popup menu.

All breakpoints are deleted and the window is cleared.

## 7.4 Disabling Breakpoints

During the course of a debugging session you may find that you tend to focus on particular areas of code for a period of time and then look at other areas, but want to return to the previous ones afterwards. When concentrating on these areas you will want to set breakpoints to stop your program execution at useful points. If you have set these breakpoints and wish to move on to another area of investigation, but know that you will want to return to the current area later, it is frustrating to have to delete all the breakpoints you have set only to have to set them all again when you return. Fortunately, HDI eases this problem by allowing you to disable breakpoints, while still leaving them in the breakpoint list.

### 7.4.1 Disabling a Breakpoint

To disable an individual breakpoint, select the breakpoint in the list by clicking on the line corresponding to it and choose the [Disable] menu option from the popup menu.

Alternatively, double-click in the Enable column of the breakpoint you need to disable.

The symbol in the Enable column is cleared to show that the breakpoint is no longer enabled.

### 7.4.2 Enabling a Breakpoint

When you want to re-enable a breakpoint in the Breakpoints window list, select the breakpoint in the list by clicking on the line corresponding to it and choose the [Enable] menu option from the popup menu.

Alternatively, double-click in the Enable column of the breakpoint you need to enable.

The symbol in the Enable column is set to show that the breakpoint is again enabled.

## 7.5 Temporary Breakpoints

There are times when you may want to start running your program and want it to stop if it hits one or more addresses, but do not want to set permanent breakpoints at these addresses. For example you may want to perform something like the Go To Cursor operation, but the destination may be outside the Source window or you may want to stop at several addresses. To do this you can use HDI's temporary breakpoint feature to run as it supports up to ten temporary breakpoints that are cleared when you break. Temporary breakpoints are set in the Run Program dialog box, which is opened by choosing the [Run-> Run...] menu option.

The Run Program dialog box opens:
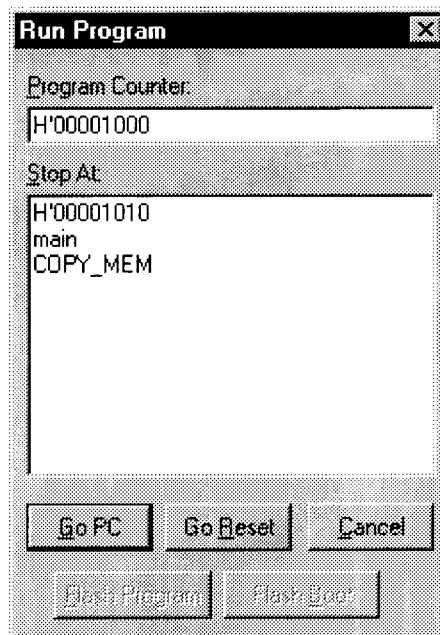


**Figure 7.3    Run Program Dialog Box**

Enter the symbols or address values for the points at which you want the program to stop (up to ten points) in the Stop At field. When an overloaded function or a class name is entered, the Select Function dialog box opens for you to select a function. For details, refer to *section 10, Selecting Functions*.

Click the [Go PC] button to start running from the current Program Counter address, as displayed in the Program Counter field. Click the [Go Reset] button to reset the CPU and start running from the reset vector address.

When the program halts the temporary breakpoints that you specified are cleared from the current breakpoint list. However, when the dialog is opened again, the list is retained in the Stop At field and will be set again if you click the [Go PC] or [Go Reset] buttons.

## 7.6 Hardware Breakpoints(Event)

**Note**   Hardware breakpoints are specific to the debugging platform you have selected. Their operation depends on the breakpoint features available in the debugging platform. For details on debugging platform specific breakpoints, see the separate *Debugging Platform User's Manual*.

# 8. Looking at Variables

This section describes how to look at the variables and data objects that your program uses. It shows you how to view variables, set up watch items and look at the contents of the CPU's general, FPU, DSP and on-chip peripheral registers.

## 8.1 Tooltip Watch

The quickest way to look at a variable in your program is to use the Tooltip Watch feature.

➲ To use Tooltip Watch:

1. Open the Source window showing the variable that you want to examine.

2. Rest the mouse cursor over the variable name that you want to examine - a tooltip will appear near the variable containing basic watch information for that variable.

```
 _MEM 221    void COPY_MEM(void)
      222    {
      223        unsigned short u;
      224        for( u=0; u < sizeof(NAME); u++ )
      225            *(Temp2_Name+u) = *(NAME+u);
      226
      227    }                        u = H'7E21
```
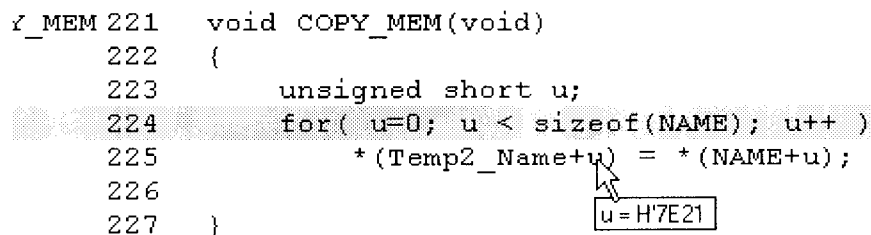
**Figure 8.1    Tooltip Watch**

## 8.2 Instant Watch

To look at the variable in more detail, use the Instant Watch feature.

➲ To use Instant Watch:

1. Open the Source window showing the variable that you want to examine.

2. Click on the variable. You should see a cursor on the variable.

3. Choose the [Instant Watch] menu option from the popup menu.

The Instant Watch dialog box opens:
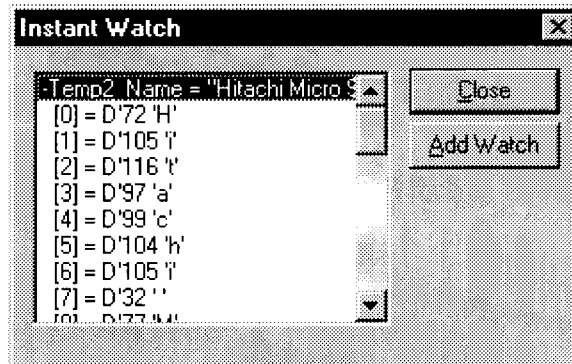


**Figure 8.2     Instant Watch Dialog Box**

You can add this variable to the list of watch items in the Watch window by clicking on the [**Add Watch**] button.

## 8.3 Using Watch Items

When you are debugging your program you may find it useful to be able to look at variables of interest and see their values at different times during the program. HDI allows you to open Watch windows, which contain a list of variables and their values. To open a Watch window choose the [**View->Watch**] menu option; or click on the Watch Window toolbar button [▣] if it is visible. A Watch window opens. Initially the contents of the window will be blank.

### 8.3.1 Adding a Watch

There are two ways to add Watch items to the Watch window; the quick method accessed from the Source window, and the full method using the Add Watch dialog box in the Watch window.

**Quick Method**

The quickest way to add a variable to the Watch window is to use the Add Watch feature.

➲ To use Add Watch from a Source Window:

1. Open the Source window showing the variable that you want to examine.

2. Click on the variable. You should see a cursor on the variable.

3. Choose the **[Add Watch]** menu option from the popup menu.

The variable is added as a watch item and the Watch window updates.

## Full Method

The full method uses a dialog that allows you to enter more complex watch expressions, for example arrays, structures or pointers.

➲ To use Add Watch from a Watch Window:

1. Open the Watch window.

2. Choose the **[Add Watch]** menu option from the popup menu.

The Add Watch dialog box opens:
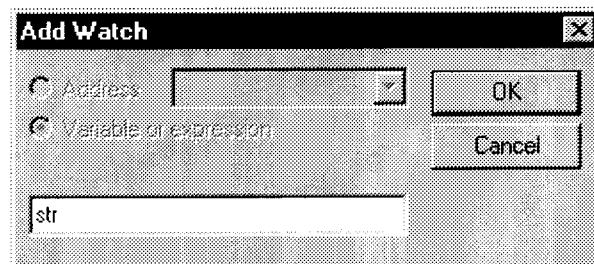
**Figure 8.3    Add Watch Dialog Box**

Enter the name of the variable that you wish to watch and click **[OK]**. The variable is added to the Watch window.
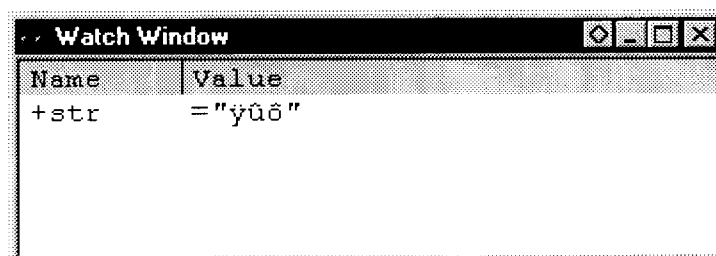
**Figure 8.4    Watch Window**

**Note**   If the variable that you have added is a local variable that is not currently in scope, HDI will add it to the Watch window but its value will be blank, or set to a question mark, "?".

### 8.3.2 Expanding a Watch

If a watch item is a pointer, array, or structure, then you will see a plus sign (+) expansion indicator to left of its name, this means that you can expand the watch item. To expand a watch item, double-click on it. The item expands to show the elements (in the case of structures and arrays) or data value (in the case of pointers) indented by one tab chracter, and the plus sign changes to a minus sign (-). If the elements of the watch item also contain pointers, structures, or arrays then they will also have expansion indicators next to them.
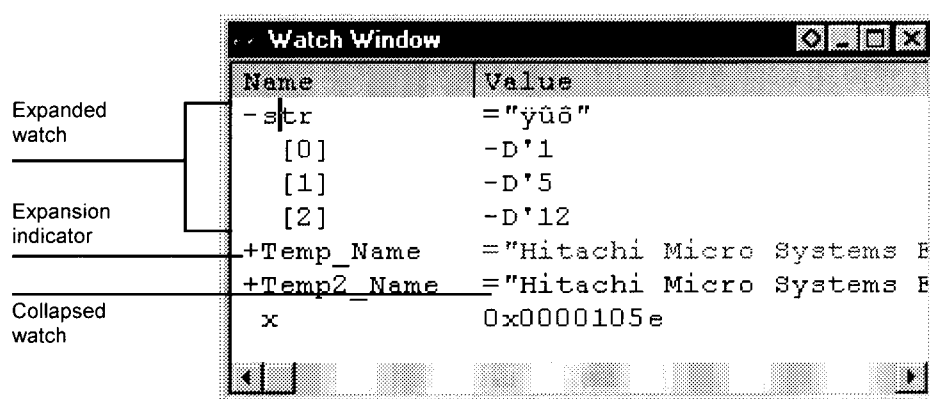


**Figure 8.5    Expanding a Watch**

To collapse an expanded watch item, double-click on the item again. The item's elements will collapse back to the single item and the minus sign changes back to a plus sign.

### 8.3.3 Modifying Radix for Watch Item Display

To change the radix of watch item, select the corresponding item by clicking it, and click the right mouse button on the item. Then a popup menu will be displayed. Choose the [Radix] menu option from the popup menu. Then choose the radix in which you wish the selected watch item to be displayed. The value will be updated immediately. (The [Setup->Radix] menu option from the main menu is irrelevant to this radix modification.)

### 8.3.4 Changing a Watch Item's Value

You may wish to change the value of a watch variable, e.g. for testing purposes or if the value is incorrect due to a bug in your program. To change a watch item's value use the Edit Value function.

➲ Editing a watch item's value:

1. Select the item to edit by clicking on it, you will see a flashing cursor on the item.

2. Choose the **[Edit Value]** menu option from the popup menu.

The Edit Value dialog box opens:
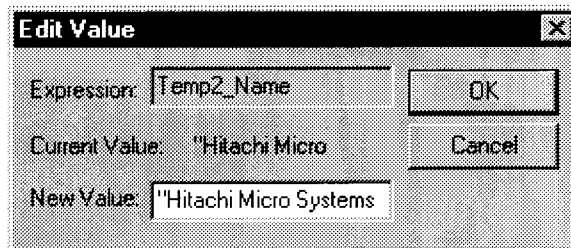


**Figure 8.6     Edit Value Dialog Box**

Enter the new value or expression in the New Value field and click **[OK]**. The Watch window is updated to show the new value.

### 8.3.5  Deleting a Watch

To delete a watch item, select it and choose the **[Delete]** menu option from the popup menu. The item is deleted and the Watch window updated.

**Note**    Watch items that you have set in the Watch window can be saved in a session file. See *section 11, Configuring the User Interface.*

## 8.4 Looking at Local Variables

To look at local variables, open the Locals window by choosing the **[View->Locals]** menu option.

The Locals window opens:
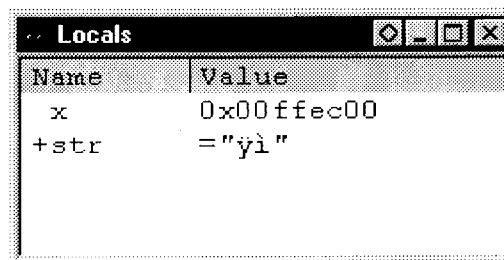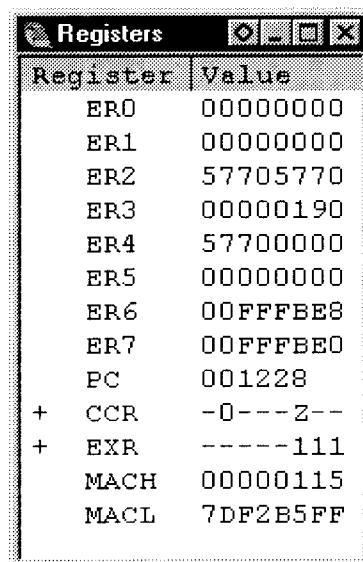


**Figure 8.7     Locals Window**

As you debug your program the Locals window will be updated, following a step or break from run, to show current local variables and their values. If a local variable is not initialized when defined, then the value in the Locals window will be undefined until a value is assigned to the local variable.

The local variable values and the radix for local variable display can be modified in the same manner as in the Watch window.

## 8.5 Looking at Registers

If you are debugging at assembly-language level, using the Source window in assembly language or mixed display, then you will probably find it useful to see the contents of the CPU's general, FPU and DSP registers. You can do this using the Registers window.

```
Registers
Register   Value
     ER0   00000000
     ER1   00000000
     ER2   57705770
     ER3   00000190
     ER4   57700000
     ER5   00000000
     ER6   00FFFBE8
     ER7   00FFFBE0
     PC    001228
+    CCR   -0---Z--
+    EXR   -----111
     MACH  00000115
     MACL  7DF2B5FF
```

**Figure 8.8    Registers Window**

To open a Registers window choose the [View->Registers] menu option or click the CPU Register Window toolbar button [[🔳]]. A Registers window opens showing all of the CPU's general, FPU and DSP registers and their values, displayed in hexadecimal.

### 8.5.1 Expanding a Bit Register

If a register is used to control or display status using flags at the bit level, then you will see a plus sign (+) expansion indicator to left of its name, this means that you can expand it. To do this, double-click on the plus sign to show the flags indented by one tab character, and the plus sign changes to a minus sign (-). If the flags have sub-groups, for example register masks, they will also have expansion indicators next to them.

```
Registers                     [◇][_][□][✕]
Register  Value
   R0      0000
   R1      0000
   R2      0000
   R3      0000
   R4      0000
   R5      0000
   R6      0000
   R7      0000
   PC      1004
-  CCR     I0---Z--
    I       1
    U       0
    H       0
    U       0
    N       0
    Z       1
    V       0
    C       0
```

Standard Register ———————————— → R4

Expansion indicator ———————————— → - CCR

Expanded bit register ————————————
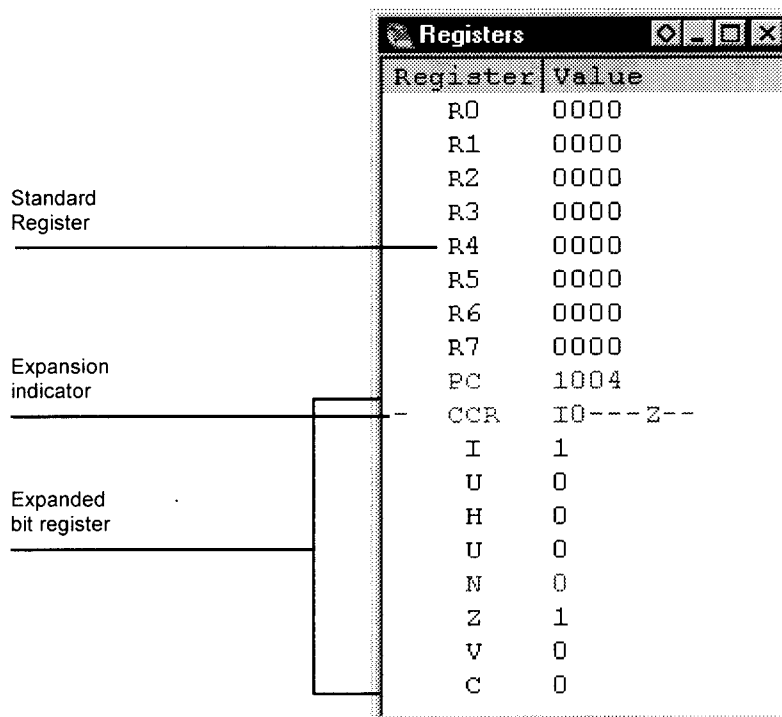
**Figure 8.9    Expanding a Flag Register**

To collapse an expanded flag register, double-click on the minus sign. The flags collapse back to the single item and the minus sign changes back to a plus sign.

### 8.5.2 Modifying Register Contents

There are two ways that you can change a register's contents. The quick edit method that allows you to enter values by typing directly into the window, but is limited to hexadecimal values only. The full edit method that requires you to enter values via a dialog box, but allows you to enter values in any base and use complex expressions.

## Quick Edit

The quick way to change a register's contents is to select the digit that you wish to change, by clicking or dragging on it. You will see the selected digit is highlighted. Type the new value for the digit; it must be in the range 0-9 or a-f. The new value is written into the digit and the cursor moves to the next digit in the register. When you enter a value into the least significant digit of the register, the cursor moves on to the most significant digit of the next register. If the digit of the register display indicates a bit e.g. in the CPU condition code register (CCR) then you can press **SPACE** to toggle the bit's value.

## Full Edit

The full way to change a register's contents is accessed via a Register dialog box. Open the Register dialog box in one of three ways:

1. Double-click the register you want to change.

2. Select the register you want to change, and press **ENTER**.

3. Select the register you want to change, and choose the [**Edit...**] menu option from the popup menu.
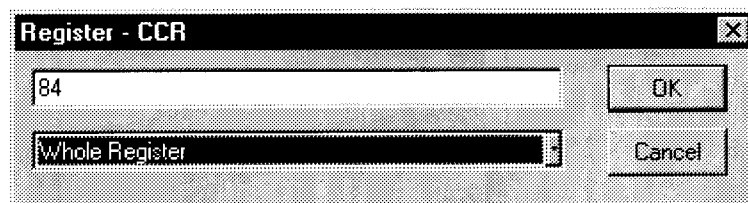


**Figure 8.10   Register Dialog Box**

As in any other data entry field in HDI, you can enter a formatted number or C/C++ expression (see *section 2.2, Data Entry*).

You can choose whether to modify the whole register contents(High Word, Low Word,etc), a masked area, floating or flag bits by selecting an option from the drop list box (the contents of this list depend on the CPU model and selected register).

When you have entered the new number or expression, click the [**OK**] button or press **ENTER**, the dialog box closes and the new value is written into the register.

### 8.5.3 Using Register Contents

It can be useful to be able to use the value contained in a CPU register when you are entering a value elsewhere in HDI, for example when displaying a specified address in the Source or Memory windows. You can do this by specifying the register name prefixed by the "#" character, e.g.: #R1, #PC, #R6L, or #ER3.

## 8.6 Looking at I/O

As well as a CPU and ROM/RAM, the microcomputer also contains on-chip peripheral modules. The exact number and type of peripheral modules differ between devices but typical modules are DMA controllers, serial Communications interfaces, A/D converters, integrated timer units, a bus state controller and a watchdog timer. These on-chip peripherals are programmed by accessing registers mapped to the micro-controller's address space.

Since the setting up and use of these on-chip peripheral registers is usually very important in an embedded micro-controller application, it is useful to be able to look clearly at the contents of these registers. The Memory window only allows you to look at data in memory as byte, word, long word, single-precision floating-point, double-precision floating-point, or ASCII values, so HDI also provides an I/O Registers window to ease inspection and setting up of these registers.

### 8.6.1 Opening an I/O Registers Window

To open an I/O Registers window select the [View->I/O Area] menu option or click the I/O Register Window toolbar button [[▣]]. The I/O register information is organized by modules that match the on-chip peripherals. When an I/O Registers window is first opened, only a list of module names is displayed.
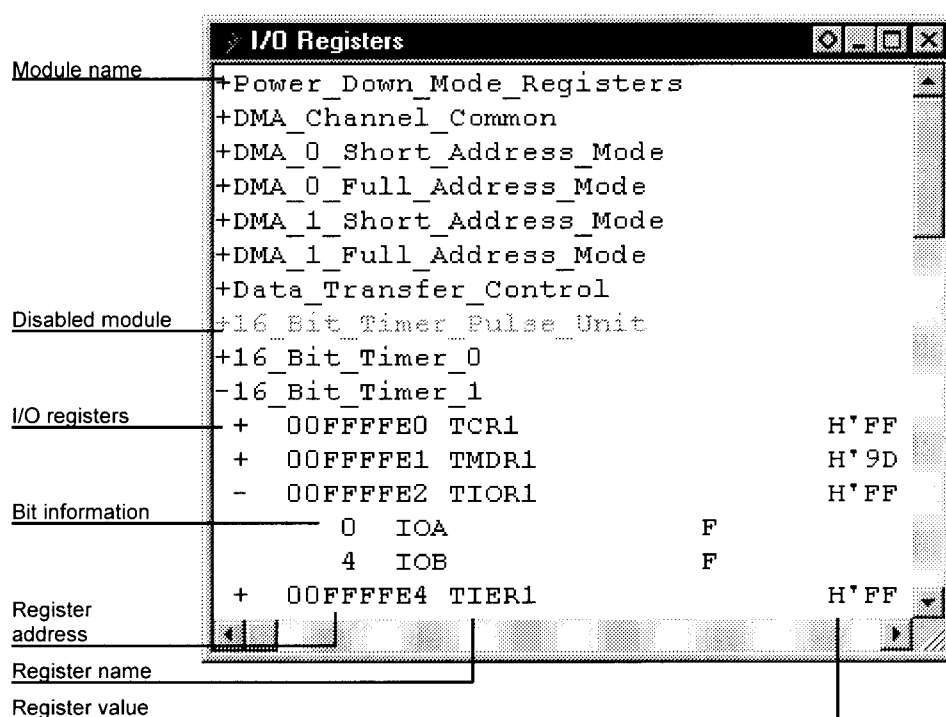
**Figure 8.11    I/O Registers Window**

## 8.6.2 Expanding an I/O Register Display

To display the names, addresses and values of the I/O registers, double-click on the module name or select the module name, by clicking on it or using the cursor keys, and press **ENTER**. The module display will expand to show the individual registers of that peripheral module and their names, addresses and values. Double-clicking (or pressing **ENTER**) again on the module name will close the I/O register display.

To display to bit level, expand the I/O register in a similar way.

The bits are color coded as follows:

| Black | Normal read/write |
|-------|-------------------|
| Red   | Value changed since last update |
| Grey  | Peripheral disabled (by peripheral control registers) |

### 8.6.3 Modifying I/O Register Contents

To edit the value in an I/O register you can type hex values directly into the view. To enter more complex expressions double-click or press **ENTER** on the register to open a dialog box to modify the register contents:
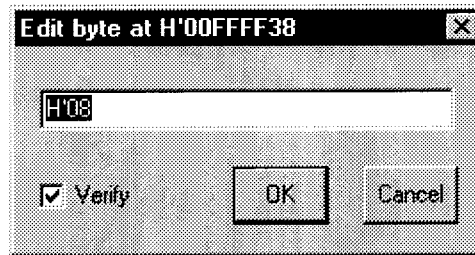


**Figure 8.12   Dialog Box for Modifying I/O Register Contents**

When you have entered the new number or expression, click the **[OK]** button or press **ENTER**; the dialog box closes and the new value is written into the register.

**Note**   If you are using an emulator based debugging platform, when it reads data from an I/O register this can sometimes affect the operation of your program. For example, reading a data register can cancel a pending interrupt. Data is only read from I/O modules that have been expanded in the I/O Registers window (so that the register values are displayed). Therefore, as long as I/O modules are collapsed when they no longer need to be displayed, this will not cause a problem. In order to check whether this is affecting your program try running it without the I/O Registers window. Also, note that having a Memory window (or Disassembly window) open on the I/O area can have the same effect.

# 9. Overlay Function

Programs making use of the overlay function can be debugged. This section explains the settings for using the overlay function.

## 9.1 Displaying Section Group

When the overlay function is used, that is, when several section groups are assigned to the same address range, the address ranges and section groups are displayed in the Overlay dialog box.

Open the Overlay dialog box by choosing the [Setup->Overlay] menu option.
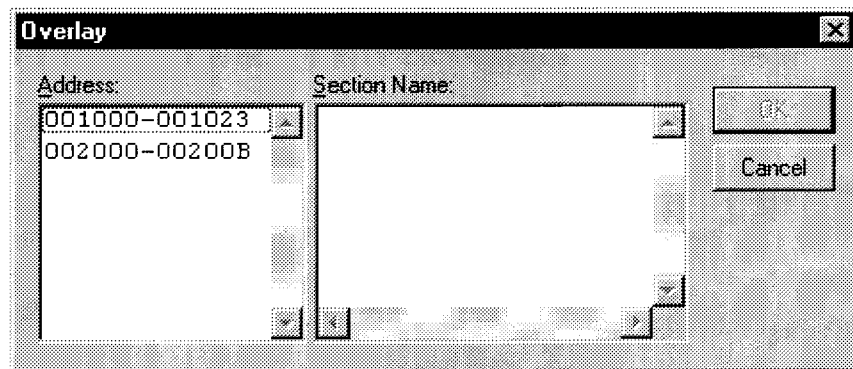


**Figure 9.1     Overlay Dialog Box (at Opening)**

This dialog box has two areas: the Address list box and the Section Name list box.

The Address list box displays the address ranges used by the overlay function. Click to select one of the address ranges in the Address list box.
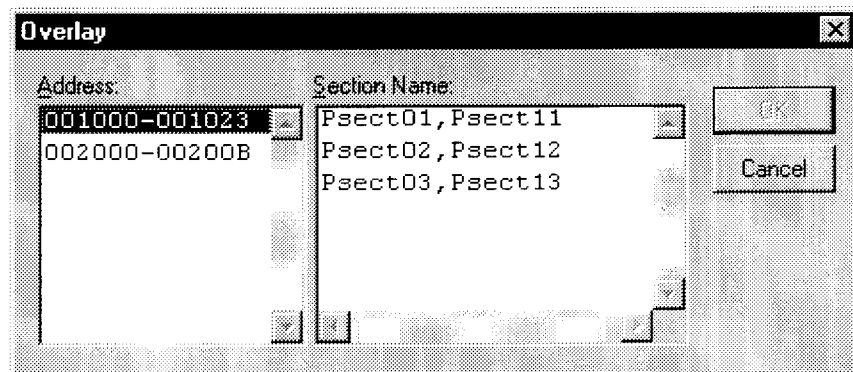


**Figure 9.2     Overlay Dialog Box (Address Range Selected)**

The Section Name list box displays the section groups assigned to the selected address range.

## 9.2 Setting Section Group

When using the overlay function, the highest-priority section group must be selected in the Overlay dialog box; otherwise HDI will operate incorrectly.

First click one of the address ranges displayed in the Address list box. The section groups assigned to the selected address range will then be displayed in the Section Name list box.

Click to select the section group with the highest-priority among the displayed section groups.
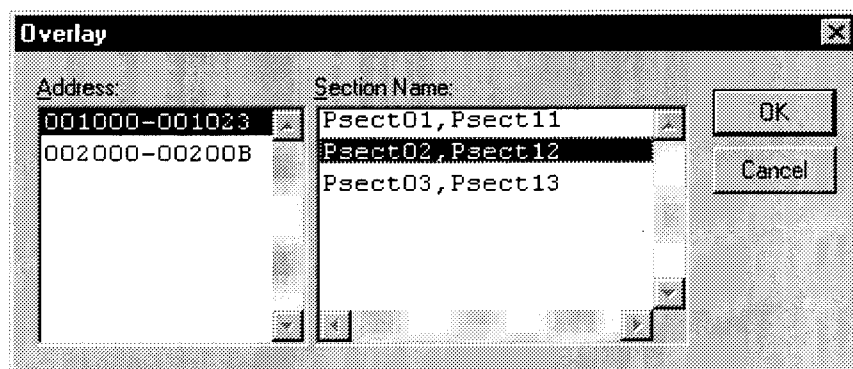


**Figure 9.3    Overlay Dialog Box (Highest-Priority Section Group Selected)**

After selecting a section group, clicking the [OK] button stores the priority setting and closes the dialog box. Clicking the [Cancel] button closes the dialog box without storing the priority setting.

**Note**   Within the address range used by the overlay function, the debugging information for the section specified in the Overlay dialog box is referred to. Therefore, the same section of the currently loaded program must be selected in the Overlay dialog box.

# 10. Selecting Functions

When selecting overloaded functions or member functions that can be used in C++ programs, follow the description in this section.

## 10.1 Displaying Functions

Use the Select Function dialog box to display overloaded functions and member functions.

A function can be selected in the following cases.

- When setting a breakpoint

- When specifying a function in the Run Program dialog box

- In the Set Address dialog box for opening the Source window

- In the Set Address dialog box for opening the Memory window

- When adding or modifying a symbol

- When specifying a function for performance analysis

When overloaded functions have the same specified function name, or when a class name including a member function is specified, the Select Function dialog box opens.
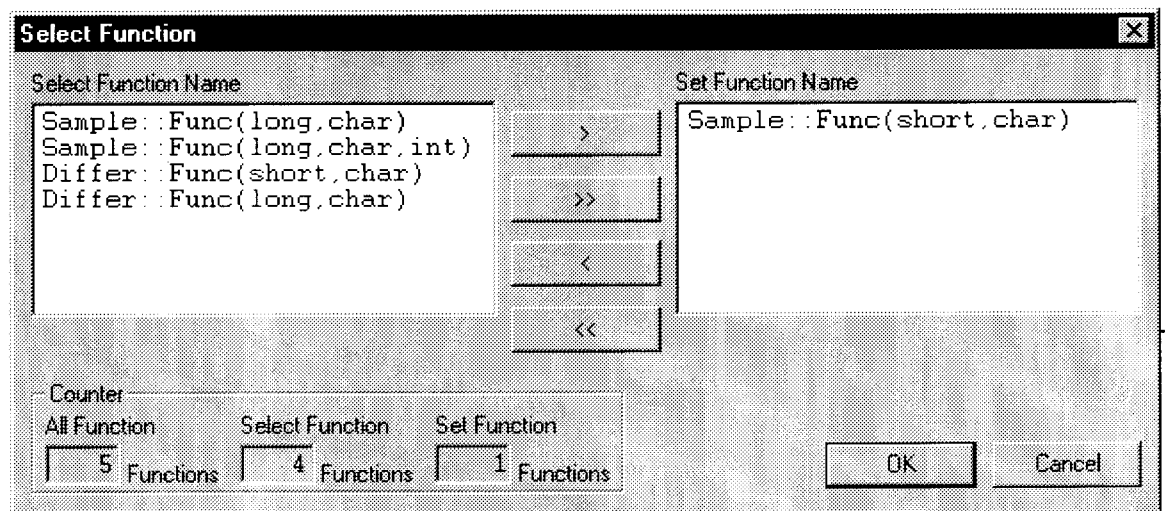


Figure 10.1    Select Function Dialog Box

This dialog box has three areas.

- Select Function Name list box

  Displays the overloaded functions or member functions and their detailed information.

- Set Function Name list box

  Displays the function to be set and their detailed information.

- Counter group edit box

  | | |
  |---|---|
  | All Function | Displays the number of overloaded functions or member functions. |
  | Select Function | Displays the number of functions displayed in the Select Function Name list box. |
  | Set Function | Displays the number of functions displayed in the Set Function Name list box. |

## 10.2 Specifying Functions

Select overloaded functions or member functions in the Select Function dialog box. Generally, one function can be selected at one time; only for setting breakpoints, The breakpoint is set The plural can be selected in the function setting by the Run Program dialog box and setting the function of the performance analysis.

### 10.2.1 Selecting a Function

Click the function you wish to select in the Select Function Name list box, and click the [>] button. You will see the selected function in the Set Function Name list box. To select all functions in the Select Function Name list box, click the [>>] button.

### 10.2.2 Deleting a Function

Click the function you wish to delete from the Set Function Name list box, and click the [<] button. To delete all functions in the Set Function Name list box, click the [<<] button.

### 10.2.3 Setting a Function

Click the [OK] button to set the functions displayed in the Set Function Name list box. The functions are set and the Select Function dialog box closes.

Clicking the [Cancel] button closes the dialog box without setting the functions.

# 11. Configuring the User Interface

When we designed the user interface for HDI we tried to make all the frequently used operations quickly accessible and have related operations grouped in a logical order. However, when you are in the middle of a heavy debugging session you may find it more useful to have a different arrangement of the user interface items or you may just have a personal preference for the way you want it arranged. We realize this and so HDI allows you to customize the user interface so that you can be satisfied with the tool that you are using for debugging your program. This section describes how you can arrange the user interface windows, customize various aspects of the display and save the configuration.

## 11.1 Arranging Windows

### 11.1.1 Minimizing Windows

If you have temporarily finished using an open window but want to be able to look at it in its current state later, you can reduce it to an icon. This is called *minimizing* the window. To minimize a window, either click on the minimize button of the window, or choose the [▓] -> **Mi_nimize**] window menu option.

Control menu
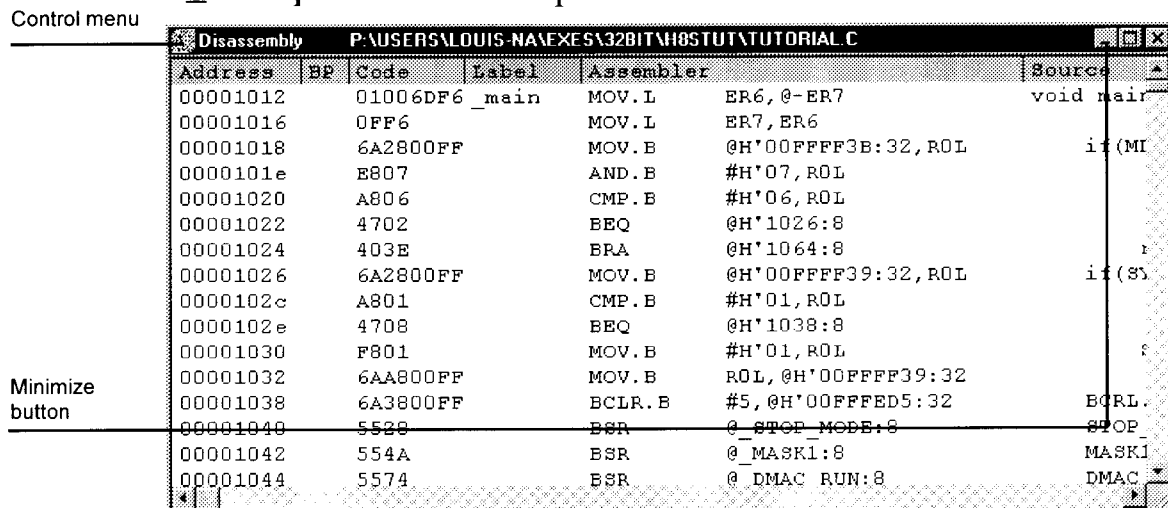


Figure 11.1   Minimizing a Window

The window is minimized to an icon at the bottom left of the HDI application window; for the above Disassembly window example the icon is:



Figure 11.2   Disassembly Window Icon

**Note**   You may not be able to see the icon if you have a window open over the bottom of the screen.

To restore the icon back to a window, either double-click on the icon, or choose the **[Restore]** menu option from the control menu.

## 11.1.2 Arranging Icons

Although the icons will be put at the bottom left of the HDI application window by default when you minimize a window, you can move them anywhere you like in the application window by simply clicking and dragging them to a new position. When you restore the icon to a window, the window will be at the same position that it was in when you minimized it. Similarly, when you minimize it again, the icon will be placed at the last position that you moved it to.

When you have many minimized windows as icons, the display can look rather messy. To tidy up the icons, choose the **[Window->Arrange Icons]** menu option.

The icons will be arranged in order from the bottom left of the application window:
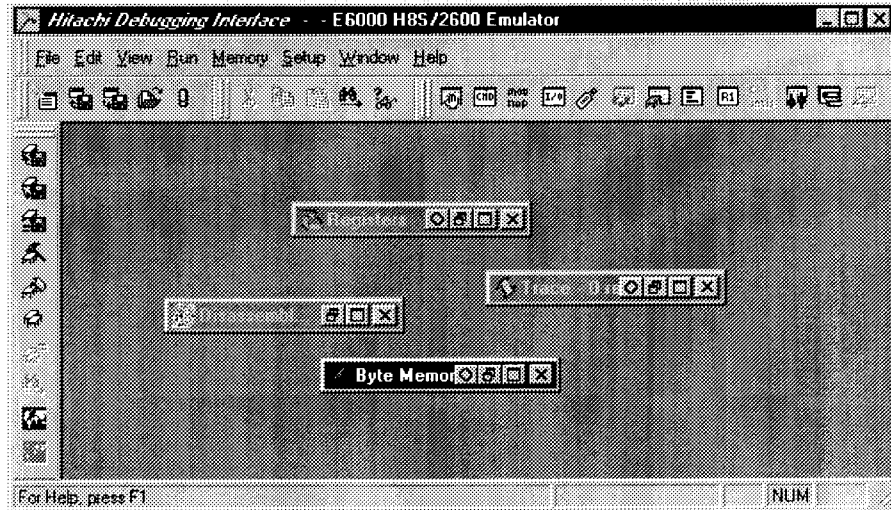


**Figure 11.3    Icons Before Arrangement**
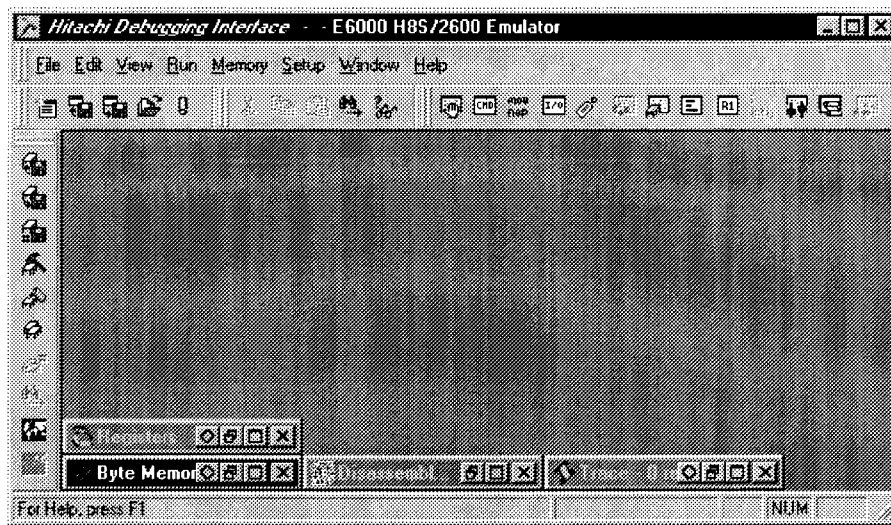
**Figure 11.4   Icons After Arrangement**

## 11.1.3 Tiling Windows

After some heavy debugging you may find that you have many windows open on the screen. You can arrange all the windows in a tile format with none of them overlapping each other using the Tile function by choosing the **[Window->Tile]** menu option.

All currently open windows are arranged in a tile format. Windows that are minimized to icons are not affected.

## 11.1.4 Cascading Windows

Open windows can also be arranged in a cascading format with only their left and top border visible under the window in front of them by choosing the **[Window->Cascade]** menu option. All currently open windows are arranged in a cascading format. Windows that are minimized to icons are not affected.

# 11.2 Locating Currently Open Windows

When you have many windows open in the HDI application window it is quite easy to *lose* one of them behind the others. There are two methods that you can use to find the lost window:

## 11.2.1 Locating the Next Window

To bring the next window in the window list to the front of the display, choose **[Next]** from the window menu, or press **CTRL+F6.** Repeating this operation will cycle selection of all windows (open and minimized).

### 11.2.2 Locating a Specific Window

To select a specific window, choose from the list of windows (open and minimized) at the bottom of the [**Window**] menu. The currently selected window has a check mark next to it in the window list. In the following example, the Disassembly window is the currently selected window:
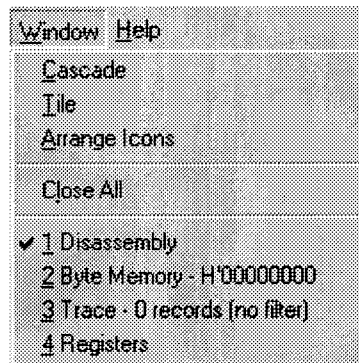


**Figure 11.5    Selecting a Window**

The window that you select will be brought to the front of the display. If it is minimized the icon is restored to a window.

## 11.3 Enabling/Disabling the Status Bar

You can select whether or not the Status bar is displayed at the bottom of the HDI application window; by default it will be displayed. To disable display of the Status bar, choose the [**Setup->Status Bar**] menu option.

The Status bar will be disabled and removed from the HDI application window display. To re-enable the Status bar display, choose the [**Setup->Status Bar**] menu option again. The Status bar will be enabled and added to the HDI application window display.

## 11.4 Customizing the Toolbar

To control the selection and arrangement of buttons displayed on the toolbar, choose the [**Setup->Customize->Toolbar...**] menu option.

The Customize Toolbar dialog box opens and contains two panes. The first pane 'Toolbars' is used to set the overall appearance of the toolbars, while the second pane 'Commands' is used to set the individual buttons in each toolbar.

## 11.4.1 Overall Appearance

Select the Toolbars pane to set the overall appearance of the toolbars:
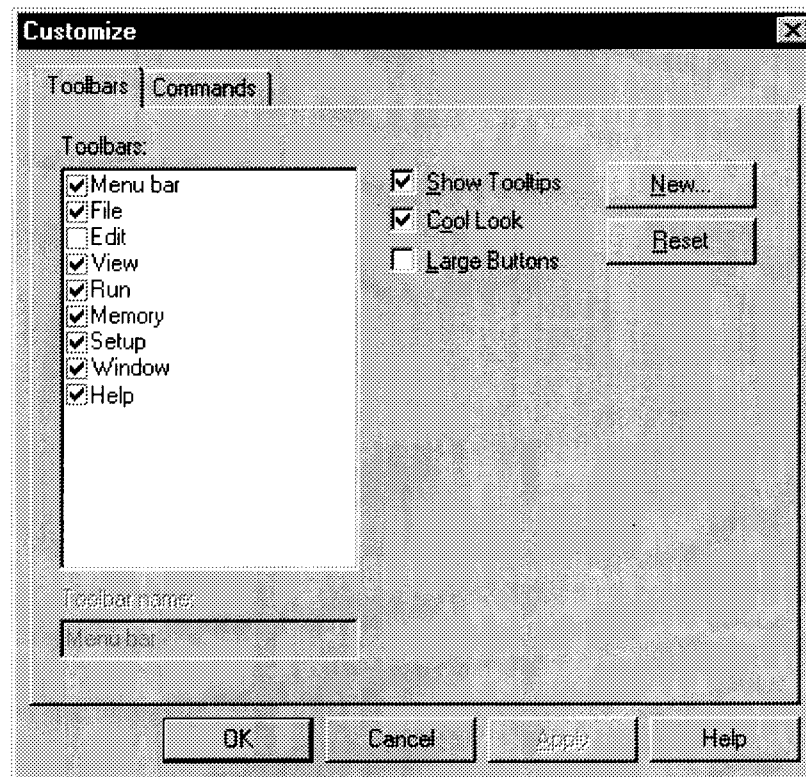


**Figure 11.6   Customize Toolbar (Toolbars) Dialog Box**

The toolbars are listed in a multi-selection list box - to individually switch off a toolbar, clear the check box next to the name (this name is displayed in a mini-title bar when the toolbar is not attached to the border of the main frame window).

**Note**   The menu bar cannot be switched off.

If you need to conserve desktop area (for example, when using a portable) then clear the 'Cool Look' check box to revert to the classic Windows® 3.1 style menu and toolbars.

It is possible to add user-defined toolbars - click on the [**New...**] button and enter a name for your toolbar. This can be edited later in the Toolbar Name edit box (feature only available for user defined toolbars). The new toolbar, in this case called 'My Toolbar', will appear floating at the top-left of the main frame but will have no buttons. To add buttons, you will now have to customize your toolbar.

## 11.4.2 Customizing Individual Toolbars

Customizing individual toolbars requires a mouse or other pointing device - the feature is not available if only the keyboard is available. This is because the toolbars only operate with a mouse, so customizing them would be unnecessary unless you have a mouse.

Select the Commands pane to set the individual buttons in each toolbar:

Button
categories

Buttons
available

Description
of button's
operation
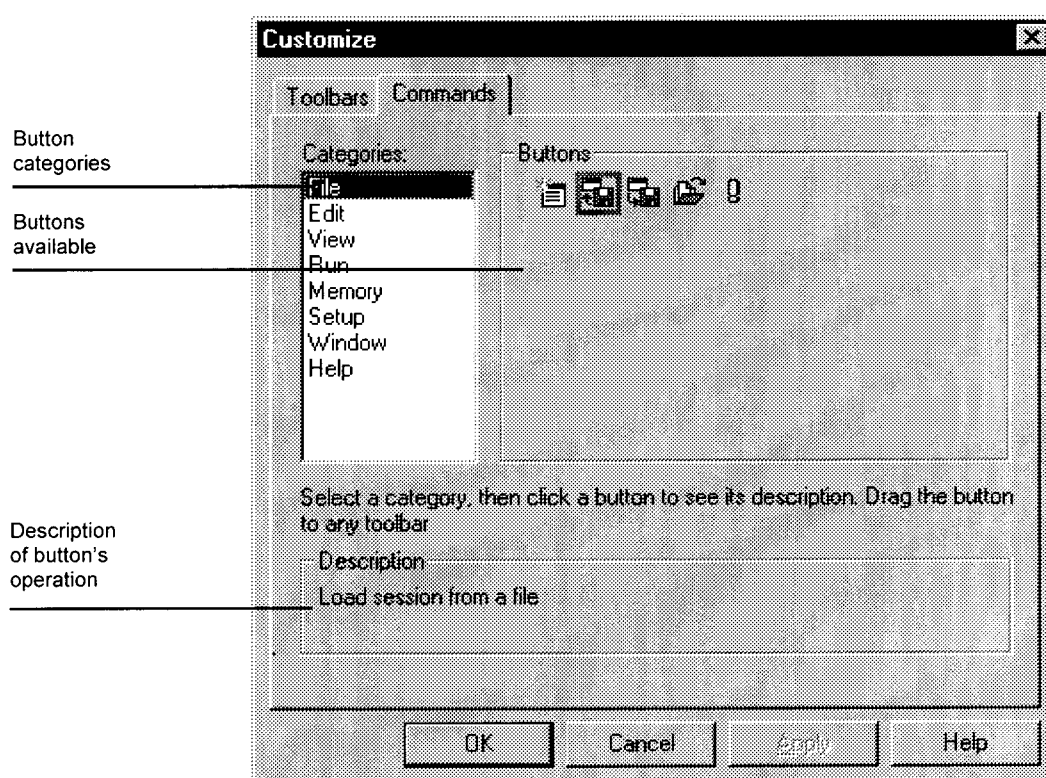


**Figure 11.7   Customize Toolbar (Commands) Dialog Box**

## 11.4.3 Button Categories

At the top left of the dialog box is a list of button categories. For each category a list of buttons within that category will be displayed to the right. Click on a button operation option in the list and you will see a description of the button's operation in the Button Description field.

## 11.4.4 Adding a Button to a Toolbar

➲ To add a button to a toolbar:

1. Select the button category from the button category list.

2. Select the button item from the operation list.

3. Drag the button from the dialog to the toolbar location you wish to add the new button.

The button is inserted into the toolbar..

### 11.4.5 Positioning a Button in a Toolbar

➲ To move a button position in a toolbar:

1. Select the button in a toolbar.

2. Drag the button to the new position in the toolbar or another toolbar.

**Note**    Holding down the Ctrl key while dragging will copy the button.

### 11.4.6 Removing a Button from a Toolbar

➲ To remove a button in a toolbar:

1. Select the button in a toolbar.

2. Drag the button out of the toolbar (anywhere into the main frame).

## 11.5 Customizing the Fonts

You can customize the display font for text style windows (e.g. Source and Memory windows), or change the default font that is used when a new window is opened.

To change the display font, choose the [Setup->Customize->Font] menu option. This will launch the Font dialog box:
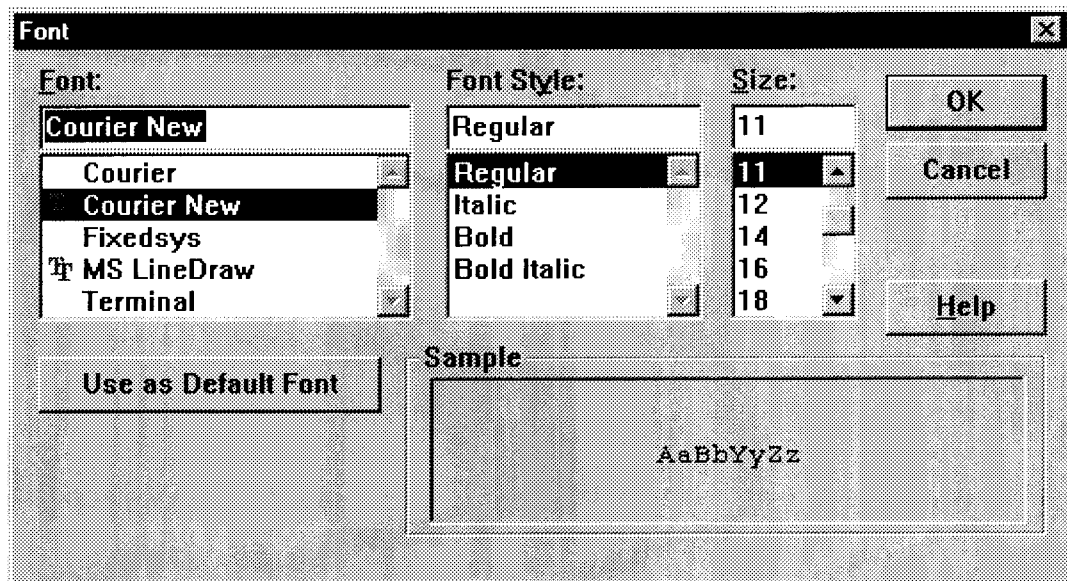


**Figure 11.8   Font Dialog Box**

The dialog box is based on the standard Windows® font selection dialog box, except that only fixed width fonts are listed in the Font list box.

## 11.6 Customizing the File Filters

You can customize the file filters displayed in the Open dialogs.

To change the filters, choose the [Setup->Customize->File Filter] menu option. This will launch the Customize File Filter dialog box:
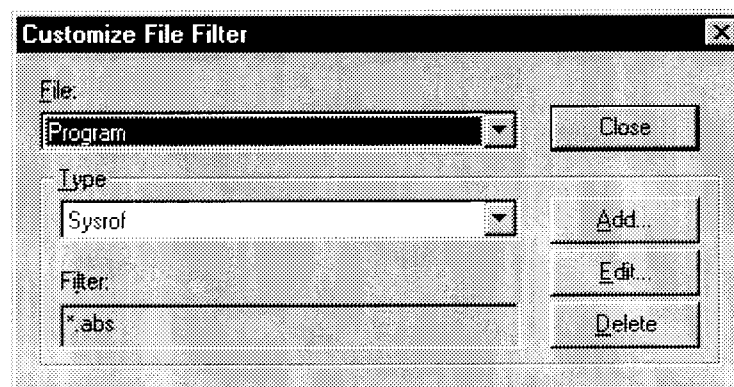


**Figure 11.9   Customize File Filter Dialog Box**

**Note**   Changes are made immediately when using this dialog. There is no option to cancel changes made.

⮥ To edit an existing filter:

1. Select the file group from the File drop list.

2. Select the file type name from the Type drop combo.

3. Click the [**Edit...**] button to open the Edit Filter dialog box. The dialog title will display the file group that is being changed. The edit controls on this dialog are limited to accept only valid characters.

4. Change the filter name and/or extension. If more than one extension is required, then separate each extension with a semi-colon. For example:

    *.mot;*.a20;*.a37

⮥ To enter a new filter:

1. Select the file group from the File drop list.

2. Click the [**Add...**] button to open the Add Filter dialog box. The dialog title will display the file group that is being changed. The edit controls on this dialog are limited to accept only valid characters.

3. Enter a name for the filter type and the extensions you want to use for the filter.

**Note**    If the filter type entered matches an existing type, then the filter for the existing type will be changed to the newly entered filter.

⮥ To remove a filter:

1. Select the file group from the File drop list.

2. Select the file type name from the Type drop combo.

3. The file type will be removed when the [**Delete**] button is clicked.

## 11.7 Saving a Session

If you have downloaded user program into the debugging platform, have the corresponding source files displayed and a number of auxiliary windows open, then it can take some time to setup this information the next time the program is loaded. To help with this, HDI can save the current settings to a file.

If you are already using a named session, or want to create a session with the same name as the current object file, choose the [File->Save Session] menu option.

To save the current setting under a new name, choose the [File->Save Session As...] menu option. This will launch a common file dialog box prompting you for a file name. Up to three files are saved, an HDI session file (*.hds); a target session file (*.hdt); and a watch session file (*.hdw). The first includes the HDI interface settings, e.g. all the open windows and their positions. The second includes the settings specific to the debugging platform/user system, e.g. the name of the debugging platform and its configuration. The third is only created if a Watch window is open and it includes a list of the variables currently being watched.

The session name is then displayed as the second entry in HDI's title bar. For example, if you are using the H8 smart card simulator and the session name is "MANUAL", then the title bar will contain the text:

"*Hitachi Debugging Interface* – MANUAL – H8/3103 Simulator"

**Note**    The session file does not include symbol or memory information. To use modified information again in later sessions, save the symbol and memory information in appropriate files. For details, see *section 5.7, Saving an Area of Memory* and *section 13.5.8, Save As....*

## 11.8 Loading a Session

To reload a saved session, choose the [File->Load Session...] menu option. This will launch a standard Windows[R] file dialog box prompting you for an HDI session file name (*.hds).

Any currently open windows will be closed, and the connection to the debugging platform initialized. If user program has been downloaded to the user system, then the status bar will display the percentage done.

When the download is complete, windows will be opened and refreshed to show the latest information from the user system.

## 11.9 Setting HDI Options

There are a number of settings available to help you to use the HDI interface. Choosing the [Setup->Options...] menu option will launch the HDI Options dialog box:
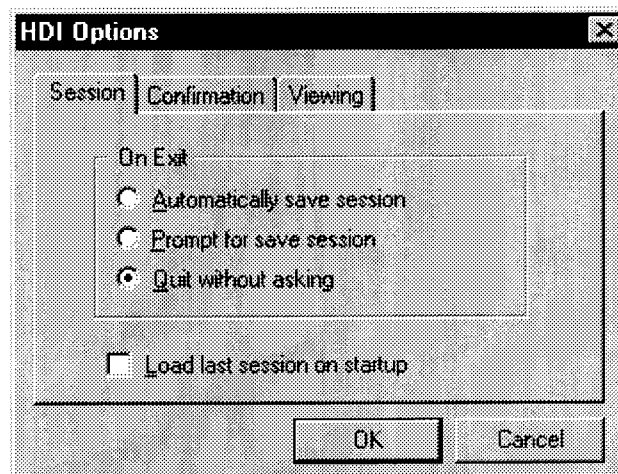


**Figure 11.10 HDI Options (Session) Dialog Box**

The 'On Exit' group of radio buttons automates saving the current session when the program is shut down:

- Automatically save session - this will save the session information in the current session file. If there is no current session file then you will be prompted to enter an HDI session file name.

- Prompt for save session - this will always ask you if you want to save the current session when the program shuts down. If you select 'Yes', then the session information is saved in the current session file. If there is no current session file then you will be prompted to enter a session file name.

- Quit without asking - this shuts down the program and does not prompt you, nor save the current session information.

Check the 'Load last session on startup' check box if you want to automatically load the last saved session the next time the program is started.
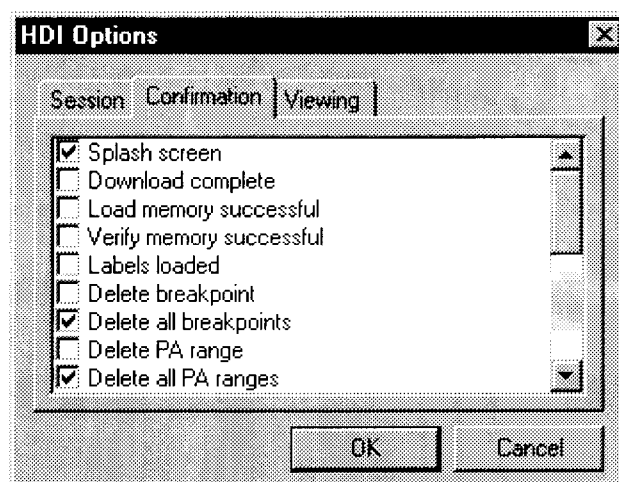
**Figure 11.11 HDI Options (Confirmation) Dialog Box**

Confirmation message boxes can be switched off or on by using the appropriate confirmation check box.
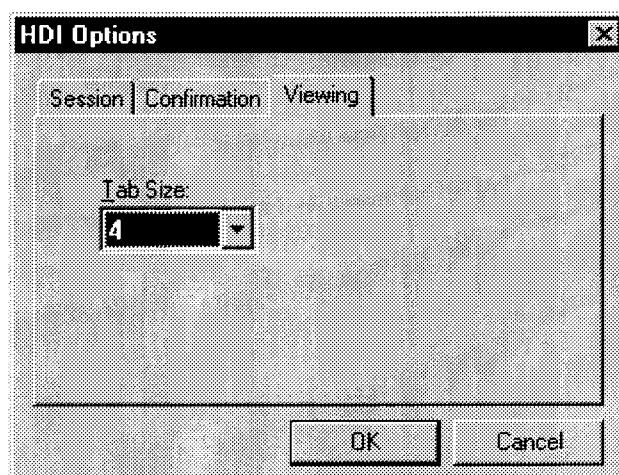


**Figure 11.12 HDI Options (Viewing) Dialog Box**

The 'Tab Size' list box can be used to set the number of spaces that a tab character will be expanded to within the views. Valid values are between 2 and 8 - the best value will be the same as your normal editor.

## 11.10 Setting the Default Input Radix

HDI can accept input in several numerical bases. The default is hexadecimal (except Count fields which are always decimal), but you can also use one of the prefixes described in *section 2.2.2, Data Formats*. To improve usability, you can select one of these formats as the default, i.e. you will not need to enter the corresponding prefix to use that radix.

To change the default radix, choose the **[Setup->Radix]** menu option. This will display a list of possible numbering systems with a check mark to the left of the current radix:
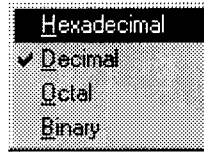


**Figure 11.13 Setting Radix**

# 12. Menus

This document uses the standard Microsoft® menu naming convention.

Menu title

Menu bar

Drop-down men...

Menu option

Ellipsis

Check mark

Cascading menu
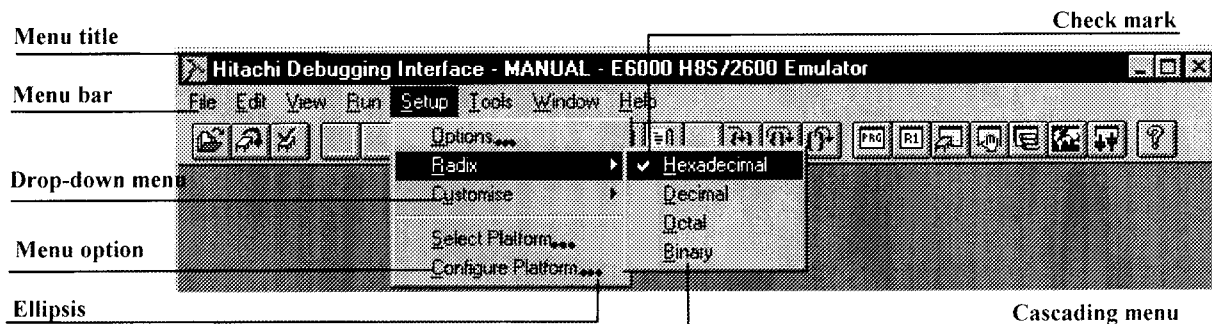
**Figure 12.1 Menus**

Check marks indicate that the feature provided by the menu option is selected.

Ellipsis indicates that selecting the menu option will open a dialog box that requires extra information to be entered.

Refer to your Windows® user manual for details on how to use the Windows® menu system.

## 12.1 File

The File menu is used for aspects of the program that access program files.

### 12.1.1 New Session...

Launches the Select Session dialog box allowing the user to select a new debugging platform.

### 12.1.2 Load Session...

Launches the Select Session dialog box allowing the user to load a session from a selected session file (*.hds extension). A session file contains the debugging platform's settings, and the current program and the position of open child windows (views) - it contains symbols, breakpoints, or current register values.

### 12.1.3 Save Session

Updates the session file for the current session file. If there is no current session file defined, this acts in a similar manner to the Save Session As... menu option.

### 12.1.4 Save Session As...

Launches the Save As dialog box allowing the user to save the current session details under a new file name. A session file contains the debugging platform's settings, and the current program and the position of open child windows (views) - it contains symbols, breakpoints, or current register values.

### 12.1.5 Load Program...

Launches the Load Object File dialog box, allowing the user to select an object file in either S-Record (*.mot; *.s20; and *.obj extensions), SYSROF (*.abs extension), or ELF/DWARF (*.abs extension) format and download it to the debugging platform's memory. This will also load the symbols if they are available in the selected file.

### 12.1.6 Initialize

This will attempt to re-initialize the debugging system. It will close down any open child windows and shut down the link to the debugging platform. If this is successful, an attempt to re-establish the link to the debugging platform will be made. The message 'Link up' will appear in the left-most box of the status bar if this is successful. (See also *section 12.4.1, Reset CPU*)

### 12.1.7 Exit

This will close down the HDI. The actions that are carried out by the HDI can be defined by the user in the 'On Exit' section of the HDI Options dialog box. (See also *section 12.6.2, Options...*)

## *12.2 Edit*

The Edit menu is used for aspects of the program that access or modify data in the child windows and debugging platform.

### 12.2.1 Cut

Only available if a block is highlighted in a child window whose contents can be modified.

This will remove the contents of the highlighted block from the window and place it on the clipboard in the standard Windows® manner.

### 12.2.2 Copy

Only available if a block is highlighted in a child window.

This will copy the contents of the highlighted block to the clipboard in the standard Windows® manner.

### 12.2.3 Paste

Only available if the contents of the child window can be modified.

This will copy the contents of the Windows® clipboard into the child window at the current cursor position.

### 12.2.4 Find...

Only available if the window contains text.

This will launch the Find dialog box allowing the user to enter a word and locate occurrences within the text. If a match is found, the cursor will move to the start of the word.

### 12.2.5 Evaluate...

Launches the Evaluate dialog box allowing the user to enter a numeric expression, e.g. "(#pc + 205)*2", and display the result in all currently supported radices.

## 12.3 View

The View menu is used to select and open new child windows. If the menu option is grayed, then the features provided by the window are not available with the current debugging platform.

### 12.3.1 Breakpoints

Opens the Breakpoints window allowing the user to view and edit current breakpoints.

### 12.3.2 Command Line

Opens the Command Line window allowing the user to enter text-based commands to control the debugging platform. These commands can be piped in from a batch file, and the results piped out to a log file, allowing automatic tests to be performed.

### 12.3.3 Disassembly...

Launches the Set Address dialog box allowing the user to specify the memory block position that you wish to view.

### 12.3.4 I/O Area

Opens the I/O Registers window allowing the user to control the user systems on-chip input/output functionality, e.g. an interrupt controller.

### 12.3.5 Labels

Launches the Labels window allowing the user to manipulate the current program's symbols (labels).

### 12.3.6 Locals

Opens the Locals window allowing the user to view and edit the values of the variables defined in the current function. The contents are blank unless the PC is within a C/C++ source-level function.

### 12.3.7 Memory...

Launches the Open Memory Window dialog box allowing the user to specify a memory block and view format to display within a Memory window.

### 12.3.8 Performance Analysis

Launches the Performance Analysis window allowing the user to set up and view the number of times that particular sections of the user program have been called.

### 12.3.9 Registers

Opens the Registers window allowing the user to view all the current CPU registers and their contents.

### 12.3.10 Source...

Launches the Open dialog box allowing the user to enter a file name of the source file (in either C/C++ or assembly language format) to view. If the source file is not included within the current program or there is no debugging information for the file within the 'absolute' (*.abs) file, then the message "Cannot load program. No Source level debugging available" is displayed.

### 12.3.11 Status

Opens the System Status window allowing the user to view the debugging platform's current status and the current session and program names.

### 12.3.12 Trace

Opens the Trace window allowing the user to see the current trace information.

### 12.3.13 Watch

Opens the Watch window allowing the user to enter C/C++-source level variables and view and modify their contents.

## *12.4 Run*

The Run menu controls the execution of the user program in the debugging platform.

### 12.4.1 Reset CPU

Resets the user system hardware and sets the PC to the *Reset Vector* address. (See also *section 12.1.6, Initialize*)

### 12.4.2 Go

Starts executing the user program at the current PC.

### 12.4.3 Reset Go

Resets the user system hardware and sets the PC to the *Reset Vector* address before executing the user program.

### 12.4.4 Go To Cursor

Starts executing the user program at the current PC and continues until the PC equals the address indicated by the current text cursor (not mouse cursor) position.

### 12.4.5 Set PC To Cursor

Changes the value of the Program Counter (PC) to the address at the row of the text cursor. Disabled if no address is available for the current row.

### 12.4.6 Run...

Launches the Run Program dialog box allowing the user to enter temporary breakpoints before executing the user program.

### 12.4.7 Step In

▣ Executes a block of user program before breaking. The size of this block is normally a single instruction but may be set by the user to more than one instruction or a C/C++-source line (see also *section 12.4.10, Step...*). If a subroutine call is reached, then the subroutine will be entered and the view is updated to include its code.

### 12.4.8 Step Over

▣ Executes a block of user program before breaking. The size of this block is normally a single instruction but can be set by the user to more than one instruction or a C/C++-source line (see also *section 12.4.10, Step...*). If a subroutine call is reached, then the subroutine will not be entered and sufficient user program will be executed to set the current PC position to the next line in the current view.

### 12.4.9 Step Out

▣ Executes sufficient user program to reach the end of the current function and set the PC to the next line in the calling function before breaking.

### 12.4.10 Step...

▣ Launches the Step Program dialog box allowing the user to modify the settings for stepping.

### 12.4.11 Halt

▣ Stops the execution of the user program.

## 12.5 Memory

The Memory menu is used for aspects of the program that access memory.

### 12.5.1 Refresh

Forces a manual update of the contents of all open Memory windows.

### 12.5.2 Load...

Launches the Load Memory File dialog box, allowing the user to select an offset address in the memory area, and file name to load from an S-Record format file on disk.

### 12.5.3 Save...

Launches the Save Memory As dialog box, allowing the user to select a start and an end address in the memory area, to save to an S-Record format file on disk. If a block of memory is highlighted in a Memory window, these will be automatically entered as the start and end addresses when the dialog box is displayed.

### 12.5.4 Verify...

Launches the Verify S-Record File with Memory dialog box, allowing the user to select a start and an end address in the memory area to check against the contents of an S-Record file on disk.

### 12.5.5 Test...

Launches the Test Memory dialog box allowing the user to specify a block of memory to test for correct read/write operation. The exact test is target dependent. However, in all cases the current contents of the memory will be overwritten - YOUR PROGRAM AND DATA WILL BE ERASED.

### 12.5.6 Fill...

Launches the Fill Memory dialog box allowing the user to fill a block of the debugging platform's memory with a value.

### 12.5.7 Copy...

Launches the Copy Memory dialog box allowing the user to copy a block of the debugging platform's memory to an address within the same memory area. The blocks may overlap, in which case any data within the overlapped region of the source block will be overwritten.

### 12.5.8 Compare...

[icon] Launches the Compare Memory dialog box, allowing the user to select a start and an end address in the memory area, to check against another area in memory. If a block of memory is highlighted in a Memory window, these will be automatically entered as the start and end addresses when the dialog box is displayed.

Similar to Verify memory, but compares two blocks in memory.

### 12.5.9 Configure Map...

[icon] Opens the Memory Mapping window allowing the user to view and (if supported) edit the debugging platform's current memory map. In some debugging platforms, the Memory Map dialog box will open.

### 12.5.10 Configure Overlay...

[icon] Launches the Overlay dialog box. When the overlay function is used, the target section group can be selected in the dialog box.

## 12.6 Setup

The Setup menu is used to modify the settings of the HDI user interface, and the configuration of the debugging platform.

### 12.6.1 Status Bar

Toggles the status bar feature on and off. If the feature is enabled then a check mark will be displayed to the left of the menu text.

### 12.6.2 Options...

[icon] Launches the HDI Options dialog box allowing the user to modify the settings that are specific to the HDI (not debugging platform dependent settings).

### 12.6.3 Radix
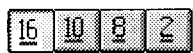
[icon] Cascades a menu displaying a list of radix in which the numeric values will be displayed and entered by default (without entering the radix prefix). The current radix has a check mark to its left and the associated toolbar button is locked down.

For example, if the current radix is decimal then the number ten will be displayed as "10" and may be entered as "10", "H'A", "0x0a", etc.;

if the current radix is hexadecimal then the number ten will be displayed as "0A" and entered as "A", "D'10", etc.

### 12.6.4 Customize
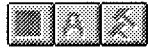
Cascades a menu displaying a list of options that can be customized by the user.

**Toolbar** :When this cascade menu option is selected, the Customize dialog box is launched.

**Font** :When this cascade menu option is selected, the Font dialog box is launched, allowing a fixed width font to be selected.

**File Filter** : When this cascade menu option is selected, the File Filter dialog box is launched, allowing the browser file filters for Object, Source and Memory files to be changed to match the user's requirements.

### 12.6.5 Configure Platform...

Launches a set-up dialog box specific to the selected debugging platform. Refer to the debugging platform's user manual for more detail about the options available in the dialog box.

## 12.7 Window

The Window menu modifies the display of currently open child windows. The following menu options are always displayed, and a numbered list of current child windows will be appended - the topmost child window will have a check mark.

### 12.7.1 Cascade

Arranges the child windows in the standard cascade manner, i.e. from the top left such that the title bar of each child window is visible.

### 12.7.2 Tile

Arranges the child windows in the standard tile manner, i.e. sizes each window such that all are displayed without overlapping.

### 12.7.3 Arrange Icons

Lines up any iconized windows neatly along the bottom of the parent frame in the standard manner.

### 12.7.4 Close All

Closes all the child windows.

## 12.8 Help

The Help menu accesses additional information on how to use the functionality provided by HDI.

### 12.8.1 Index

Opens the main help file at the index.

### 12.8.2 Using Help

Opens a help file allowing the user to find out how to use Windows$^{®}$ hypertext help system.

### 12.8.3 Search for Help on

Opens the main help file and launches the Search dialog box allowing the user to enter and browse through the file's keywords.

### 12.8.4 About HDI

Launches the About HDI dialog box allowing the user to view the version of HDI and the currently loaded DLLs.

# 13. Windows

This section describes each child window type, the features that each window supports and the options available through their associated popup menu.

## 13.1 Breakpoints



| Enable | File/Line | Symbol | Address | Type |
|--------|-----------|--------|---------|------|
| ● | startup.src/24 | startup_VECT | 00000000 | Type=PC |
| ○ | | | 00000100 | Type=Access |
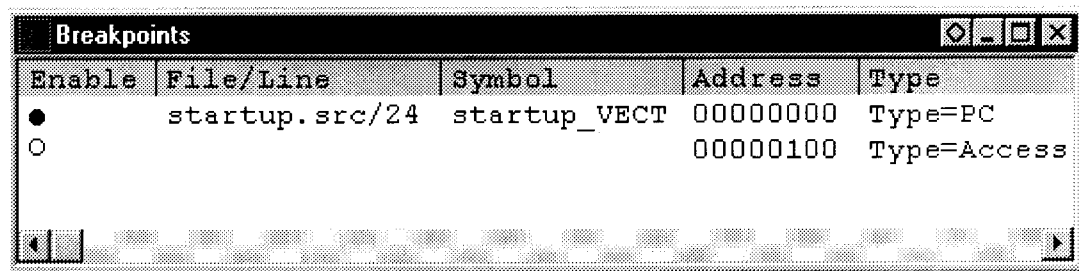
**Figure 13.1   Breakpoints Window**

Allows the user to view and control current breakpoints and to view the hardware breakpoint resources. For more information regarding supported breakpoint types and resources, refer to the *Debugging Platform User's Manual*.

A popup menu containing the following options is available by right clicking within the window.

### 13.1.1 Add...

Launches the Breakpoint/Event Properties dialog box allowing the user to enter a new breakpoint. The dialog box is dependent on the debugging platform.

### 13.1.2 Edit...

Only enabled if a breakpoint is selected. Launches the Breakpoint/Event Properties dialog box allowing the user to modify the properties of an existing breakpoint. The dialog box is dependent on the debugging platform.

### 13.1.3 Delete

Only enabled if a breakpoint is selected. Removes the selected breakpoint. To retain the details of the breakpoint but not have it cause a break when its conditions are met, use the Disable option (see *section 13.1.5, Disable/Enable*).

### 13.1.4 Delete All

Removes all breakpoints from the list.

### 13.1.5 Disable/Enable

Only enabled if a breakpoint is selected. Toggles the selected breakpoint between enabled and disabled (when disabled, a breakpoint remains in the list, but does not cause a break when the specified conditions are satisfied). When a breakpoint is enabled, a check mark is shown to the left of the menu text (and a circle is shown in the Enable column for the breakpoint).

### 13.1.6 Go To Source

Opens Source or Disassembly window at address of breakpoint.

## 13.2 Command Line
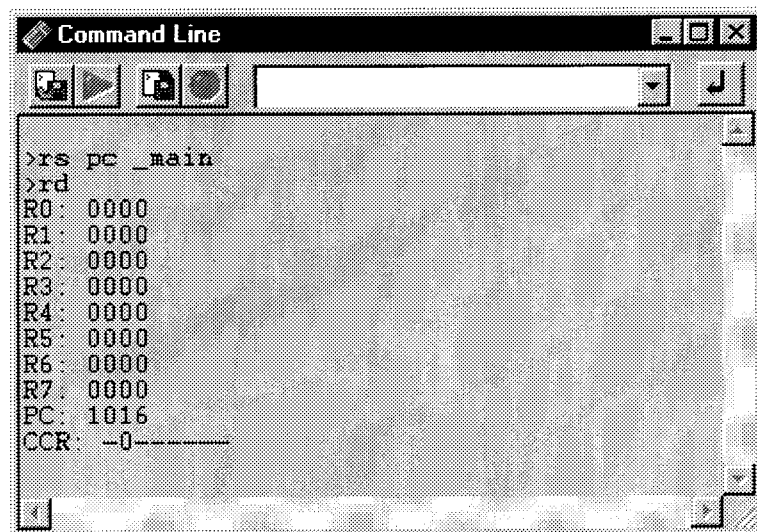


**Figure 13.2   Command Line Window**

Allows the user to control the debugging platform by sending text-based commands instead of the window menus and commands. It is useful if a series of predefined commands need to be sent to the debugging platform by calling them from a batch file and, optionally, recording the output in a log file. The command can be executed by pressing 'Enter' after the command is input to the text box (Or, the Enter button in the right of the text box is clicked). For information about the available commands, refer to the on-line help.

If available, the window title displays the current batch and log file names separated by colons.

The functionality of the toolbar buttons is identical to the popup menu options shown below.

### 13.2.1 Run Batch File...

Launches the Run Batch File dialog box, allowing the user to enter the name of an HDI command file (*.hdc). The batch file is then run automatically. The name of the file is shown on the window title bar.

### 13.2.2 Play

Runs the last entered HDI command file (*.hdc). It is displayed in a recessed state while the batch file is running and can be used to stop an executing batch file and return control to the user.

### 13.2.3 Set Log File...

Launches the Open Log File dialog box, allowing the user to enter the name of an HDI log file (*.log). The logging option is automatically set and the name of the file shown on the window title bar.

Opening a previous log file will ask the user if they wish to append or over-write the current log.

### 13.2.4 Logging
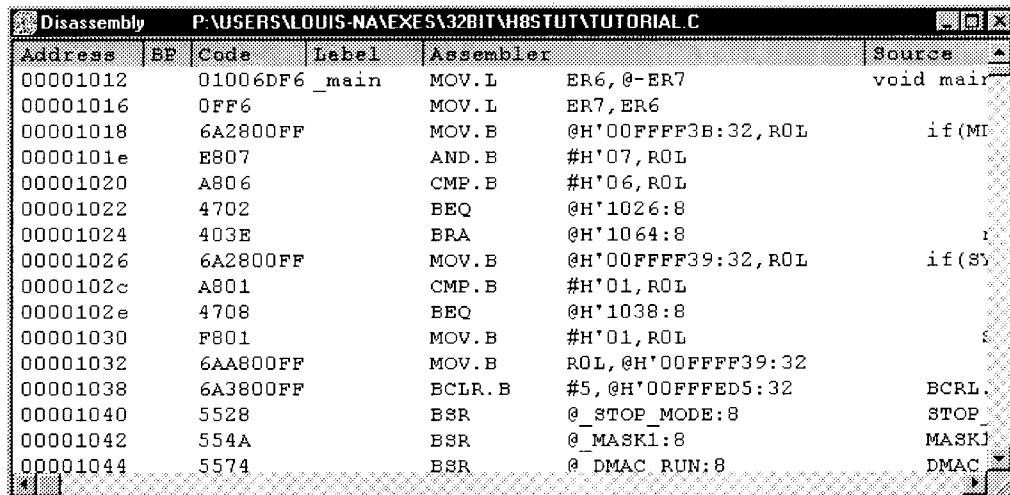
Toggles logging to file on and off. When logging is active, the button becomes effective. Note that the contents of the log file cannot be viewed until logging is completed, or temporarily disabled by clearing the check box. Re-enabling logging will append to the log file.

## 13.3 Disassembly

This window is used to display code at the assembly-language level.

This window layout has a different layout to the Source window, with an additional column Label which displays the symbol/label name (if available) for that address. Assembler information is obtained by disassembling the memory contents, and may be edited or viewed directly from memory without requiring debug information from the object file.

| Disassembly | P:\USERS\LOUIS-NA\EXES\32BIT\H8STUT\TUTORIAL.C | | | | |
|---|---|---|---|---|---|
| Address | BP | Code | Label | Assembler | Source |
| 00001012 | | 01006DF6 | _main | MOV.L | ER6,@-ER7 | void mair |
| 00001016 | | 0FF6 | | MOV.L | ER7,ER6 | |
| 00001018 | | 6A2800FF | | MOV.B | @H'00FFFF3B:32,R0L | if(MI |
| 0000101e | | E807 | | AND.B | #H'07,R0L | |
| 00001020 | | A806 | | CMP.B | #H'06,R0L | |
| 00001022 | | 4702 | | BEQ | @H'1026:8 | |
| 00001024 | | 403E | | BRA | @H'1064:8 | r |
| 00001026 | | 6A2800FF | | MOV.B | @H'00FFFF39:32,R0L | if(S) |
| 0000102c | | A801 | | CMP.B | #H'01,R0L | |
| 0000102e | | 4708 | | BEQ | @H'1038:8 | |
| 00001030 | | F801 | | MOV.B | #H'01,R0L | s |
| 00001032 | | 6AA800FF | | MOV.B | R0L,@H'00FFFF39:32 | |
| 00001038 | | 6A3800FF | | BCLR.B | #5,@H'00FFFED5:32 | BCRL. |
| 00001040 | | 5528 | | BSR | @_STOP_MODE:8 | STOP_ |
| 00001042 | | 554A | | BSR | @_MASK1:8 | MASK1 |
| 00001044 | | 5574 | | BSR | @_DMAC_RUN:8 | DMAC |

**Figure 13.3   Disassembly Window**

It supports column-specific double-click actions:

- BP - Toggles standard event types at that address.

- Address - Launches the Set Address dialog box, allowing the user to enter a new address. If the address is in a source file, then that file will be opened in a new window (a current source view will be brought into focus) with the cursor set to the specified address. Finally, if the address does not correspond to a source file, then this window will scroll to that location. When an overloaded function or a class name is entered in the Set Address edit field, the Select Function dialog box opens for you to select a function.

- Code and Assembler - Launches the Assembler dialog box allowing the user to modify the instruction at that address. Note that changes to the machine code do not modify the source file, and any changes will be lost at the end of the session.

- Label - Launches the Label dialog box, allowing the user to enter a new label, or to clear or edit the name of an existing label.

- Source - Launches editor at location in source (set by optional startup parameters in Windows[®] Start menu HDI shortcut).

Within the BP column a list of currently supported standard breakpoint types can be displayed by right clicking. The currently selected standard breakpoint is shown by a check mark to the left of the menu text.

A popup menu containing the following options is available by right clicking within the window, but outside the BP column:

### 13.3.1 Copy

Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

### 13.3.2 Set Address...

Launches the Set Address dialog box, allowing the user to enter a new start address. The window will be updated so that this is the first address displayed in the top-left corner. When an overloaded function or a class name is entered, the Select Function dialog box opens for you to select a function.

### 13.3.3 Go To Cursor

Commences to execute the user program starting from the current PC address. The program will continue to run until the PC reaches the address indicated by the text cursor (not the mouse cursor) or another break condition is satisfied. Grayed if not supported by the debugging platform.

### 13.3.4 Set PC Here

Changes the value of the PC to the address indicated by the text cursor (not the mouse cursor).

### 13.3.5 Instant Watch...

Launches the Instant Watch dialog box with the name extracted from the view at the current text cursor (not mouse cursor) position. Only valid in the source column.

### 13.3.6 Add Watch

Adds the name extracted from the view at the current text cursor (not mouse cursor) position to the list of watched variables. If a Watch window is not open, then it is opened and brought to the top of the child windows. Only valid in the source column.

## 13.4 I/O Registers
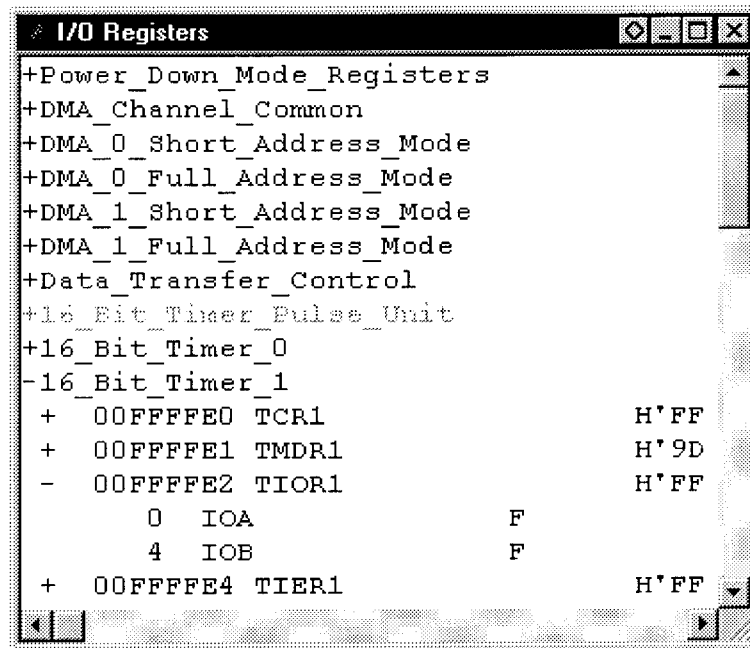


**Figure 13.4   I/O Registers Window**

Allows the user to view and control the user system hardware's on-chip peripherals. The peripherals are organized by modules, and the level of displayed detail can be changed with a '+' indicating that the information may be expanded by double-clicking on the variable name, and a '-' indicating that the information may be collapsed.

Double-click on the '+' and '-' character, or use the plus and minus keys, to expand and contract the register information.

A popup menu containing the following options is available by right clicking within the window:

### 13.4.1 Copy

Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

### 13.4.2 Edit...

Launches a dialog box to modify the selected register's contents.

### 13.4.3 Expand/Collapse

Expands/collapses the selected module.

## 13.5 Labels

```
Labels                                          [◇][_][□][X]
BP | Value        | Name                                    ▲
     H'00001000    startup_boot
     H'00001012    _main
     H'0000106A    _STOP_MODE
     H'0000108E    _MASK1
     H'000010BA    _DMAC_RUN
     H'00001180    _MASK2
     H'000011AC    _DTC_REGS
     H'0000129E    _DTC_SCIO_ACT
     H'000012E2    _DTC_SCIO_RUN
     H'00001316    _MASK3
     H'00001342    _WDT_RUN
     H'00001358    _DENDOA
    .H'0000136C    _WOVI
     H'0000139E    _TXIO
  ●  H'000013C6    _COPY_MEM
     H'00001412    __INITSCT
     H'0000145C    __D_ROM                              ▼
```

**Figure 13.5   Labels Window**

You can view symbols sorted either alphabetically (by ASCII code) or by address value by clicking on the respective column heading.

It supports column-specific double-click actions:

- BP - Toggles through the standard event types at that address.

- Value - Opens a Source window at the start of the function.

- Name - Launches the Edit Labels dialog box.

Within the BP column a list of currently supported standard breakpoint types can be displayed by right clicking. The currently selected standard breakpoint is shown by a check mark to the left of the menu text.

A popup menu containing the following options is available by right clicking within the window, but outside the BP column:

### 13.5.1 Add...

Launches the **Add Label** dialog box:
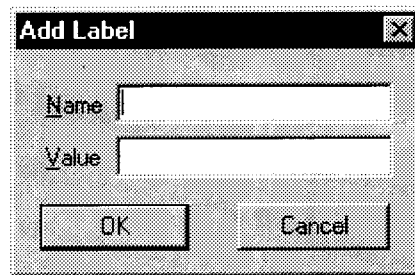
**Figure 13.6   Add Label Dialog Box**

Enter the new label name into the Name field and the corresponding value into the Value field and press **[OK]**. The Add Label dialog box closes and the label list is updated to show the new label. When an overloaded function or a class name is entered in the Value field, the Select Function dialog box opens for you to select a function. For details, refer to *section 10, Selecting Functions.*

## 13.5.2 Edit...

Launches the Edit Label dialog box:
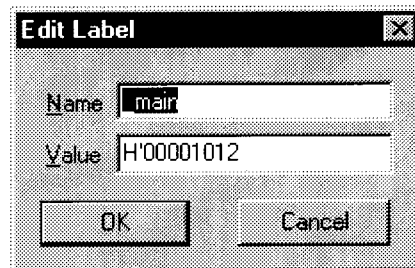


**Figure 13.7   Edit Label Dialog Box**

Edit the label name and value as required and then press **[OK]** to save the modified version in the label list. The list display is updated to show the new label details. When an overloaded function or a class name is entered in the Name field, the Select Function dialog box opens for you to select a function. For details, refer to *section 10, Selecting Functions.*

### 13.5.3 Find...

Launches the **Find Label Containing** dialog box:
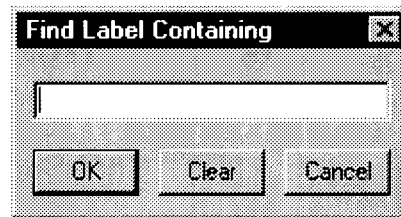


**Figure 13.8   Find Label Containing Dialog Box**

Enter all or part of the label name that you wish to find into the edit box and click **[OK]** or press **ENTER**. The dialog box closes and HDI searches the label list for a label name containing the text that you entered.

**Note**   Only the label is stored by 1024 characters of the start, therefore the label name must not overlap mutually in 1024 characters or less. Labels are case sensitive.

### 13.5.4 Delete

Deletes the currently selected label from the symbol list. Alternatively use the Delete accelerator key. A confirmation message box appears:
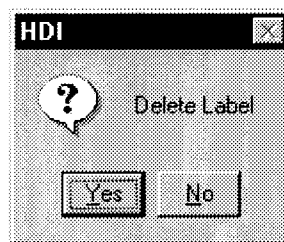


**Figure 13.9   Message Box for Confirming Label Deletion**

If you click on the **[Yes]** button the label is removed from label list and the window display is updated. If the message box is not required then do not select the Delete Label option of the Confirmation seat in the HDI Options dialog box.

### 13.5.5 Delete All

Deletes all the labels from the list. A confirmation message box appears:
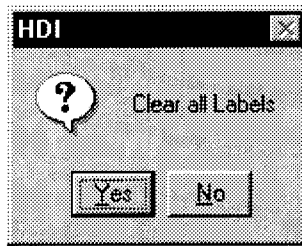
**Figure 13.10 Message Box for Confirming All Label Deletion**

If you click on the [**Yes**] button all the labels are removed from the HDI system's symbol table and the list display will be cleared. If the message box is not required then do not select the Delete All Labels option of the Confirmation seat in the HDI Options dialog box.

## 13.5.6 Load...

Merges a symbol file into HDI's current symbol table. The Load Symbols dialog box opens:
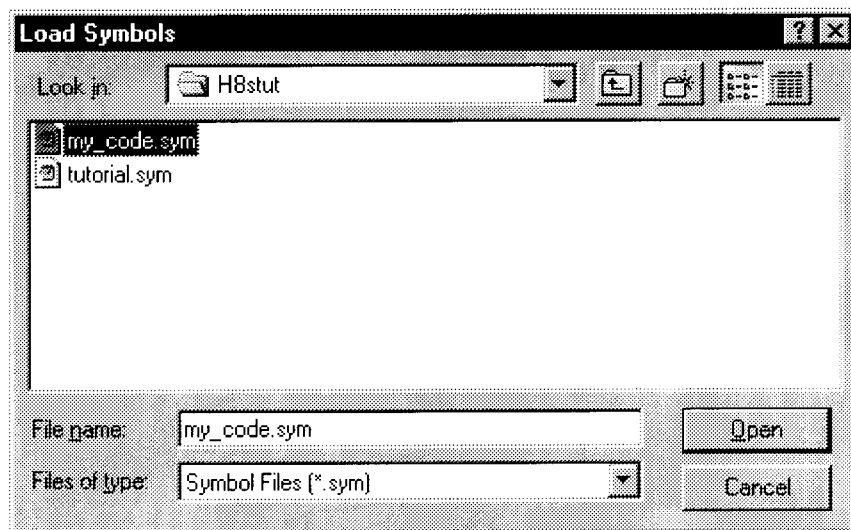


**Figure 13.11 Load Symbols Dialog Box**

The dialog box operates like a standard Windows® open file dialog box; select the file and click **[Open]** to start loading. The standard file extension for symbol files is ".sym". When the symbol loading is complete a confirmation message box may be displayed showing how many symbols have been loaded (this can be switched off in the Confirmations tab on the HDI Options dialog).

### 13.5.7 Save

Saves HDI's current symbol table to a symbol file.

### 13.5.8 Save As...

The Save Symbols dialog box operates like a standard Windows® Save File As dialog box. Enter the name for the file in the **File name** field and click **[Open]** to save HDI's current label list to a symbol file. The standard file extension for symbol files is ".sym".

See Appendix F for symbol file format.

## 13.6 Locals
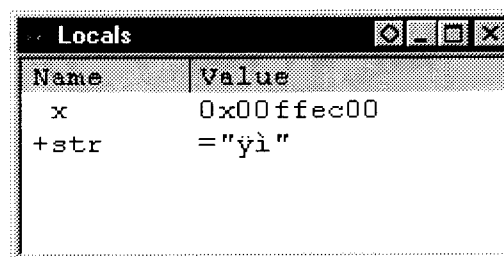


**Figure 13.12 Locals Window**

Allows the user to view and modify the values of all the local variables. The contents of this window are blank unless the current PC can be associated to a function containing local variables in the source files *via* the debugging information available in the object file.

The variables are listed with a plus indicating that the information may be expanded by double-clicking on the variable name, and a minus indicating that the information may be collapsed. Alternatively, the plus and minus keys may be used. For more information on the display of information, refer to section 8.3.2, Expanding a Watch.

A popup menu containing the following options is available by right clicking within the window:

### 13.6.1 Copy

[icon]    Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.
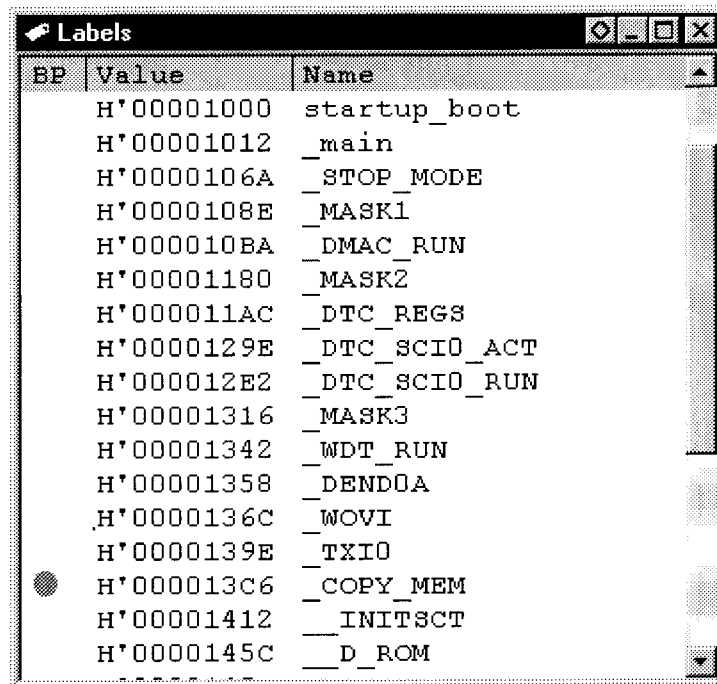
### 13.6.2 Edit Value...

Launches a dialog box to modify the selected variable's value.

### 13.6.3 Radix

Changes the radix for the selected local variable display.

## 13.7 Memory Mapping
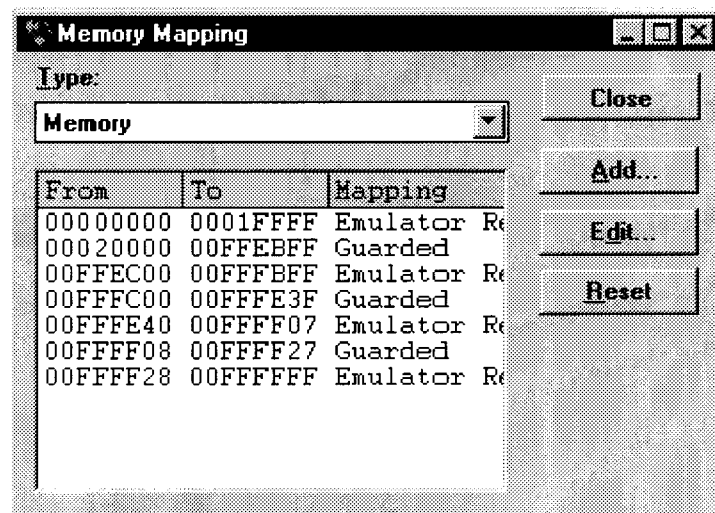


**Figure 13.13 Memory Mapping Window**

Allows the user to view and modify the debugging platform's memory map and to view its memory configuration and resources. The exact memory map configuration available will depend on the debugging platform selected, however, HDI includes a default dialog that can but used by most platforms.

A popup menu containing the following options is available by right clicking within the window.

### 13.7.1 Add...

Launches the Edit Memory Mapping dialog box allowing the user to enter the details of a new memory area to add to the map. Grayed if the debugging platform does not support editing of its maps.

## 13.7.2 Edit...

Launches the Edit Memory Mapping dialog box allowing the user to modify the details of the currently selected memory map. Grayed if the debugging platform does not support editing of its maps.
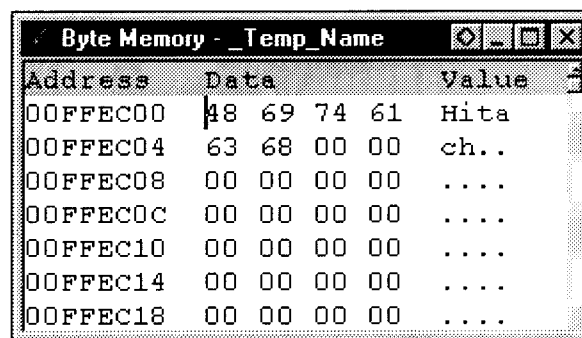
## 13.7.3 Reset

Returns the map information to the debugging platform's default values. Grayed if the debugging platform does not support editing of its maps.

## 13.7.4 Help

Launches the help file.

## *13.8 Memory*



| Byte Memory - _Temp_Name |  |  |  |  |  |
|---|---|---|---|---|---|
| Address | Data |  |  |  | Value |
| 00FFEC00 | 48 | 69 | 74 | 61 | Hita |
| 00FFEC04 | 63 | 68 | 00 | 00 | ch.. |
| 00FFEC08 | 00 | 00 | 00 | 00 | .... |
| 00FFEC0C | 00 | 00 | 00 | 00 | .... |
| 00FFEC10 | 00 | 00 | 00 | 00 | .... |
| 00FFEC14 | 00 | 00 | 00 | 00 | .... |
| 00FFEC18 | 00 | 00 | 00 | 00 | .... |

**Figure 13.14 Memory Window**

Allows the user to view and modify the contents of the debugging platform's memory. Memory may be viewed in ASCII, byte, word, long word, single-precision floating-point, and double-precision floating-point formats, and the title bar indicates the current view style and the address shown as the offset from the previous label (symbol).

The contents of memory may be edited by either typing at the current cursor position, or by double-clicking on a data item. The latter will launch the Edit dialog box, allowing the user to enter a new value using a complex expression. If the data at that address cannot be modified (i.e. within ROM or guarded memory) then the message "Invalid address value" is displayed.

Double-clicking within the Address column will launch the Set Address dialog box, allowing the user to enter an address. Clicking the [OK] button will update the window so that the address entered in the Set Address dialog box is the first address displayed in the top-left corner.

A popup menu containing the following options is available by right clicking within the window:

### 13.8.1 Set Address...

Launches the Set Address dialog box, allowing the user to enter a new start address. The window will be updated so that this is the first address displayed in the top-left corner. When an overloaded function or a class name is entered, the Select Function dialog box opens for you to select a function.

### 13.8.2 Load...

Launches the Load Memory dialog box, allowing the user to load to the debugging platform's memory from an S-Record file (*.mot) without deleting the current debug information. The offset field may be used to move the address values specified in the file to a different set of addresses. The optional verify flag can be used to check that the information has been downloaded correctly.

### 13.8.3 Save...

Launches the Save Memory As dialog box, allowing the user to save a block of the debugging platform's memory to an S-Record file (*.mot). The start and end fields may be set similarly to the Search option(see *section 13.8.8, Search...*).

### 13.8.4 Test...

Launches the Test Memory dialog box, allowing the user to validate a block of memory within the debugging platform. The details of the test depend on the debugging platform. The start and end fields may be set similarly to the Search option(see *section 13.8.8, Search...*).

### 13.8.5 Fill...

Launches the Fill Memory dialog box, allowing the user to fill a block of the debugging platform's memory with a specified value. The start and end fields may be set similarly to the Search option(see *section 13.8.8, Search...*).

### 13.8.6 Copy...

Launches the Move Memory dialog box, allowing the user to copy a block of memory within the debugging platform to another location within the same memory space. The blocks may overlap. The start and end fields may be set similarly to the Search option(see *section 13.8.8, Search...*).

### 13.8.7 Compare...

Launches the Compare Memory dialog box, allowing the user to select a start and an end address in the memory area, to check against another area in memory. If a block of memory is highlighted in a Memory window, these will be automatically entered as the start and end addresses when the dialog box is displayed.

Similar to Verify memory, but compares two blocks in memory.
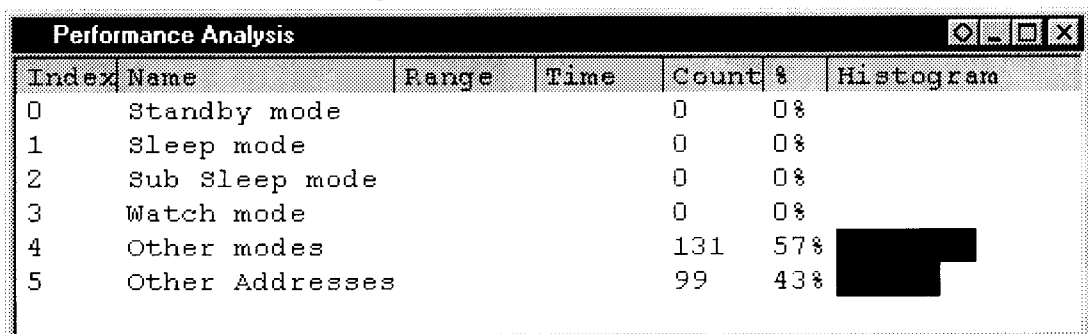
### 13.8.8 Search...

Launches the Search Memory dialog box, allowing the user to search a block of the debugging platform's memory for a specified data value. If a block of memory is highlighted, the start and end fields in the dialog box will be filled automatically with the start and end addresses corresponding to the highlighted block, respectively.

### 13.8.9 ASCII/Byte/Word/Long/Single Float/Double Float

A check mark next to these six options indicates the current view format. The user may select a different option to change to that format.

## 13.9 Performance Analysis

| Index | Name | Range | Time | Count | % | Histogram |
|-------|------|-------|------|-------|---|-----------|
| 0 | Standby mode | | | 0 | 0% | |
| 1 | Sleep mode | | | 0 | 0% | |
| 2 | Sub Sleep mode | | | 0 | 0% | |
| 3 | Watch mode | | | 0 | 0% | |
| 4 | Other modes | | | 131 | 57% | |
| 5 | Other Addresses | | | 99 | 43% | |

**Figure 13.15 Performance Analysis Window**

Allows the user to view and control the performance analysis data. The items displayed as default cannot be deleted or modified by the user. The display contents and operation depend on the debugging platform. See the supplied *Debugging Platform User's Manual* for more information. A popup menu containing the following options is available by right clicking within the view area:

### 13.9.1 Add Range...

Launches the Add PA Range dialog box, allowing the user to add a new user range based on either source lines or an address range. The name of the range can be edited.

### 13.9.2 Edit Range...

Only enabled when the highlighting bar is on a user-defined range. Launches the Edit PA Range dialog box, allowing the user to modify the range's settings.

### 13.9.3 Delete Range

Only enabled when the highlighting bar is on a user-defined range. Deletes the range and immediately recalculates the data for the other ranges.

### 13.9.4 Reset Counts/Times

Clears the current performance analysis data.
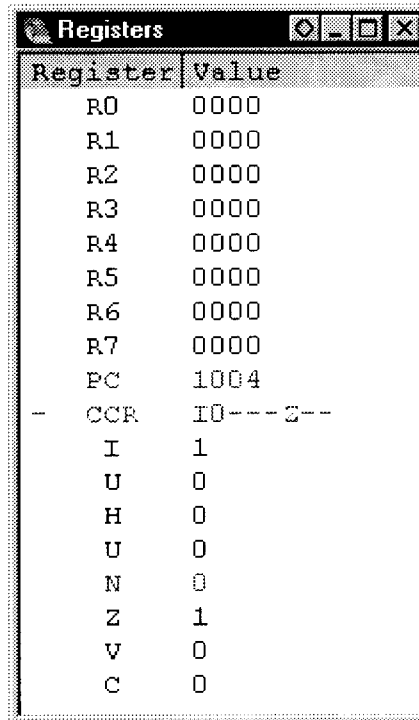
### 13.9.5 Delete All Ranges

Deletes all the current user-defined ranges, and clears the performance analysis data.

### 13.9.6 Analysis Enabled

Toggles the collection of performance analysis data. When performance analysis is active, a check mark is shown to the left of the text.

# 13.10 Registers



**Figure 13.16 Registers Window**

Allows the user to view and modify the current register values.

A popup menu containing the following options is available by right clicking within the window:

## 13.10.1 Copy

Only available if a block of text is highlighted. This copies the selected text into the Windows® clipboard, allowing it to be pasted into other applications.

## 13.10.2 Edit...

Launches the Register dialog box, allowing the user to set the value of the register indicated by the text cursor (not mouse cursor).

## 13.10.3 Toggle Bit

Only available if the text cursor is placed on a bit-field, e.g. a flag within a status register. Changes the current state of the bit to its other state, e.g. a set overflow flag can be cleared.

## 13.11 Source

The Source window can be used to view any source file that was included within the object file's debug information - this may be C/C++ and assembly language.



```
Tutorial.c                                                _ □ ☒
Line    Address    BP    Label      Source                            ▲
28      00001012         _main      void main(void)
29                                  {                                 ─┘
30
31      00001018                       if(MDCR.BIT.MDS!=0x6
32                                     /*      printf("Sele
33      00001024  ●                       return;
34                                     }
35      00001026                       if(SYSCR.BYTE!=0x01)
36      00001030                          SYSCR.BYTE=0x1;
37
38      00001038                       BCRL.BIT.EAE = 0;
39
40      00001040                       STOP_MODE();              ▼
◄                                                              ►
```
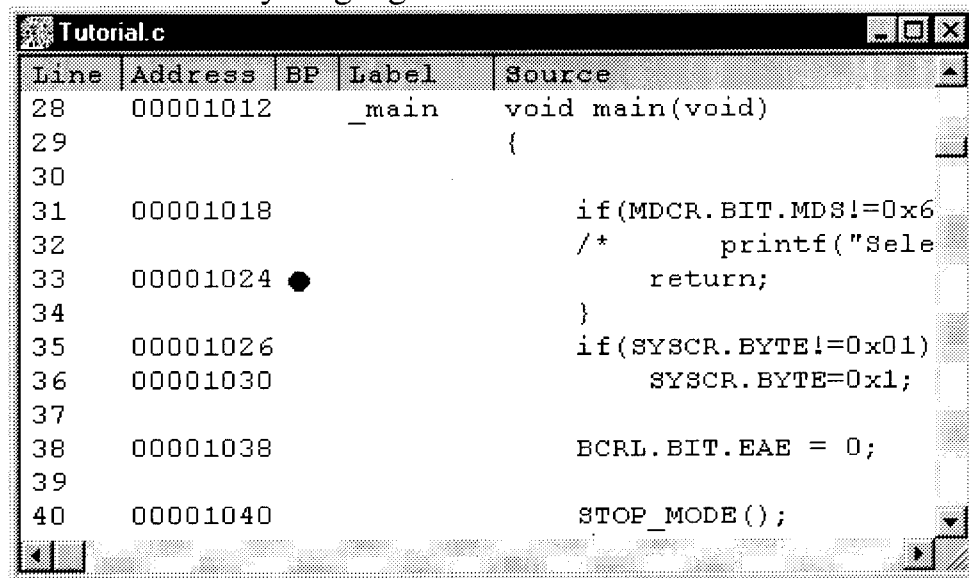
**Figure 13.17 Source View**

It supports column-specific double-click actions:

- BP - Sets/clears a program (PC) breakpoint at that address.

- Address - Launches the Set Address dialog box, allowing the user to enter a new address. If the address is within the range of this file, then the view will scroll such that the cursor can be positioned correctly. If the address is in a different source file, then that file will be opened in a new window with the cursor set to the specified address. Finally, if the address does not correspond to a source file, then a new Disassembly window will be opened. When an overloaded function or a class name is entered, the Select Function dialog box opens for you to select a function.

- Label - Launches the Label dialog box, allowing the user to enter a new label and edit the name of an existing label.

- Line - Launches the Set Line dialog box, allowing the user to go directly to a line in the source file.

- Source - Opens the source file in the editor (specified in the Startup menu HDI shortcut) at this source line.

Within the BP column a list of currently supported standard breakpoint types can be displayed by right clicking. The currently selected standard breakpoint is shown by a check mark to the left of the menu text.

A popup menu containing the following options is available by right clicking in any of the other columns within the window:

### 13.11.1 Copy

Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

### 13.11.2 Find...

Launches the Find dialog box, allowing the user to search the source file for a string.

### 13.11.3 Set Address...

Launches the Set Address dialog box, allowing the user to enter a new start address. The window will be updated so that this is the first address displayed in the top-left corner. When an overloaded function or a class name is entered, the Select Function dialog box opens for you to select a function.

### 13.11.4 Set Line...

Launches the Set Line dialog box, allowing the user to display and move the text cursor (not the mouse cursor) to a specific line.

### 13.11.5 Go To Cursor

Commences to execute the user program starting from the current PC address. The program will continue to run until the PC reaches the address indicated by the text cursor (not the mouse cursor) or another break condition is satisfied. Grayed if not supported by the debugging platform.

### 13.11.6 Set PC Here

Changes the value of the PC to the address indicated by the text cursor (not the mouse cursor).

### 13.11.7 Instant Watch...

Launches the Instant Watch dialog box with the name extracted from the view at the current text cursor (not mouse cursor) position.
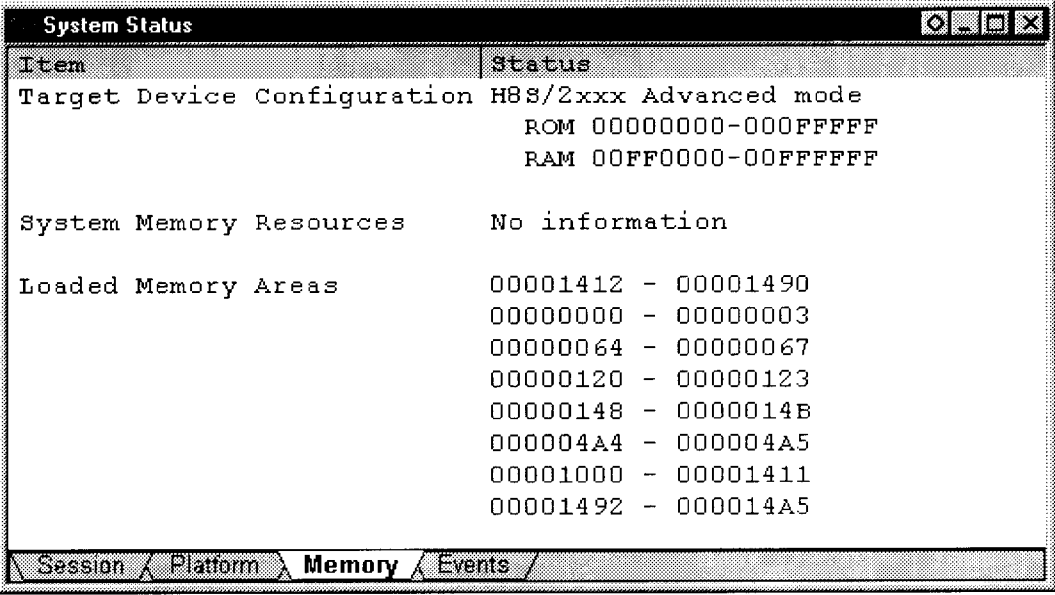
### 13.11.8 Add Watch

Adds the name extracted from the view at the current text cursor (not mouse cursor) position to the list of watched variables. If the Watch window is not open, then it is opened and brought to the top of the child windows.

### 13.11.9 Go To Disassembly

Opens a Disassembly view at the address matching the current source line.

## 13.12 System Status

```
┌─────────────────────────────────────────────────────────────┐
│  System Status                                    [O][_][□][X]│
├─────────────────────────────────────────────────────────────┤
│ Item                      │Status                            │
│ Target Device Configuration H8S/2xxx Advanced mode           │
│                            ROM 00000000-000FFFFF             │
│                            RAM 00FF0000-00FFFFFF             │
│                                                              │
│ System Memory Resources    No information                    │
│                                                              │
│ Loaded Memory Areas        00001412 - 00001490              │
│                            00000000 - 00000003              │
│                            00000064 - 00000067              │
│                            00000120 - 00000123              │
│                            00000148 - 0000014B              │
│                            000004A4 - 000004A5              │
│                            00001000 - 00001411              │
│                            00001492 - 000014A5              │
├─────────────────────────────────────────────────────────────┤
│ \ Session /\ Platform \ Memory / Events /                    │
└─────────────────────────────────────────────────────────────┘
```

**Figure 13.18 System Status Window**

Allows the user to view the current status of the debugging platform. See the supplied *Debugging Platform User's Manual* for more information.

The System Status window is split into four panes:

1. Session - contains information about the current session including the connected debugging platform and the names of loaded files.

2. Platform - contains information about the current status of the debugging platform, typically including CPU type and mode; run status; and timing information.

3. Memory - contains information about the current memory status including the memory mapping resources and the areas used by the currently loaded object file.

4. Events - contains information about the current event (breakpoint) status, including resource information.

A popup menu containing the following options is available by right clicking within the window:

### 13.12.1 Update

Updates the displayed data.

### 13.12.2 Copy

Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

## 13.13 Trace



**Figure 13.19 Trace Window**

Allows the user to view the sequence of instructions leading up to the debugging platform's current status. The exact view will depend on the selected debugging platform - in the above picture the target returns only a single column, but uses spaces to columnize the data. Other platforms may return multiple columns with different headings.

Double-clicking on a row will open the Source or Disassembly view for the address.

When mouse's right button is clicked in the window, the pop-up menu is displayed. The following options are included in this menu.

## 13.13.1 Find...

Launches the Trace Search dialog box, allowing the user to search the current trace buffer for a specific trace record.

## 13.13.2 Find Next

If a find operation is successful, and the item found is non-unique, then this will move to the next similar item.

## 13.13.3 Filter...

Launches the Filter Trace dialog box, allowing the user to mask out all unnecessary trace entries.

## 13.13.4 Acquisition...

Launches the Trace Acquisition dialog box, allowing the user to define the area of user program to be traced. This is useful to focus tracing on problem areas.

## 13.13.5 Halt

Stops tracing data and updates the trace information without stopping execution of the user program.

## 13.13.6 Restart

Starts tracing data.

## 13.13.7 Snapshot

Updates the trace information to show the debugging platform's current status without stopping user program execution.

## 13.13.8 Clear

Empties the trace buffer in the debugging platform. If more than one trace window is open, all Trace windows will be cleared as they all access the same buffer.

### 13.13.9 Save...

Launches the Save As file dialog box, allowing the user to save the contents of the trace buffer as a text file. It is possible to define a numeric range based on the Cycle number or to save the complete buffer (saving the complete buffer may take several minutes). Note that this file cannot be reloaded into the trace buffer.

### 13.13.10 View Source

Opens a Source or Disassembly window for the address.

### 13.13.11 Trim Source

Removes white space from the left side of the source.

## 13.14 Watch
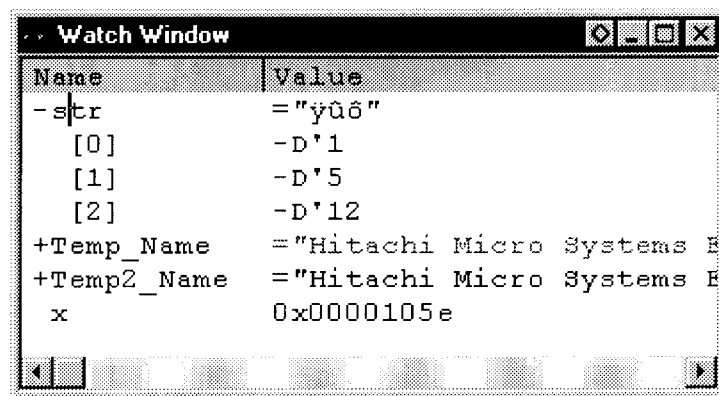


**Figure 13.20 Watch Window**

Allows the user to view and modify C/C++-source level variables. The contents of this window are blank unless the current user program can be associated to a C/C++-source file *via* the debugging information available in the absolute file (*.abs).

The variables are listed with a plus indicating that the information may be expanded by double-clicking on the variable name, and a minus indicating that the information may be collapsed. Alternatively, the plus and minus keys may be used.

A popup menu containing the following options is available by right clicking within the windows:

### 13.14.1 Copy

Only available if a block of text is highlighted. This copies the highlighted text into the Windows® clipboard, allowing it to be pasted into other applications.

### 13.14.2 Delete

Removes the variable indicated by the text cursor (not the mouse cursor) from the Watch window.

### 13.14.3 Delete All

Removes all the variables from the Watch window.

### 13.14.4 Add Watch...

Launches the Add Watch dialog box, allowing the user to enter a variable or expression to be watched.

### 13.14.5 Edit Value...

Launches the Edit Watch dialog box, allowing the user to change the variable's value. Particular care should be taken when the value of a pointer is changed as it may no longer point to valid data.

### 13.14.6 Radix

Modifies the radix for the selected watch item display.

# Appendix A - System Modules

The following section describes the architecture of the HDI debugging system.
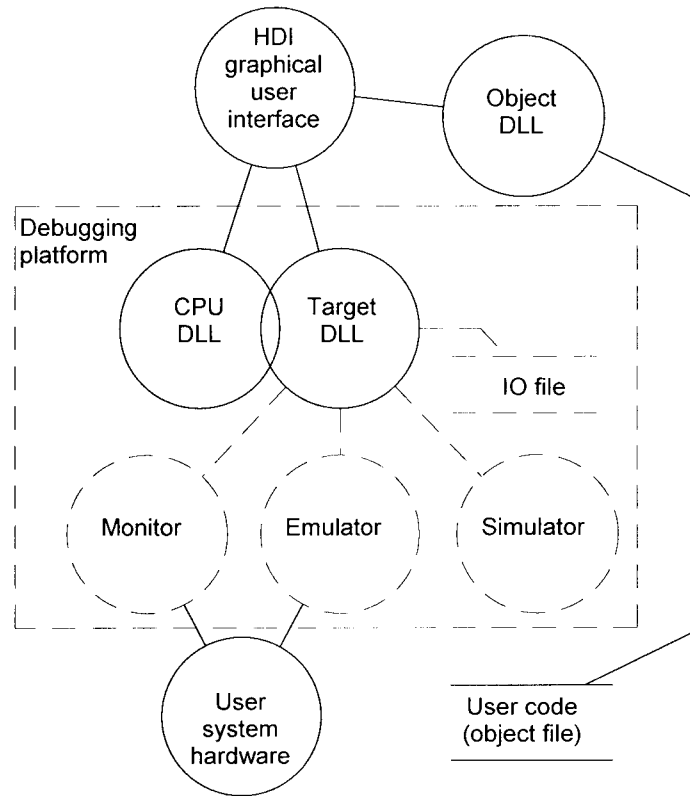
Figure A.1    HDI System Modules

In normal operation, the user program will be placed directly into the target hardware (for example as an EPROM). HDI uses this information to provide a Windows®-based debugging system.

To decrease the learning curve when swapping between different debugging platforms and/or user system hardware, HDI provides a single unified interface (the GUI) and a family of target specific modules. Normally, the user will only interact with the standard GUI - once the appropriate target module has been selected, the rest of the system configures itself automatically by loading the appropriate modules.

## Graphical User Interface

This is the main HDI.EXE program that runs under Windows®. It uses familiar Windows® operations, with menus and windows to give a user-friendly view into the debugging system. The GUI is the only contact between the user and the rest of the system, it processes commands and provides the required information about the user program. It also provides the interface between the module DLLs and the host file system, i.e., the PC.

## Object DLL

When creating the user program, a compiler will generate an *absolute object file*. This file contains the actual machine code and data that the microcomputer processes to execute the functions making up the target application. In order to debug the user program as original source code, the compiler must provide more information to the debugger. For this reason, nearly all compilers have a debug option that puts all the information necessary for debugging your source code into the absolute file, which is usually called a *debug object file*.

The object DLL extracts this information from the object file for display to the user. Since the format of data is compiler dependent, more than one object DLL may be present in the HDI directory - HDI will try each in turn until it finds one that can understand the object file's format.

## CPU DLL

The CPU DLL module contains information specific to the target microcomputer. For example, it contains the number and types of registers available to the microcomputer. It also translates the raw machine code in the target into more familiar assembly-language mnemonics displayed in the Source window, and vice versa.

## Target DLL

The target DLL informs HDI about the debugging platform's capabilities and selects the correct CPU DLL. Since some capabilities of the debugging platform cannot be generic (for example, target configuration), the target DLL also includes extensions to the standard GUI to provide the user with access to these capabilities.

For a detailed description of the features available using your target DLL, refer to the supplied *Debugging Platform User's Manual*.

# Appendix B - Command Line Interface

## HDI Built-In Commands

The following is a list of the standard HDI built in commands.

### !(COMMENT)

**Abbreviation:** none

**Description:**
Allows a comment to be entered, useful for documenting batch & log files.

**Syntax:**
! <text>

| Parameter | Type | Description |
|-----------|------|-------------|
| <text> | Text | Output text |

**Example:**

| | |
|---|---|
| ! Start of test routine | Outputs comment 'Start of test routine' into the Command Line window (and to the log file, if logging is active). |

### ACCESS

**Abbreviation:** AC

**Description:**
Sets or displays the illegal access handling.

**Syntax:**
access [<state>]

| Parameter | Type | Description |
|-----------|------|-------------|
| none | | Displays the current setting |
| <state> | Keyword | Action to be taken on illegal access |
| | break | Break emulation (default setting) |
| | none | No action |

Illegal accesses are writes to protected areas during RUN, writes to internal ROM, or any access to an unmapped area of memory.

**Example:**

| | |
|---|---|
| ACCESS break | Break on guarded/write-protected access. (default setting). |
| AC | Displays current illegal access handing. |
| AC none | Sets no action on an illegal access. |

## ANALYSIS

**Abbreviation:** AN

**Description:**
Enables/disables performance analysis. Counts are not automatically reset before running.

**Syntax:**
an [<state>]

| Parameter | Type | Description |
|---|---|---|
| none | | Displays the analysis state |
| <state> | Keyword | Enables/disables analysis |
| | enable | Enables analysis |
| | disable | Disables analysis |
| | reset | Resets analysis counts |

**Example:**

| | |
|---|---|
| ANALYSIS | Displays analysis state. |
| AN enable | Enables analysis. |
| AN disable | Disables analysis. |
| AN reset | Resets analysis counts. |

## ANALYSIS_RANGE

**Abbreviation:** AR

**Description:**
Sets performance analysis range, or displays performance analysis ranges if no parameters are specified. The syntax depends on the debugging platform. See the supplied *Debugging Platform User's Manual*.

**Syntax:**
ar [<start> <end> [<name>]]

| Parameter | Type | Description |
|---|---|---|
| none | | Displays all analysis ranges |
| <start> | Numeric | Start address of range |
| <end> | Numeric | End address of range |
| <name> | String | User range description |

**Example:**

| | |
|---|---|
| ANALYSIS_RANGE H'0 H'100 | Defines a performance analysis range from address H'0 to H'100. |
| AR H'1000 H'3FFF | Defines a performance analysis range from H'1000 to H'3FFF. |
| AR | Displays the current analysis ranges set. |

## ANALYSIS_RANGE_DELETE

**Abbreviation:** AD

**Description:**
Deletes the specified performance analysis range, or all ranges if no parameters are specified (it does **not** ask for confirmation).

**Syntax:**
ad [<index>]

| Parameter | Type | Description |
|-----------|------|-------------|
| none | | Deletes all analysis ranges |
| <index> | Numeric | Index number of range to delete |

**Example:**

| ANALYSIS_RANGE _DELETE 6 | Deletes the analysis range with index number 6 from the system. |
|---|---|
| AD | Deletes all user defined analysis ranges. |

## ASSEMBLE

**Abbreviation:** AS

**Description:**
Assembles instructions into memory. In assembly mode, '.' exits, '^' steps back a byte, the ENTER key steps forward a byte.

**Syntax:**
as <address>

| Parameter | Type | Description |
|-----------|------|-------------|
| <address> | Numeric | Address at which to start assembling |

**Example:**

| AS H'1000 | Starts assembling from H'1000. |
|---|---|

## ASSERT

**Abbreviation:** none

**Description:**
Checks if an expression is true or false. It can be used to terminate the batch file when the expression is false. If the expression is false, an error is returned. This command can be used to write test harnesses for subroutines.

**Syntax:**

assert &lt;expression&gt;

| Parameter | Type | Description |
|---|---|---|
| &lt;expression&gt; | Expression | Expression to be checked |

**Example:**

| | |
|---|---|
| ASSERT #R0 == 0x100 | Returns an error if R0 does not contain 0x100. |

# DISASSEMBLE

**Abbreviation:** DA

**Description:**

Disassembles memory contents to assembly-language code. Disassembly display is fully symbolic.

**Syntax:**

da &lt;address&gt; [&lt;length&gt;]

| Parameter | Type | Description |
|---|---|---|
| &lt;address&gt; | Numeric | Start address |
| &lt;length&gt; | Numeric | Number of instructions (optional, default = 16) |

**Example:**

| | |
|---|---|
| DISASSEMBLE H'100 5 | Disassembles 5 lines of code starting at H'100. |
| DA H'3E00 20 | Disassembles 20 lines of code starting at H'3E00. |

# ERASE

**Abbreviation:** ER

**Description:**

Clears the Command Line window

**Syntax:**

er

| Parameter | Type | Description |
|---|---|---|
| none | | Clears the Command Line window |

**Example:**

| | |
|---|---|
| ER | Clears the Command Line window. |

# EVALUATE

**Abbreviation:** EV

**Description:**
Provides a calculator function, evaluating simple and complex expressions, with parentheses, mixed radices, and symbols. All operators have the same precedence but parentheses may be used to change the order of evaluation. The operators have the same meaning as in C/C++. Expressions can also be used in any command where a number is required, but they cannot contain spaces since these are used to separate parameters. Register names may be used, but must always be prefixed by the '#' character. The result is displayed in hexadecimal, decimal, octal, or binary.

**Note:** It is not possible to evaluate expressions containing C/C++ variable, structure, or array references.

**Syntax:**
ev <expression>

| Parameter | Type | Description |
|---|---|---|
| <expression> | Expression | Expression to be evaluated |

Valid operators:

| && logical AND | \|\| logical OR | << left arithmetic shift | >> right arithmetic shift |
|---|---|---|---|
| + addition | - subtraction | * multiplication | / division |
| % modulo | \| bitwise OR | & bitwise AND | ~ bitwise NOT |
| ^ bitwise exclusive OR | ! logical NOT | == equal to | != unequal to |
| > greater than | < less than | >= greater than or equal to | <= less than or equal to |

**Example:**

EV H'123 + (D'73 | B'10)     Result: H'16E D'366 O'556
                                          B'00000000000000000000000101101110

EV #R2H * #R2L               Result: H'121 D'289 O'441
                                          B'00000000000000000000000100100001

# FILE_LOAD

**Abbreviation:** FL

**Description:**
Loads an object code file to memory with, or without, the specified offset. Existing symbols are cleared, but the new ones will override any existing ones with the same names. If an offset is specified this will be added to the symbols. The file extension default is **.MOT**.

**Syntax:**
fl <filename> [<offset>] [<state>]

| Parameter | Type | Description |
|---|---|---|
| <filename> | String | File name |
| <offset> | Numeric | Offset to be added to load address (optional, default = 0) |
| <state> | Keyword | Verify flag (optional, default = V) |
| | V | Verify |
| | N | No verify |

**Example:**

| | |
|---|---|
| FILE_LOAD A:\\BINARY\\TESTFILE.A22 | Loads S-Record file "testfile.a22". |
| FL ANOTHER.MOT H'200 | Loads Motorola S-Record file "another.mot" with an offset of H'200 bytes. |

# FILE_SAVE

**Abbreviation:** FS

**Description:**
Saves memory area to a file. The data is saved in Motorola S-Record format. The user is warned if about to overwrite an existing file.
The file extension default is **.MOT**. Symbols are **not** automatically saved.

**Syntax:**
fs <filename> <start> <end>

| Parameter | Type | Description |
|---|---|---|
| <filename> | String | File name |
| <start> | Numeric | Start address |
| <end> | Numeric | End address |

**Example:**

| | |
|---|---|
| FILE_SAVE TESTFILE.MOT H'0 H'2013 | Saves address range H'0-H'2013 as Motorola S-Record file "TESTFILE.MOT". |
| FS D:\\USER\\ANOTHER.A22 H'4000 H'4FFF | Saves address range H'4000-H'4FFF as S-Record format file "ANOTHER.A22". |

## FILE_VERIFY

**Abbreviation: FV**

**Description:**
Verifies file contents against memory. The file data must be in a Motorola S-Record format. The file extension default is .**MOT**.

**Syntax:**
  fv <filename> [<offset>]

| Parameter | Type | Description |
|---|---|---|
| <filename> | String | File name |
| <offset> | Numeric | Offset to be added to file address (optional, default = 0) |

**Example:**

| | |
|---|---|
| FILE_VERIFY<br>A:\\BINARY\\TEST.A22 | Verifies S-Record file "TEST.A22" against memory. |
| FV ANOTHER 200 | Verifies Motorola S-Record file "ANOTHER.MOT" against memory with an offset of H'200 bytes. |

## GO

**Abbreviation: GO**

**Description:**
Runs object code (the user program).
While the user program is running, only the Performance Analysis window is updated.
When execution stops, the register values and reason for break are displayed.

**Syntax:**
  go [<state>] [<address>]

| Parameter | Type | Description |
|---|---|---|
| <state> | Keyword | Specifies whether or not to continue command processing during program execution (optional, default = wait) |
| | wait | Causes command processing to wait until program stops |
| | continue | Continues command processing during execution |
| <address> | Numeric | Start address for PC (optional, default = PC value) |

Wait is the default and this causes command processing to wait until program stops running.

Continue allows you to continue to enter commands (but they may not work depending on the facilities of the debugging platform).

**Example:**

| | |
|---|---|
| GO | Runs the user program from the current PC value (does not continue command processing). |
| GO CONTINUE H'1000 | Runs the user program from H'1000 (continues command processing). |

# GO_RESET

**Abbreviation:** GR

**Description:**

Runs the user program starting at the address specified in the reset vector.

While the user program is running, only the Performance Analysis window is updated.

**Syntax:**

gr [<state>]

| Parameter | Type | Description |
|---|---|---|
| <state> | Keyword | Specifies whether or not to continue command processing during program execution (optional, default = wait) |
| | wait | Causes command processing to wait until program stops |
| | continue | Continues command processing during execution |

Wait is the default and this causes command processing to wait until program stops running.

Continue allows you to continue to enter commands (but they may not work depending on the facilities of the debugging platform)

**Example:**

| | |
|---|---|
| GR | Runs the user program starting at the address specified in the reset vector (does not continue command processing). |

## GO_TILL

**Abbreviation:** GT

**Description:**
Runs the debugging platform program from the current PC with temporary breakpoints. This command takes multiple addresses as parameters, and these are used to set temporary PC breakpoints (these breakpoints only exist for the duration of the command).

**Syntax:**
    gt [<state>] <address>...

| Parameter | Type | Description |
|---|---|---|
| <state> | Keyword | Specifies whether or not to continue command processing during program execution (optional, default = wait) |
| | wait | Causes command processing to wait until program stops |
| | continue | Continues command processing during execution |
| <address>... | Numeric | Temporary breakpoint address (list) |

Wait is the default and this causes command processing to wait until program stops running
Continue allows you to continue to enter commands (but they may not work depending on the facilities of the debugging platform)

**Example:**

| | |
|---|---|
| GO_TILL H'1000 | Runs emulation until the PC reaches address H'1000. |

## HALT

**Abbreviation:** HA

**Description:**
Halts the user program (can be used after a "go continue" command).

**Syntax:**
    ha

| Parameter | Type | Description |
|---|---|---|
| none | | Halts the user program |

**Example:**

| | |
|---|---|
| HA | Halts the user program. |

## HELP

**Abbreviation:** HE

**Description:**
Opens a window displaying the help file.
For context sensitive help, the F1 key should be pressed. Help on a particular command can be retrieved by entering HELP or HE followed by the command name at the command line.

**Syntax:**
  he [<command>]

| Parameter | Type | Description |
|---|---|---|
| none | | Displays the contents of the help |
| <command> | String | Displays the help for the specified command |

**Example:**

| | |
|---|---|
| HE | Displays the contents of the help. |
| HE GO | Displays help for the GO command. |

## INITIALISE

**Abbreviation:** IN

**Description:**
Initializes HDI (including debugging platform) and the user system (as if you had reselected the target DLL). All breakpoints, memory mapping, etc. are reset.

**Syntax:**
  in

| Parameter | Type | Description |
|---|---|---|
| none | | Initialized HDI |

**Example:**

| | |
|---|---|
| IN | Initializes HDI. |

## INTERRUPTS

**Abbreviation:**  IR

**Description:**
Enables or disables interrupts or sets the interrupt priority level of the CPU. This command operates by changing the CPU status register (SR or CCR).

**Note:**  Some debugging platforms do not support this command.

**Syntax:**
   ir [<state>|<level>]

| Parameter | Type | Description |
|-----------|------|-------------|
| none | | Displays the current interrupt state |
| <state> | Keyword | Enables or disables interrupts |
| | enable | Enables interrupts |
| | disable | Disables interrupts |
| <level> | Numeric | Sets the interrupt priority level |

**Example:**

| | |
|---|---|
| IR | Displays the CPU interrupt status |
| IR ENABLE | Enables all interrupts |
| IR DISABLE | Disables all interrupts (except NMI). |
| IR 5 | Sets interrupt priority level 5. |

# LOG

**Abbreviation:** LO

**Description:**
Controls logging of command output to file. If no parameters are specified, logging status is displayed. If an existing file is specified, you will be warned; if you answer 'No', data will be appended to the existing file, otherwise the file will be truncated. Logging is only supported for the command line interface.

**Syntax:**
   lo [<state>|<filename>]

| Parameter | Type | Description |
|-----------|------|-------------|
| none | | Displays logging status |
| <state> | Keyword | Starts or suspends logging |
| | + | Starts logging |
| | - | Suspends logging |
| <filename> | Numeric | Specifies the logging output file |

**Example:**

| | |
|---|---|
| LOG TEST | Logs the output to the list box in file TEST. |
| LO - | Suspends logging. |
| LOG + | Resumes logging. |
| LOG | Displays logging status |

## MAP_DISPLAY

**Abbreviation:** MA

**Description:**
Displays memory mapping.

**Syntax:**
ma

| Parameter | Type | Description |
|---|---|---|
| none | | Displays the current memory mapping |

**Example:**

| | |
|---|---|
| MA | Displays the current memory mapping. |

## MEMORY_DISPLAY

**Abbreviation:** MD

**Description:**
Displays memory contents.

**Syntax:**
md <address> [<length>] [<mode>]

| Parameter | Type | Description |
|---|---|---|
| <address> | Numeric | Start address |
| <length> | Numeric | Length (optional, default = H'100 bytes) |
| <mode> | Keyword | Display format (optional, default = byte) |
| | byte | Displays as bytes |
| | word | Displays as words (2 bytes) |
| | long | Displays as long words (4 bytes) |
| | ascii | Displays as ASCII |
| | single | Displays as single-precision floating-point |
| | double | Displays as double-precision floating-point |

**Example:**

| | |
|---|---|
| MEMORY_DISPLAY H'C000 H'100 WORD | Dumps H'100 bytes of memory starting at H'C000 in the word format. |
| MEMORY_DISPLAY H'1000 H'FF | Dumps H'FF bytes of memory starting at H'1000 in the byte format |

# MEMORY_EDIT

**Abbreviation:** ME

**Description:**
Allows memory contents to be modified. When editing memory the current location may be modified in a similar way to that described in the **ASSEMBLE** command description.
When editing, '.' exits edit mode, '^' goes back a unit, and blank line goes forward without change.

**Syntax:**
    me <address> [<mode>] [<state>]

| Parameter | Type | Description |
|-----------|------|-------------|
| <address> | Numeric | Address to edit |
| <mode> | Keyword | Format (optional, default = byte) |
| | byte | Edits as bytes |
| | word | Edits as words |
| | long | Edits as long words |
| | ascii | Edits as ASCII |
| | single | Edits as single-precision floating-point |
| | double | Edits as double-precision floating-point |
| <state> | Keyword | Verify flag (optional, default = V) |
| | V | Verify |
| | N | No verify |

**Example:**

| | |
|---|---|
| ME H'1000 WORD | Modifies memory contents as words starting from H'1000 (with verification) |

# MEMORY_FILL

**Abbreviation:** MF

**Description:**
Fills an area of memory.

**Syntax:**
mf <start> <end> <data> [<mode>] [<state>]

| Parameter | Type | Description |
| --- | --- | --- |
| <start> | Numeric | Start address |
| <end> | Numeric | End address |
| <data> | Numeric | Data value |
| <mode> | Keyword | Data size (optional, default = byte) |
| | byte | Byte |
| | word | Word |
| | long | Long word |
| | single | Single-precision floating-point |
| | double | Double-precision floating-point |
| <state> | Keyword | Verify flag (optional, default = V) |
| | V | Verify |
| | N | No verify |

**Example:**

| | |
| --- | --- |
| MEMORY_FILL H'C000 H'C0FF H'55AA WORD | Fills memory from H'C000 to H'C0FF with word data H'55AA. |
| MF H'5000 H'7FFF H'21 | Fills memory from H'5000 to H'7FFF with data H'21. |

# MEMORY_MOVE

**Abbreviation:** MV

**Description:**
Moves memory.

**Syntax:**
mv <start> <end> <dest> [<state>]

| Parameter | Type | Description |
| --- | --- | --- |
| <start> | Numeric | Source start address |
| <end> | Numeric | Source end address (including this address) |
| <dest> | Numeric | Destination start address |
| <state> | Keyword | Verify flag (optional, default = V) |
| | V | Verify |
| | N | No verify |

**Example:**

| | |
|---|---|
| MEMORY_MOVE H'1000 H'1FFF H'2000 | Moves area H'1000-H'1FFF to H'2000. |
| MV H'FB80 H'FF7F H'3000 | Moves area H'FB80-H'FF7F to H'3000. |

# MEMORY_TEST

**Abbreviation:** MT

**Description:**
A full read/write/verify test is performed on the address range specified, destroying the original contents. The test will access the memory according to the map settings.

**Syntax:**
    mt <start> <end>

| Parameter | Type | Description |
|---|---|---|
| <start> | Numeric | Start address |
| <end> | Numeric | End address (including this address) |

**Example:**

| | |
|---|---|
| MEMORY_TEST H'8000 H'BFFF | Tests from H'8000 to H'BFFF. |
| MT H'4000 H'5000 | Tests integrity from H'4000 to H'5000. |

# QUIT

**Abbreviation:** QU

**Description:**
Exits HDI. Closes log file if open.

**Syntax:**
    qu

| Parameter | Type | Description |
|---|---|---|
| none | | Exits HDI |

**Example:**

| | |
|---|---|
| QU | Exits HDI. |

## RADIX

**Abbreviation:** RA

**Description:**
Sets default input radix. If no parameters are specified, the current radix is displayed. Radix can be changed by using B'/H'/D'/O' before numeric data.

**Syntax:**
ra [<mode>]

| Parameter | Type | Description |
|---|---|---|
| none | | Displays current radix |
| <mode> | Keyword | Sets radix to specified type |
| | H | Sets radix to hexadecimal |
| | D | Sets radix to decimal |
| | O | Sets radix to octal |
| | B | Sets radix to binary |

**Example:**

RADIX                    Displays the current radix.

RA H                     Sets the radix to hexadecimal.

## REGISTER_DISPLAY

**Abbreviation:** RD

**Description:**
Displays CPU register values.

**Syntax:**
rd

| Parameter | Type | Description |
|---|---|---|
| none | | Displays all register values |

**Example:**

RD                       Displays all register values.

## REGISTER_SET

**Abbreviation:** RS

**Description:**
Changes the contents of a register.

**Syntax:**

rs <register> <value> <mode>

| Parameter | Type | Description |
|---|---|---|
| <register> | Keyword | Register name |
| <value> | Numeric | Register value |
| <mode> | Keyword | Data size (default = register size) |
|  | byte | Byte |
|  | word | Word |
|  | long | Long word |
|  | single | Single-precision floating-point |
|  | double | Double-precision floating-point |

**Example:**

| | |
|---|---|
| RS PC _StartUp | Sets the program counter to the address defined by the symbol _StartUp |
| RS R0 H'1234 WORD | Sets word data H'1234 to R0. |

# RESET

**Abbreviation:** RE

**Description:**

Resets the microprocessor. All register values are set to the initial state for the device. Memory mapping and breakpoints are not affected.

**Syntax:**

re

| Parameter | Type | Description |
|---|---|---|
| none | | Resets the microprocessor |

**Example:**

| | |
|---|---|
| RE | Resets the microprocessor. |

## SLEEP

**Abbreviation:** none

**Description:**
Delays command execution for a specified number of milliseconds.

**Syntax:**
sleep <milliseconds>

| Parameter | Type | Description |
|---|---|---|
| < milliseconds > | Numeric | Delayed time (millisecond) |

Default radix (it is not always decimal) is used, if you do not specify D'.

**Example:**
SLEEP D'9000          Delays for 9 seconds.

## STEP

**Abbreviation:** ST

**Description:**
Single-step (source line or instruction) execution.
Performs a specified number of instructions, from current PC.
Default is stepping by lines if source debugging is available. Count default is 1.

**Syntax:**
st [<mode>] [<count>]

| Parameter | Type | Description |
|---|---|---|
| <mode> | Keyword | Type of stepping (optional) |
| | instruction | Steps by assembly instruction |
| | line | Steps by source code line |
| <count> | Numeric | Number of steps (optional, default = 1) |

**Example:**
STEP 9          Steps code for 9 steps.

## STEP_OUT

**Abbreviation:** SP

**Description:**
Step the program out of the current function. (i.e., a step up). This works for both assembly-language and source level debugging.

**Syntax:**
    sp

| Parameter | Type | Description |
|-----------|------|-------------|
| none | | Steps the program out of the current function |

**Example:**
    SP                          Steps the program out of the current function.

# STEP_OVER

**Abbreviation:** SO

**Description:**
Step-over (function call, source line or instruction) execution.
Performs a specified number of instructions, from current PC.

This command differs from STEP in that it does not perform single-step operation
in subroutines or interrupt routines. These are executed at full speed.

**Syntax:**
    so [<mode>] [<count>]

| Parameter | Type | Description |
|-----------|------|-------------|
| <mode> | Keyword | Type of stepping (optional) |
| | instruction | Steps by assembly instruction |
| | line | Step by source code line |
| <count> | Numeric | Number of steps (optional, default = 1) |

**Example:**
    SO                          Steps over 1-step code.

# STEP_RATE

**Abbreviation:** SR

**Description:**
Controls the speed of stepping in the STEP and STEP_OVER commands. A rate
of 6 causes the fastest stepping. A value of 1 is the slowest.

**Syntax:**
    sr <rate>

| Parameter | Type | Description |
|-----------|------|-------------|
| none | | Displays the step rate |
| <rate> | Numeric | Step rate 1 to 6 (6 = fastest) |

**Example:**
    SR                          Displays the current step rate.

    SR 6                        Specifies the fastest step rate.

## SUBMIT

**Abbreviation:** SU

**Description:**
Executes a file of commands. Nested submit files are permitted. Any error aborts the file. The [stop] button terminates the process.

**Syntax:**
su <filename>

| Parameter | Type | Description |
|---|---|---|
| <filename> | String | File name |

**Example:**

| | |
|---|---|
| SUBMIT COMMAND.HDC | Processes the file COMMAND.HDC. |
| SU A:SETUP.TXT | Processes the file SETUP.TXT on drive A:. |

## SYMBOL_ADD

**Abbreviation:** SA

**Description:**
Adds a symbol, or changes an existing one.

**Syntax:**
sa <symbol> <value>

| Parameter | Type | Description |
|---|---|---|
| <symbol> | String | Symbol name |
| <value> | Numeric | Value |

**Example:**

| | |
|---|---|
| SYMBOL_ADD start H'1000 | Defines start to be H'1000. |
| SA END_OF_TABLE 1000 | Defines END_OF_TABLE to be 1000 using current default radix. |

# SYMBOL_CLEAR

**Abbreviation:** SC

**Description:**
Deletes a symbol. If no parameters are specified, deletes all symbols (after confirmation).

**Syntax:**
  sc [<symbol>]

| Parameter | Type | Description |
|-----------|------|-------------|
| none | | Deletes all symbols |
| <symbol> | String | Symbol name |

**Example:**

| SYMBOL_CLEAR | Deletes all symbols (after confirmation). |
|--------------|-------------------------------------------|
| SC start | Deletes the symbol 'start'. |

# SYMBOL_LOAD

**Abbreviation:** SL

**Description:**
Loads symbols from file. File must be in XLINK Pentica-b format (i.e. 'XXXXH name'). The symbols are added to the existing symbol table. The symbol file extension default is **.SYM.**

**Syntax:**
  sl <filename>

| Parameter | Type | Description |
|-----------|------|-------------|
| <filename> | String | File name |

**Example:**

| SYMBOL_LOAD TEST.SYM | Loads the file TEST.SYM. |
|----------------------|--------------------------|
| SL MY_CODE.SYM | Loads the file MY_CODE.SYM. |

# SYMBOL_SAVE

**Abbreviation:** SS

**Description:**

Saves symbols to a file in XLINK Pentica-b format. The symbol file extension default is **.SYM**. If the file name already exists, then a prompt to overwrite the file is displayed.

**Syntax:**

ss <filename>

| Parameter | Type | Description |
|---|---|---|
| <filename> | String | File name |

**Example:**

| | |
|---|---|
| SYMBOL_SAVE TEST | Saves symbol table to TEST.SYM. |
| SS MY_CODE.SYM | Saves the symbol table to MY_CODE.SYM. |

# SYMBOL_VIEW

**Abbreviation:** SV

**Description:**

Displays all defined symbols, or those containing the case sensitive string pattern.

**Syntax:**

sv [<pattern>]

| Parameter | Type | Description |
|---|---|---|
| none | | Displays all symbols |
| <pattern> | String | Displays the symbols including the specified string pattern |

**Example:**

| | |
|---|---|
| SYMBOL_VIEW BUFFER | Displays all symbols containing the word BUFFER. |
| SV | Displays all the symbols. |

# TRACE

**Abbreviation:** TR

**Description:**
Displays the trace buffer contents. If no trace delay is set, the last (most recently executed) cycle in the buffer is 0, and older cycles have negative values. If trace delay is set, the cycle on which the level 1 breakpoint occurred will be 0 and the most recent cycle will have the trace delay value.

**Syntax:**
tr [<start rec> [<count>]]

| Parameter | Type | Description |
|---|---|---|
| <start rec> | Numeric | Offset (optional, default = most recent cycle - 9) |
| <count> | Numeric | Count (optional, default - 10) |

**Example:**

TR -10 5                         Displays five lines of trace buffer contents
                                 starting from cycle -10.

## *Debugging Platform-Specific Commands*

The following lists the debugging platform-specific commands - typically for breakpoints, tracing, memory mapping, and configuration. Refer to the supplied *Debugging Platform User's Manual* for details.

    ANALYSIS_RANGE
    BREAKPOINT
    BREAKPOINT_CLEAR
    BREAKPOINT_DISPLAY
    BREAKPOINT_ENABLE
    BREAKPOINT_SEQUENCE
    BREAK_ACCESS
    BREAK_CLEAR
    BREAK_DATA
    BREAK_DISPLAY
    BREAK_ENABLE
    BREAK_REGISTER
    BREAK_SEQUENCE
    CLOCK
    DEVICE_TYPE
    MAP_SET
    MODE
    REFRESH
    TEST_EMULATOR
    TIMER
    TRACE_ACQUISITION
    TRACE_COMPARE
    TRACE_SAVE
    TRACE_SEARCH
    USER_SIGNAL

# Appendix C - Command Line Summary Chart

| Long name | Short name | Description |
|---|---|---|
| ! | - | Comment |
| ACCESS | AC | Sets action on illegal access |
| ANALYSIS | AN | Enables or disables performance analysis |
| ANALYSIS_RANGE | AR | Sets or displays performance analysis ranges |
| ANALYSIS_RANGE_DELETE | AD | Deletes a performance analysis range |
| ASSEMBLE | AS | Assembles instructions into memory |
| ASSERT | - | Checks if an expression is true or false |
| BREAKPOINT | BP | Sets a breakpoint |
| BREAKPOINT_CLEAR | BC | Clears a breakpoint or all breakpoints |
| BREAKPOINT_DISPLAY | BD | Displays breakpoints |
| BREAKPOINT_ENABLE | BE | Enables or disables one or all breakpoints |
| BREAKPOINT_SEQUENCE | BS | Defines the events which arm or reset an event |
| BREAK_ACCESS | BA | Sets a memory range access as a breakpoint |
| BREAK_CLEAR | BC | Deletes a breakpoint |
| BREAK_DATA | BD | Sets a memory data value as a break condition |
| BREAK_DISPLAY | BI | Displays breakpoints |
| BREAK_ENABLE | BE | Enables or disables one or all breakpoints |
| BREAK_REGISTER | BR | Sets a register value as a break condition |
| BREAK_SEQUENCE | BS | Sets sequential breakpoints |
| CLOCK | CK | Sets emulator CPU clock rate |
| DEVICE_TYPE | DE | Selects device type to emulate |
| DISASSEMBLE | DA | Disassembles memory contents |
| ERASE | ER | Clears the Command Line window |
| EVALUATE | EV | Evaluates an expression |
| FILE_LOAD | FL | Loads an object (program) file |
| FILE_SAVE | FS | Saves memory to a file |
| FILE_VERIFY | FV | Verifies file contents against memory |
| GO | GO | Runs program |
| GO_RESET | GR | Runs program from reset |
| GO_TILL | GT | Runs program until temporary breakpoint |
| HALT | HA | Halts program |
| HELP | HE | Gets help for command line or help on a command |
| INITIALISE | IN | Initializes HDI and debugging platform system |
| INTERRUPTS | IR | Enables or disables debugging platform system interrupts |
| LOG | LO | Controls command output logging |
| MAP_DISPLAY | MA | Displays memory mapping |
| MAP_SET | MS | Sets up memory mapping |
| MEMORY_DISPLAY | MD | Displays memory contents |
| MEMORY_EDIT | ME | Modifies memory contents |
| MEMORY_FILL | MF | Fills a memory area |

| Long name | Short name | Description |
|-----------|-----------|-------------|
| MEMORY_MOVE | MV | Moves a block of memory |
| MEMORY_TEST | MT | Tests a block of memory |
| MODE | MO | Sets or displays CPU mode |
| QUIT | QU | Exits HDI |
| RADIX | RA | Sets default input radix |
| REFRESH | RF | Refreshes memory-related window contents |
| REGISTER_DISPLAY | RD | Displays CPU register values |
| REGISTER_SET | RS | Changes CPU register contents |
| RESET | RE | Resets CPU |
| SLEEP | - | Delays command execution. |
| STEP | ST | Steps program (by instructions or source lines) |
| STEP_OUT | SP | Steps out of the current function |
| STEP_OVER | SO | Steps program, not stepping into functions |
| STEP_RATE | SR | Sets rate of stepping |
| SUBMIT | SU | Executes a file of commands |
| SYMBOL_ADD | SA | Defines a symbol |
| SYMBOL_CLEAR | SC | Deletes a symbol or all symbols |
| SYMBOL_LOAD | SL | Loads symbols from a file |
| SYMBOL_SAVE | SS | Saves symbols to a file |
| SYMBOL_VIEW | SV | Displays symbols |
| TEST_EMULATOR | TE | Tests emulator hardware |
| TIMER | TI | Sets or displays the timer resolution |
| TRACE | TR | Displays trace buffer contents |
| TRACE ACQUISITION | TA | Sets or displays trace acquisition parameters |
| TRACE_COMPARE | TC | Compares a saved trace file with the current trace data |
| TRACE_SAVE | TV | Saves the trace data to a file in binary format |
| TRACE_SEARCH | TS | Searches trace data |
| USER_SIGNALS | US | Enables or disables user signals (NMI, Reset, etc.) |

# Appendix D - GUI Command Summary

| Menu | Item | Accelerator | Toolbar Graphic |
|------|------|-------------|-----------------|
| File | New Session... | Ctrl+N |  |
|      | Load Session... | Ctrl+O |  |
|      | Save Session | Ctrl+S |  |
|      | Save Session As... | | |
|      | Load Program... | |  |
|      | Initialize | |  |
|      | Exit | Alt+F4 | |
| Edit | Cut | Ctrl+X |  |
|      | Copy | Ctrl+C |  |
|      | Paste | Ctrl+V |  |
|      | Find | F3 |  |
|      | Evaluate | |  |
| View | Toolbar | | |
|      | Status bar | | |
|      | Breakpoint | Ctrl+B |  |
|      | Command Line | Ctrl+L |  |
|      | Disassembly... | Ctrl+D |  |
|      | I/O Area | Ctrl+I |  |
|      | Labels | Ctrl+A |  |
|      | Locals | Ctrl+Shift+W |  |
|      | Memory... | Ctrl+M |  |
|      | Performance Analysis | Ctrl+P |  |
|      | Registers | Ctrl+R |  |

| Menu | Item | Accelerator | Toolbar Graphic |
|------|------|-------------|-----------------|
| | Source... | Ctrl+K | |
| | Status | Ctrl+U | |
| | Trace | Ctrl+T | |
| | Watch | Ctrl+W | |
| Run | Reset CPU | | |
| | Go | F5 | |
| | Reset Go | Shift+F5 | |
| | Go To Cursor | | |
| | Set PC To Cursor | | |
| | Run... | | |
| | Step In | F8 | |
| | Step Over | F7 | |
| | Step Out | | |
| | Step... | | |
| | Halt | Esc | |
| Memory | Refresh | F12 | |
| | Load... | | |
| | Save... | | |
| | Verify... | | |
| | Test... | | |
| | Fill... | | |
| | Copy... | | |
| | Compare... | | |
| | Search... | | |

| Menu | Item | Accelerator | Toolbar Graphic |
|---|---|---|---|
| | Configure Map... | | ▨ |
| | Configure Overlay... | | ▨ |
| Setup | Options... | | ▨ |
| | Radix (Input)        ➤ | | |
| |    Hexadecimal | | 16 |
| |    Decimal | | 10 |
| |    Octal | | 8 |
| |    Binary | | 2 |
| | Customize        ➤ | | |
| |    Toolbar... | | ▨ |
| |    Font... | | A |
| |    File Filter... | | |
| | Configure Platform... | | ▨ |
| Window | Cascade | | ▨ |
| | Tile | | ▨ |
| | Arrange Icons | | ▨ |
| | Close All | | |
| Help | Index | F1 | ▨ |
| | Using Help | | |
| | Search for Help on | | |
| | About HDI | | |

# Appendix E - I/O Register File Format

HDI formats the I/O Registers window based on information it finds in an I/O Register definition file. When you select a debugging platform using the [Setup->Configure Platform...] menu option, HDI will look for a "<device>.IO" file corresponding to the selected device and load it if it exists. This file is a formatted text file that describes the I/O modules and the address and size of their registers. You can edit this file, with a text editor, to add support for memory mapped registers or peripherals you may have specific to your application e.g. registers in an ASIC device mapped into the microcomputer's address space.

## File format

Each module name must be defined in the [Modules] definition section and the numbering of each module must be sequential. Each module corresponds to a register definition section and within the section each entry defines an I/O register.

The 'BaseAddress' definition is for devices where the location of I/O registers moves in the address space depending on the CPU mode. In this case, the 'BaseAddress' value is the base address of the I/O registers in one specific mode and the addresses used in the register definitions are the address locations of the registers in the same mode. When the I/O register file is actually used, the 'BaseAddress' value is subtracted from the defined register address and the resultant offset added to the relevant base address for the selected mode.

Each module has a section that defines the registers forming it along with an optional dependency, the dependency is checked to see if the module is enabled or not. Each register name must be defined in the section and the numbering of each register must be sequential. The dependency is entered in the section as dep=<reg> <bit> <value>.

1.  <reg> is the register id of the dependency.

2.  <bit> is the bit position within the register.

3.  <value> is the value that the bit must be for the module to be enabled.

The [Register] definition entry is entered in the format id=<name> <address> [<size> [<absolute>[<format>[<bitfields>]]]].

1. <name> register name to be displayed.

2. <address> address of the register.

3. <size> which may be B, W or L for byte, word, or long word (default is byte).

4. <absolute> which can be set to A if the register is at an absolute address. This is only relevant if the I/O area address range moves about on the CPU in different modes. In this case, if a register is defined as absolute the base address offset calculation is not performed and the specified address is used directly.

5. <format> Format for register output. Valid values are H for Hexadecimal, D for decimal, and B for binary.

6. <bitfields> section defining the bits within the register.

Bitfield sections define the bits within a register each entry is of the type bit<no>=<name>.

1. <no> is the bit number.

2. <name> is a symbolic name of the bit.

Comment lines are allowed and must start with a ";" character.

Example on next page.

## Example:

Comment     ; H8S/2655 Series I/O Register Definitions File

Modules

```
[Modules]
BaseAddress=0
Module1=Power_Down_Mode_Registers
Module2=DMA_Channel_Common
Module3=DMA_0_Short_Address_Mode
. . .
Module42=Bus_Controller
Module43=System_Control
Module44=Interrupt_Controller
```

. . .

Module
definition

```
[DMA_Channel_Common]
reg0=regDMAWER
reg1=regDMATCR
reg2=regDMABCRH/SAM
reg3=regDMABCRL/SAM
reg4=regDMABCRH/FAM
reg5=regDMABCRL/FAM
dep= regMSTPCRH 7 0
```

Register

Bit

Value

. . .

Register
definition

```
[regDMAWER]
id=DMAWER 0xffff00 B A H dmawer_bitfields
```

Register name

Address

Size

Absolute address flag

Format

Bitfields

. . .

Bitfields
definition

```
[dmawer_bitfields]
bit0=WE0A
bit1=WE0B
bit2=WE1A
bit3=WE1B
```

# Appendix F - Symbol File Format

In order for HDI to be able to understand and decode the symbol file correctly, the file must be formatted as a Pentica-B file:

1. The file must be a plain ASCII text file.

2. The file must start with the word "BEGIN".

3. Each symbol must be on a separate line with the value first, in hexadecimal terminated by an "H", followed by a space then the symbol text.

4. The file must end with the word "END".

Example:

```
BEGIN
11FAH  Symbol_name_1
11FCH  Symbol_name_2
11FEH  Symbol_name_3
1200H  Symbol_name_4
END
```