To all our customers

Regarding the change of names mentioned in the document, such as Hitachi Electric and Hitachi XX, to Renesas Technology Corp.

The semiconductor operations of Mitsubishi Electric and Hitachi were transferred to Renesas Technology Corporation on April 1st 2003. These operations include microcomputer, logic, analog and discrete devices, and memory chips other than DRAMs (flash memory, SRAMs etc.) Accordingly, although Hitachi, Hitachi, Ltd., Hitachi Semiconductors, and other Hitachi brand names are mentioned in the document, these names have in fact all been changed to Renesas Technology Corp. Thank you for your understanding. Except for our corporate trademark, logo and corporate statement, no changes whatsoever have been made to the contents of the document, and these changes do not constitute any alteration to the contents of the document itself.

Renesas Technology Home Page: http://www.renesas.com

Renesas Technology Corp.
Customer Support Dept.
April 1, 2003

Renesas Technology Corp.

Cautions

Keep safety first in your circuit designs!

1. Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
   Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

1. These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
2. Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
3. All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
   The information described here may contain technical inaccuracies or typographical errors.
   Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
   Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
4. When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
5. Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
6. The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
7. If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
   Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
8. Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.

Hitachi Debugging Interface

User's Manual

ADE-702-161A

Rev. 2.0
2/9/99
Hitachi, Ltd.

HS6400DIIW3SE

Cautions

1. Hitachi neither warrants nor grants licenses of any rights of Hitachi’s or any third party’s patent, copyright, trademark, or other intellectual property rights for information contained in this document. Hitachi bears no responsibility for problems that may arise with third party’s rights, including intellectual property rights, in connection with use of the information contained in this document.
2. Products and product specifications may be subject to change without notice. Confirm that you have received the latest product standards or specifications before final design, purchase or use.
3. Hitachi makes every attempt to ensure that its products are of high quality and reliability. However, contact Hitachi’s sales office before using the product in an application that demands especially high quality and reliability or where its failure or malfunction may directly threaten human life or cause risk of bodily injury, such as aerospace, aeronautics, nuclear power, combustion control, transportation, traffic, safety equipment or medical equipment for life support.
4. Design your application so that the product is used within the ranges guaranteed by Hitachi particularly for maximum rating, operating supply voltage range, heat radiation characteristics, installation conditions and other characteristics. Hitachi bears no responsibility for failure or damage when used beyond the guaranteed ranges. Even within the guaranteed ranges, consider normally foreseeable failure rates or failure modes in semiconductor devices and employ systemic measures such as fail-safes, so that the equipment incorporating Hitachi product does not cause bodily injury, fire or other consequential damage due to operation of the Hitachi product.
5. This product is not designed to be radiation resistant.
6. No one is permitted to reproduce or duplicate, in any form, the whole or part of this document without written approval from Hitachi.
7. Contact Hitachi’s sales office for any questions regarding this document or Hitachi semiconductor products.

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.

System Overview describes how the different software modules make up the HDI system and which modules are needed for a specific configuration. The different parts of the user interface are described and some common features explored.

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

Section 1 Introduction 1

1.1 Key Features...1

Section 2 System Overview...3

2.1 User Interface 3

2.1.1 Menu Bar...3

2.1.2 Toolbar 5

2.1.3 Status Bar 6

2.1.4 Pop-Up Menus 7

2.2 Data Entry 8

2.2.1 Operators...8

2.2.2 Data Formats 8

2.2.3 Precision..8

2.2.4 Expression Examples 9

2.2.5 Symbol Format...9

2.2.6 Symbol Examples...9

2.3 Help 10

2.3.1 Context Sensitive Help...10

Section 3 Preparing to Debug..11

3.1 Compiling for Debug...11

3.2 Selecting a Debugging Platform..11

3.3 Configuring the Debugging Platform...12

3.3.1 Setup..12

3.3.2 Mapping 12

3.3.3 Status..14

3.4 Downloading a Program..16

3.4.1 Selecting a File Type...17

3.4.2 Setting a File Path Name 17

Section 4 Looking at Your Program...19

4.1 Viewing the Code..19

4.1.1 Viewing Assembly-Language Code..20

4.1.2 Modifying Assembly-Language Code 21

4.1.3 Displaying Source Code..22

4.1.4 Displaying Mixed Code 22

4.2 Looking at Symbols...22

4.2.1 Listing Symbols...22

4.2.2 Finding a Symbol 23

4.2.3 Finding Again.....24
4.2.4 Adding Symbols.....24
4.2.5 Adding a Symbol from the Program Window.....24
4.2.6 Modifying Symbols.....25
4.2.7 Deleting Symbols.....26
4.2.8 Deleting All Symbols.....26
4.2.9 Loading a Symbol File.....27
4.2.10 Saving a Symbol File.....28
4.2.11 Symbol File Format.....28
4.3 Looking at a Specific Address.....29
4.3.1 Looking at the Current Program Counter Address.....29
4.4 Finding Text.....30
4.5 Looking at a Text File.....30
4.5.1 Opening a Text Window.....30
4.5.2 Copying Text.....32
4.5.3 Finding Text.....33

Section 5 Working with Memory.....35
5.1 Looking at an Area of Memory.....35
5.1.1 Displaying Memory as Bytes.....36
5.1.2 Displaying Memory as Words.....36
5.1.3 Displaying Memory as Long words.....37
5.1.4 Displaying Memory as Single-Precision Floating Point.....37
5.1.5 Displaying Memory as Double-Precision Floating Point.....38
5.1.6 Displaying Memory as ASCII.....38
5.1.7 Looking at a Different Area of Memory.....38
5.2 Modifying Memory Contents.....39
5.2.1 Quick Edit.....39
5.2.2 Full Edit.....39
5.3 Finding a Value in Memory.....40
5.4 Filling an Area of Memory with a Value.....41
5.4.1 Selecting a Memory Range.....41
5.4.2 Filling the Range.....42
5.5 Moving an Area of Memory.....43
5.6 Testing an Area of Memory.....44
5.7 Saving an Area of Memory.....45
5.8 Loading an Area of Memory.....46
5.9 Verifying an Area of Memory.....47

Section 6 Executing Your Program.....49
6.1 Running from Reset.....49
6.2 Continuing Run.....49
6.3 Running to the Cursor.....50

6.4 Running to Several Points 51
6.5 Single Step...51
6.5.1 Stepping Into a Function 51
6.5.2 Stepping Over a Function Call 51
6.6 Stepping Out of a Function...51
6.7 Multiple Steps...52

Section 7 Stopping Your Program...53
7.1 Halting Execution..53
7.2 Program Breakpoints (PC Breakpoints) 53
7.2.1 Clearing Program Breakpoints (PC Breakpoints) 54
7.3 The Breakpoints Window..54
7.3.1 Adding a Breakpoint 55
7.3.2 Modifying a Breakpoint 56
7.3.3 Deleting a Breakpoint..56
7.3.4 Deleting All Breakpoints..56
7.4 Disabling Breakpoints 56
7.4.1 Disabling a Breakpoint...57
7.4.2 Enabling a Breakpoint..57
7.5 Temporary Breakpoints 57
7.6 Hardware Breakpoints(Event) 58

Section 8 Looking at Variables 59
8.1 Instant Watch...59
8.2 Using Watch Items 60
8.2.1 Adding a Watch...60
8.2.2 Expanding a Watch 62
8.2.3 Modifying Radix for Watch Item Display...62
8.2.4 Changing a Watch Item’s Value 63
8.2.5 Deleting a Watch..64
8.3 Looking at Local Variables 64
8.4 Looking at Registers..65
8.4.1 Modifying Register Contents 65
8.4.2 Using Register Contents...67
8.5 Looking at I/O 67
8.5.1 Opening an I/O Registers Window...67
8.5.2 Expanding an I/O Register Display 68
8.5.3 Modifying I/O Register Contents 68
8.5.4 I/O Register Files 69

Section 9 Overlay Function...73
9.1 Displaying Section Group 73
9.2 Setting Section Group 74

Section 10 Selecting Functions..77
10.1 Displaying Functions..77
10.2 Specifying Functions...78
10.2.1 Selecting a Function...78
10.2.2 Deleting a Function..78
10.2.3 Setting a Function...78

Section 11 Configuring the User Interface 79
11.1 Arranging Windows 79
11.1.1 Minimizing Windows...79
11.1.2 Arranging Icons..80
11.1.3 Tiling Windows...81
11.1.4 Cascading Windows..81
11.2 Locating Currently Open Windows...81
11.2.1 Locating the Next Window 82
11.2.2 Locating a Specific Window...82
11.3 Enabling/Disabling the Status Bar...82
11.4 Enabling/Disabling the Toolbar 83
11.5 Customizing the Toolbar..83
11.5.1 Button Categories..83
11.5.2 Adding a Button to the Toolbar..84
11.5.3 Positioning a Button in the Toolbar..84
11.5.4 Removing a Button from the Toolbar 84
11.6 Customizing the Fonts...84
11.7 Saving a Session..85
11.8 Loading a Session..86
11.9 Setting HDI Options...86
11.10 Setting the Default Radix 87

Section 12 Menus..89
12.1 File 89
12.1.1 Load Program..89
12.1.2 Save Memory... 89
12.1.3 Verify Memory..90
12.1.4 Save Session..90
12.1.5 Load Session... 90
12.1.6 Save Session As... 90
12.1.7 Initialise...90
12.1.8 Exit..90
12.2 Edit 90
12.2.1 Cut...91
12.2.2 Copy..91
12.2.3 Paste 91

12.2.4 Find 91
12.2.5 Set Line 91
12.2.6 Fill Memory... 91
12.2.7 Move Memory... 92
12.2.8 Test Memory... 92
12.2.9 Update Memory 92
12.3 View 92
12.3.1 Toolbar 92
12.3.2 Status Bar 92
12.3.3 Breakpoint Window 92
12.3.4 Command Line Window 93
12.3.5 I/O Register Window 93
12.3.6 Local Variable Window 93
12.3.7 Memory Mapping Window 93
12.3.8 Memory Window... 93
12.3.9 Performance Analysis Window 93
12.3.10 Program Window... 93
12.3.11 Register Window 94
12.3.12 Status Window 94
12.3.13 Text Window... 94
12.3.14 Trace Window 94
12.3.15 Watch Window 94
12.4 Run 94
12.4.1 Go 94
12.4.2 Go Reset 94
12.4.3 Go To Cursor 95
12.4.4 Run... 95
12.4.5 Step In 95
12.4.6 Step Over 95
12.4.7 Step Out 95
12.4.8 Step 95
12.4.9 Halt Program 95
12.4.10 Set PC to Cursor 96
12.4.11 Reset CPU 96
12.5 Setup 96
12.5.1 Options... 96
12.5.2 Radix 96
12.5.3 Customise 96
12.5.4 Select Platform... 96
12.5.5 Configure Platform... 97
12.5.6 Overlay... 97
12.6 Tools 97
12.6.1 Symbols... 97

12.6.2 Evaluate...97
12.7 Window...97
12.7.1 Cascade...97
12.7.2 Tile...97
12.7.3 Arrange Icons...98
12.7.4 Close All...98
12.8 Help...98
12.8.1 Index...98
12.8.2 Using Help...98
12.8.3 Search for Help on...98
12.8.4 About HDI...98

Section 13 Windows...99
13.1 Breakpoints...99
13.1.1 Help...99
13.1.2 Add...100
13.1.3 Edit...100
13.1.4 Delete...100
13.1.5 Delete All...100
13.1.6 Disable/Enable...100
13.2 Command Line...101
13.2.1 Stop...101
13.2.2 Batch File...101
13.2.3 Log File...102
13.2.4 Logging...102
13.3 I/O Registers...102
13.3.1 Copy...103
13.3.2 Edit......103
13.3.3 Expand/Collapse...103
13.3.4 Help...103
13.4 Locals...103
13.4.1 Copy...104
13.4.2 Edit Value...104
13.4.3 Radix...104
13.5 Memory Mapping...105
13.5.1 Add...105
13.5.2 Change...105
13.5.3 Reset...106
13.5.4 Help...106
13.6 Memory...106
13.6.1 Copy...107
13.6.2 Find...107
13.6.3 Fill...107

13.6.4 Move 107
13.6.5 Test...107
13.6.6 Save...107
13.6.7 Set Address...107
13.6.8 ASCII/Byte/Word/Long/Single Float/Double Float.......................108
13.7 Performance Analysis...108
13.7.1 Add Range..108
13.7.2 Edit Range..108
13.7.3 Delete Range 108
13.7.4 Reset Counts/Times...108
13.7.5 Delete All Ranges...109
13.7.6 Analysis Enabled..109
13.8 Program...109
13.8.1 Copy 111
13.8.2 Find 111
13.8.3 Set Line 111
13.8.4 Go To Cursor...111
13.8.5 Set PC Here 111
13.8.6 Toggle Breakpoint..112
13.8.7 Instant Watch...112
13.8.8 Add Watch 112
13.8.9 Source/Mixed/Assembler 112
13.9 Registers...113
13.9.1 Copy 113
13.9.2 Edit..113
13.9.3 Toggle Bit...113
13.10 System Status 114
13.10.1 Update 114
13.10.2 Copy...114
13.10.3 Configure...115
13.11 Text 115
13.11.1 Copy...115
13.11.2 Find 116
13.12 Trace...116
13.12.1 Find 116
13.12.2 Find Next...116
13.12.3 Filter 117
13.12.4 Acquisition 117
13.12.5 Halt...117
13.12.6 Restart 117
13.12.7 Snapshot 117
13.12.8 Clear...117
13.12.9 Save..117

13.13 Watch 118
13.13.1 Copy 118
13.13.2 Delete Watch 118
13.13.3 Add Watch 118
13.13.4 Edit Value..119
13.13.5 Radix 119

Appendix

Appendix A System Modules 121
Graphical User Interface 122
Object DLL 122
CPU DLL 122
Target DLL...122
Appendix B Command Line Interface 123
HDI Built-In Commands...123
Appendix C Command Line Summary Chart 151

Figures

Figure 2.1 HDI Graphical User Interface........3
Figure 2.2 Ellipsis........4
Figure 2.3 Check Mark........4
Figure 2.4 Cascading Menu........4
Figure 2.5 Accelerator Key........5
Figure 2.6 Default Toolbar........5
Figure 2.7 File Operation........5
Figure 2.8 Copy & Move Operations........5
Figure 2.9 Execution Operations........6
Figure 2.10 Open Windows........6
Figure 2.11 Help........6
Figure 2.12 Status Bar........7
Figure 2.13 Pop-Up Menu........7
Figure 2.14 Help Button........10
Figure 3.1 Select Platform Dialog Box........11
Figure 3.2 Link up Message........12
Figure 3.3 Memory Mapping Window........13
Figure 3.4 Edit Memory Mapping Window........14
Figure 3.5 System Status Window........15
Figure 3.6 Load Object File Dialog Box........16
Figure 3.7 File Type Selection........17
Figure 3.8 Download Completion Message........17
Figure 4.1 Open Dialog Box........19
Figure 4.2 Program Window (Source Display)........20
Figure 4.3 Program Window (Assembly-Language Display)........21
Figure 4.4 Assembler Dialog Box........21
Figure 4.5 Symbols Dialog Box........23
Figure 4.6 Find Symbol Containing Dialog Box........23
Figure 4.7 Add Symbol Dialog Box........24
Figure 4.8 Label Dialog Box........25
Figure 4.9 Edit Symbol Dialog Box........26
Figure 4.10 Message Box for Confirming Symbol Deletion........26
Figure 4.11 Message Box for Confirming All Symbol Deletion........27
Figure 4.12 Load Symbols Dialog Box........27
Figure 4.13 Number of Loaded Symbols........28
Figure 4.14 Set Address Dialog Box........29
Figure 4.15 Find Dialog Box........30
Figure 4.16 Open Dialog Box........31
Figure 4.17 Text Window........32
Figure 4.18 Copying Text........33
Figure 4.19 Find Dialog Box........33

Figure 5.1 Open Memory Window Dialog Box..35
Figure 5.2 Memory Window (Bytes) 36
Figure 5.3 Memory Window (Words) 36
Figure 5.4 Memory Window (Longwords)..37
Figure 5.5 Memory Window (Single-Precision Floating Point) 37
Figure 5.6 Memory Window (Double-Precision Floating Point)..38
Figure 5.7 Memory Window (ASCII)...38
Figure 5.8 Set Address Dialog Box..39
Figure 5.9 Edit Dialog Box 40
Figure 5.10 Find Memory Dialog Box...40
Figure 5.11 Address Where Data Found..41
Figure 5.12 Address Where Data Not Found..41
Figure 5.13 Selecting a Memory Range...42
Figure 5.14 Fill Memory Dialog Box 43
Figure 5.15 Move Memory Dialog Box..44
Figure 5.16 Test Memory Dialog Box 44
Figure 5.17 Save S-Record File Dialog Box...45
Figure 5.18 Message Box for Confirming File Save Completion...46
Figure 5.19 Load Object File Dialog Box...46
Figure 5.20 Message Box for Confirming File Load Completion 47
Figure 5.21 Verify S-Record File with Memory Dialog Box 47
Figure 5.22 Message Box for Confirming Verification Completion 48
Figure 6.1 Highlighted Line Corresponding to PC Address 50
Figure 6.2 Step Program Dialog Box..52
Figure 7.1 Execution Halted by Clicking Halt Button..53
Figure 7.2 Setting a Program Breakpoint...54
Figure 7.3 Break at a Program Breakpoint...54
Figure 7.4 Breakpoints Window 55
Figure 7.5 Run Program Dialog Box 58
Figure 8.1 Selecting Instant Watch 59
Figure 8.2 Instant Watch Dialog Box 60
Figure 8.3 Add Watch Dialog Box 61
Figure 8.4 Watch Window..61
Figure 8.5 Expanding a Watch..62
Figure 8.6 Modifying Radix for Watch Item Display...63
Figure 8.7 Edit Value Dialog Box...63
Figure 8.8 Locals Window..64
Figure 8.9 Registers Window..65
Figure 8.10 Register Dialog Box 66
Figure 8.11 I/O Registers Window 68
Figure 8.12 Dialog Box for Modifying I/O Register Contents 69
Figure 9.1 Overlay Dialog Box (at Opening)...73
Figure 9.2 Overlay Dialog Box (Address Range Selected)...74

Figure 9.3 Overlay Dialog Box (Highest-Priority Section Group Selected)............................75
Figure 10.1 Select Function Dialog Box..77
Figure 11.1 Minimizing a Window..79
Figure 11.2 Code Window Icon...79
Figure 11.3 Icons Before Arrangement...80
Figure 11.4 Icons After Arrangement..81
Figure 11.5 Selecting a Window...82
Figure 11.6 Customise Toolbar Dialog Box..83
Figure 11.7 Font Dialog Box...85
Figure 11.8 Session Name Display..85
Figure 11.9 HDI Options Dialog Box..86
Figure 11.10 Setting Radix..87
Figure 12.1 Menus..89
Figure 13.1 Breakpoints Window..99
Figure 13.2 Command Line Window..101
Figure 13.3 I/O Registers Window...102
Figure 13.4 Locals Window..103
Figure 13.5 Memory Mapping Window...105
Figure 13.6 Memory Window...106
Figure 13.7 Performance Analysis Window...108
Figure 13.8 Source Format..110
Figure 13.9 Assembly-Language Format..110
Figure 13.10 Registers Window..113
Figure 13.11 System Status Window...114
Figure 13.12 Text Window..115
Figure 13.13 Trace Window..116
Figure 13.14 Watch Window...118
Figure A.1 HDI System Modules...121

Section 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++ 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*.

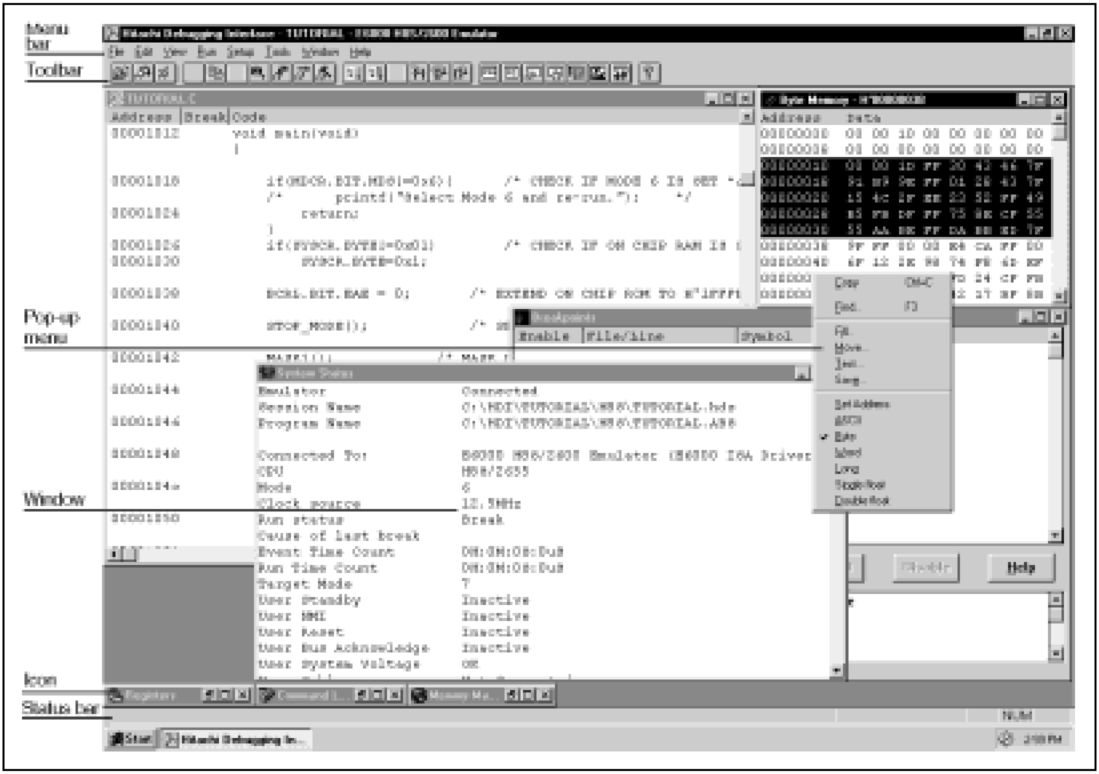
2. The HDI does not run on Windows® version 3.1.

Section 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.



```
Address   Break Code
00001012        void main(void)
                {
00001018            if(MDCR.BIT.MDS!=0x6){      /* CHECK IF MODE 6 IS SET *
                    /*       printf("Select Mode 6 and re-run.");     */
00001024                return;
                    }
00001026            if(SYSCR.BYTE!=0x01)        /* CHECK IF ON CHIP RAM IS
00001030                SYSCR.BYTE=0x1;

00001038            BCRL.BIT.EAE = 0;       /* EXTEND ON CHIP ROM TO H'1FFF

00001040            STOP_MODE();            /* ST

00001042            MASK1();            /* MASK
```

```
Emulator                  Connected
Session Name              C:\HDI\TUTORIAL\H8S\TUTORIAL.hde
Program Name              C:\HDI\TUTORIAL\H8S\TUTORIAL.ABS

Connected To:             E6000 H8S/2655 Emulator (HS6000 ISA Driver
CPU                       H8S/2655
Mode                      6
Clock source              12.5MHz
Run status                Break
Cause of last break
Event Time Count          0H:0M:0S:0uS
Run Time Count            0H:0M:0S:0uS
Target Mode               7
User Standby              Inactive
User NMI                  Inactive
User Reset                Inactive
User Bus Acknowledge      Inactive
User System Voltage       OK
```

Figure 2.1 HDI Graphical User Interface

2.1.1 Menu Bar

Debugging commands are grouped into similar areas on the Menu bar. Using the mouse you can select a command operation or invoke a dialog box or a window to interact with the system. Clicking the left mouse button on a category will pull down a menu, from which you can select an option.

If the menu option does not perform an action directly, but instead displays a dialog box or window for further user interaction, then it will be followed by an ellipsis (three dots, …):

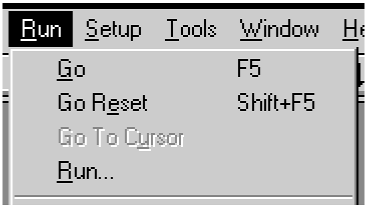


Figure 2.2 Ellipsis

If the menu action can also be invoked by pressing a function key, then the function key number (Fn) will be displayed next to the option.

If a menu option toggles a feature on or off then a check mark ( ) will be displayed next to its text when it is enabled:

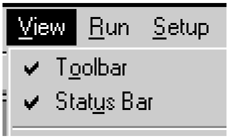


Figure 2.3 Check Mark

In this manual, a menu option selection is expressed using brackets ([ ]). For example, the above operation is expressed as **[View -> Toolbar]**. The cascading menu option selection shown below is expressed as **[Setup->Radix->Hexadecimal]**. If the menu option has a cascading menu symbol (▶) next to it then a cascading or hierarchical menu is available. Clicking on the menu option will pop up the cascading menu:

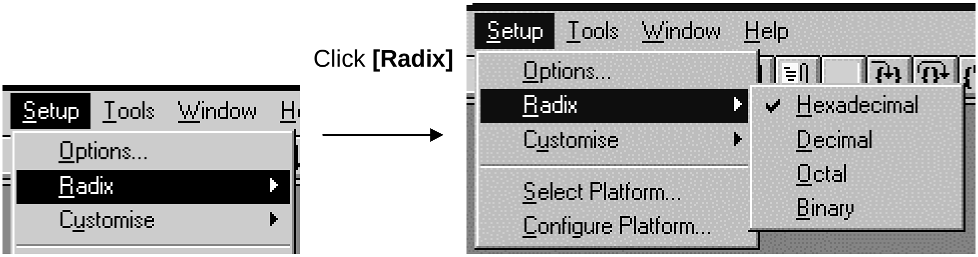


Figure 2.4 Cascading Menu

Menus can also be selected directly from the keyboard by pressing the **ALT** key followed by the corresponding key for the underlined letter or number in the menu option (called the accelerator key) that you want to select. For example, press ALT, F, L in sequence to load a program.

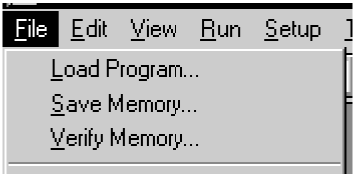


Figure 2.5 Accelerator Key

2.1.2 Toolbar

The HDI has a toolbar located below the menu bar. This provides quick access to HDI features by clicking the respective button with the mouse.

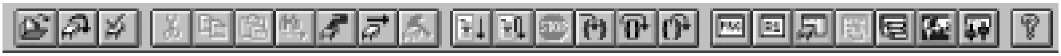


Figure 2.6 Default Toolbar

The buttons are arranged together in associated groups.

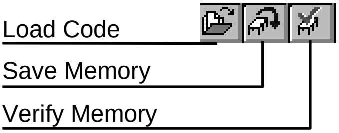


Figure 2.7 File Operation

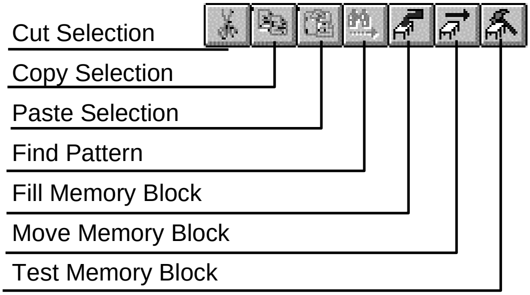


Figure 2.8 Copy & Move Operations

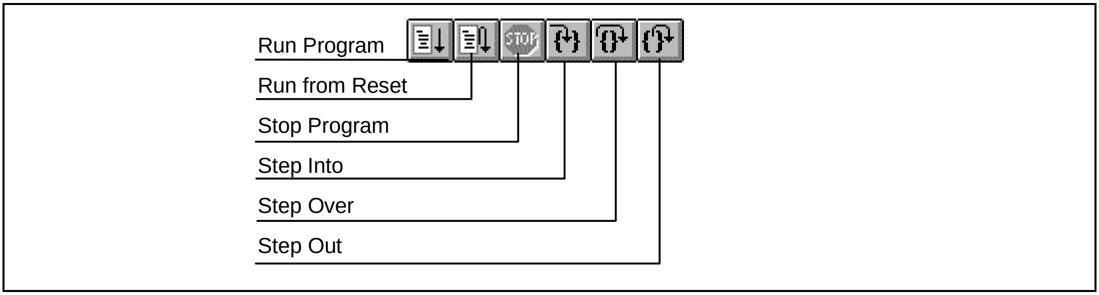


Figure 2.9 Execution Operations

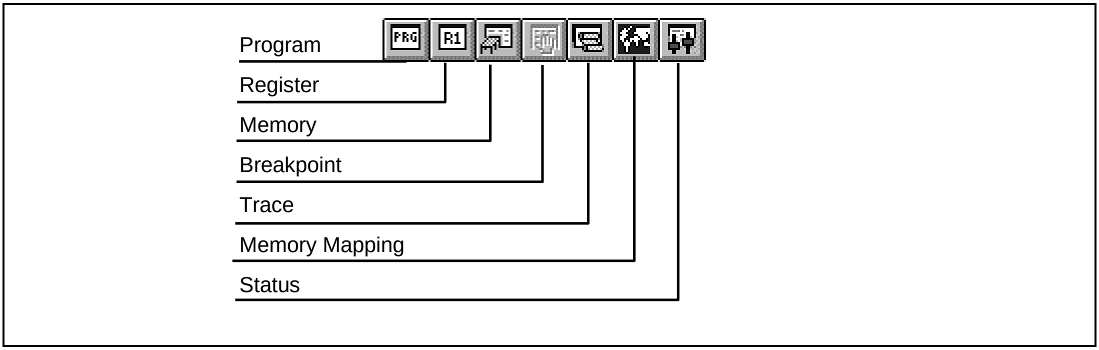


Figure 2.10 Open Windows

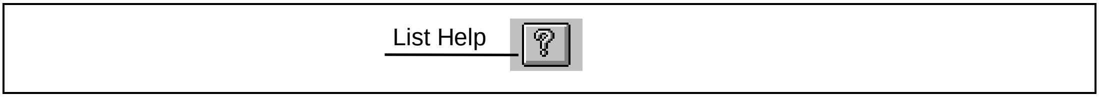


Figure 2.11 Help

The toolbar buttons can be customized to provide a button for most of the features available in HDI and can be arranged in an order that you find easiest to use.

For more details about changing the arrangement of the toolbar buttons and a detailed description of each button’s function see *section 11.5, Customizing the Toolbar*.

2.1.3 Status Bar

The Status bar is located at the bottom of the HDI application window. It provides the user with information about what the debugging system is doing or has just done and also displays the state of the Cap/Num/Scrl lock.

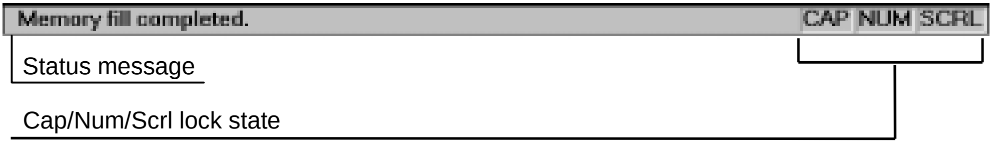
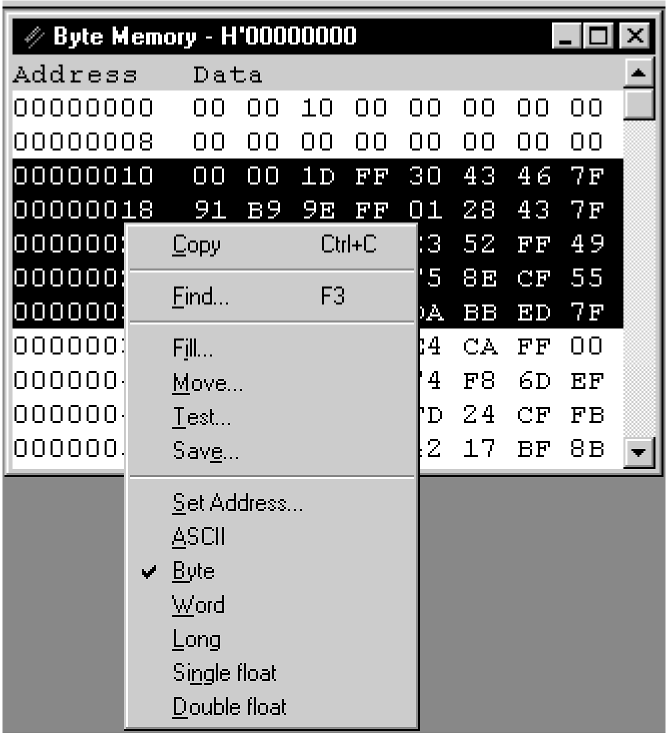

Memory fill completed.

Figure 2.12 Status Bar

2.1.4 Pop-Up Menus

Windows have Pop-up menus in order to make commonly used features easier to access. These menus are invoked by clicking the right mouse button in the window (or pressing **SHIFT+F10**) and then selecting the required menu option:


Byte Memory - H'00000000

```
Address    Data
00000000   00 00 10 00 00 00 00 00
00000008   00 00 00 00 00 00 00 00
00000010   00 00 1D FF 30 43 46 7F
00000018   91 B9 9E FF 01 28 43 7F
```

Figure 2.13 Pop-Up Menu

The contents and operation of specific Pop-up menus are explained in detail in the description of each HDI window. See *section 13, Windows*.

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++ language features such as arrays and structures is only available if an object reader that supports C/C++ language debugging is in use.

2.2.1 Operators

The C/C++ 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, with the exception of count fields which expect decimal values as a 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++ 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.
- References (&) can be specified for only one level.
- 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++)
&value                     //Specifies a reference (C/C++)
array[0]                   //Specifies an array (C/C++)
Object.*value              //Specifies reference of a member with pointer (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 location:

Figure 2.14 Help Button

2.3.1 Context Sensitive Help

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

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 location.

Section 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 debug object file.

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 puts 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

If you have only installed HDI for a single debugging platform, then it will automatically link to the installed debugging platform when launched.

However, if you have installed more than one debugging platform, then you will have to select the appropriate debugging platform at HDI start-up or with the **Select Platform** dialog box by clicking on the **[Setup->Select Platform...]** menu option.

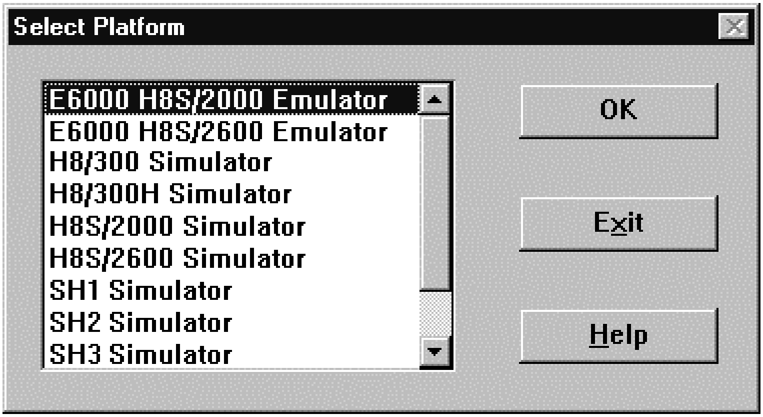


Figure 3.1 Select Platform Dialog Box

The dialog box lists all of the debugging platforms installed in the system. (These could be for hardware in-circuit emulators, software simulation engines or evaluation board monitors.) Select the debugging platform you want to use in this session.

HDI will load the target 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:

Link up

Figure 3.2 Link up Message

Note: If you have only one debugging platform installed in your system this option is disabled (the **[Setup->Select Platform...]** menu option is grayed) as HDI automatically links to that debugging platform.

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 code will be loaded.

3.3.1 Setup

To set-up the debugging platform configuration invoke 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 Platform** 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*.

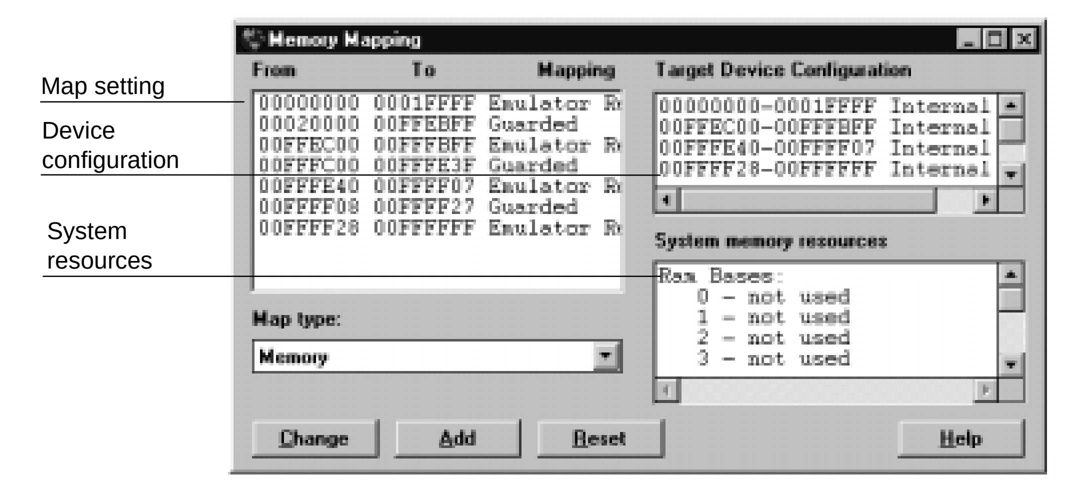
3.3.2 Mapping

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 **Configure Platform** dialog box, 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.

Again the dialog box shown will be specific to the debugging platform that you chose in the **Select Platform** dialog box. But, for example, with a hardware in-circuit emulator you will see something like:



```
From      To        Mapping
00000000  0001FFFF  Emulator R
00020000  00FFEBFF  Guarded
00FFEC00  00FFFBFF  Emulator R
00FFFC00  00FFFE3F  Guarded
00FFFE40  00FFFF07  Emulator R
00FFFF08  00FFFF27  Guarded
00FFFF28  00FFFFFF  Emulator R
```

Target Device Configuration

```
00000000-0001FFFF Internal
00FFEC00-00FFFBFF Internal
00FFFE40-00FFFF07 Internal
00FFFF28-00FFFFFF Internal
```

System memory resources

```
Ram Bases:
   0 - not used
   1 - not used
   2 - not used
   3 - not used
```

Map type: Memory

Figure 3.3   Memory Mapping Window

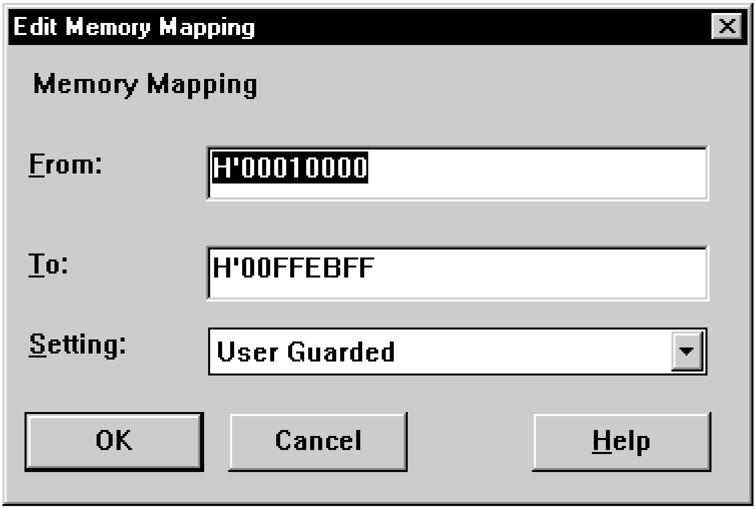
The *Map Setting* area shows how the address space is currently mapped. It lists all address ranges covering the entire address space and the type of memory to which they are set; internal or external to the emulator and any access restrictions they may have, e.g. read only or guarded (no access). This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

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. From the following dialog box you will then be able to modify the start and end addresses of the map range, and the memory type setting:



Edit Memory Mapping

Memory Mapping

From: H'00010000

To: H'00FFEBFF

Setting: User Guarded

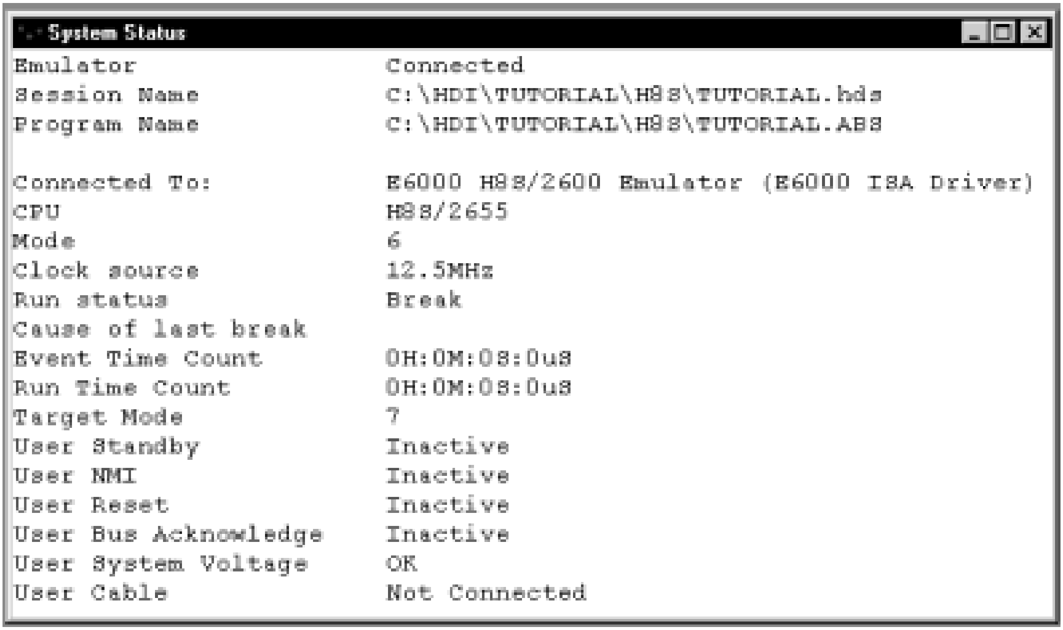
Figure 3.4 Edit Memory Mapping Dialog Box

To add a new range click on the **[Add]** button, the Add Memory Mapping dialog box will appear (it is the same as the **Edit Memory Map** 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

You can check the configuration and status of the debugging platform by looking in the **System Status** window. This is invoked from the **[View->Status Window]** menu.



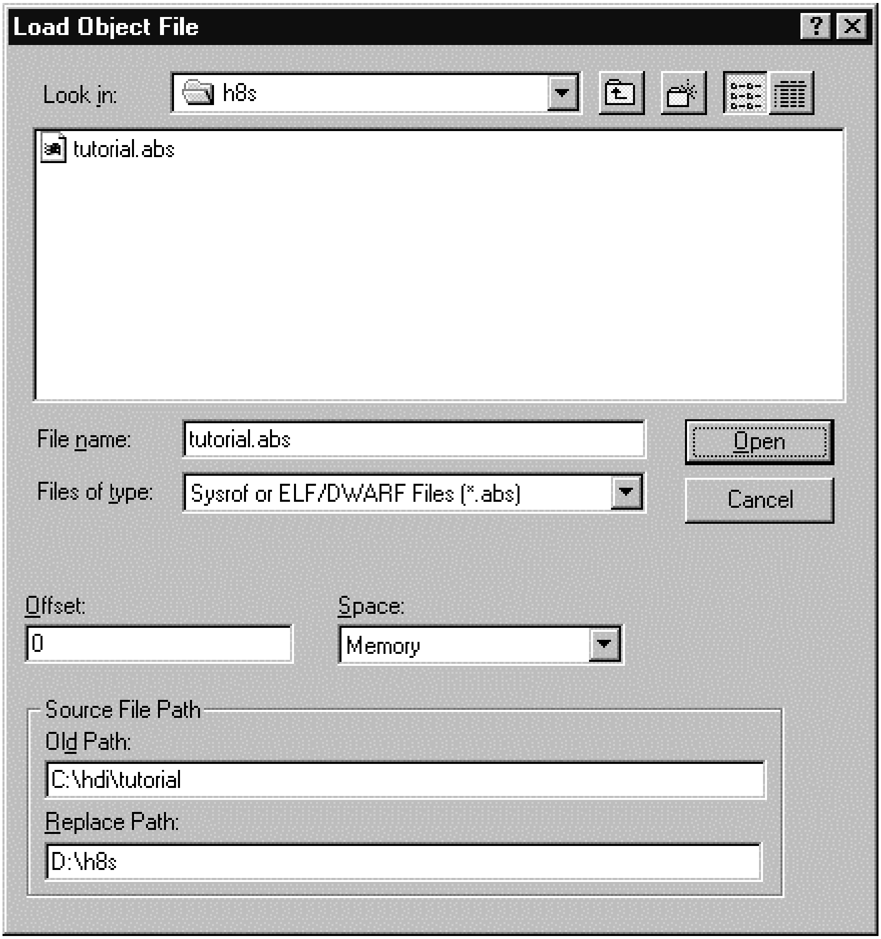
```
Emulator                   Connected
Session Name               C:\HDI\TUTORIAL\H8S\TUTORIAL.hds
Program Name               C:\HDI\TUTORIAL\H8S\TUTORIAL.ABS

Connected To:              E6000 H8S/2600 Emulator (E6000 ISA Driver)
CPU                        H8S/2655
Mode                       6
Clock source               12.5MHz
Run status                 Break
Cause of last break
Event Time Count           0H:0M:0S:0uS
Run Time Count             0H:0M:0S:0uS
Target Mode                7
User Standby               Inactive
User NMI                   Inactive
User Reset                 Inactive
User Bus Acknowledge       Inactive
User System Voltage        OK
User Cable                 Not Connected
```

Figure 3.5   System Status Window

The status in the window can be updated on demand from the pop-up menu. Click the right mouse button to pop-up the pop-up menu and select **[Configure]**.

3.4 Downloading a Program

Once you have made sure that there is memory in your system in which to download your code, you can then proceed to download a program to debug. Clicking on the **[File->Load Program...]** menu option will invoke the Load Object File dialog box:



Load Object File

Look in: h8s

tutorial.abs

File name: tutorial.abs

Files of type: Sysrof or ELF/DWARF Files (*.abs)

Offset: 0

Space: Memory

Source File Path

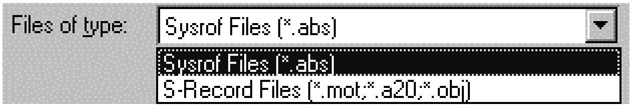
Old Path: C:\hdi\tutorial

Replace Path: D:\h8s

Figure 3.6 Load Object File Dialog Box

3.4.1 Selecting a File Type

To select a file to download, first select the type of file to display in the list area by clicking in the Files of type field and then click on the file type that you require.



Files of type: Sysrof Files (*.abs)

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.

3.4.2 Setting a File Path Name

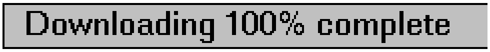
The source code can be displayed in the Program window even after moving the source file from the directory wherein it has been compiled or assembled, to another directory.

If the directory wherein the source file was compiled or assembled differs from the directory now containing the source file, specify each directory in the Load Object File dialog box before downloading the program.

In the Source File Path group edit box, enter the directory name in which the source file was compiled or assembled in the Old Path field and the directory name to which the source file was moved in the Replace Path field.

If this setting is performed before downloading the program, the source code can be displayed in the Program window.

Clicking the **[Open]** button after selecting a file will initiate the downloading. During the download HDI will give a readout of progress on the status bar, and will display the following message when it is finished.



Downloading 100% complete

Figure 3.8 Download Completion Message

A message box will be shown indicating the areas of memory in the user system that have been changed.

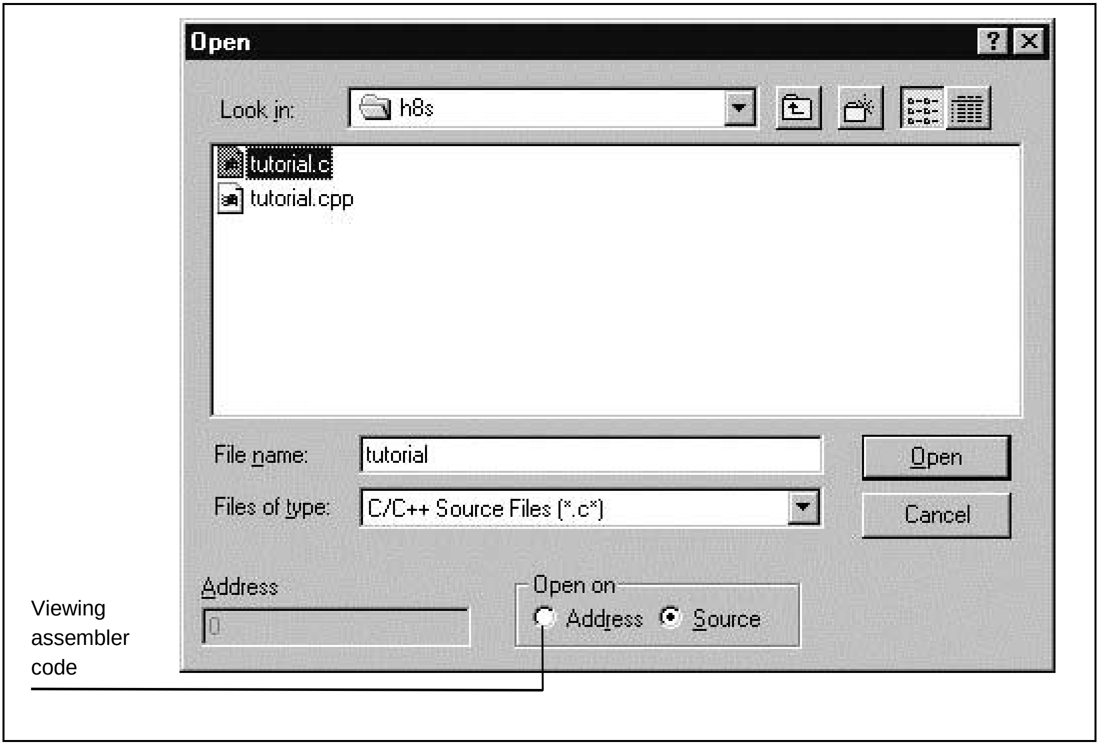
Section 4 Looking at Your Program

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.

4.1 Viewing the Code

To look at your program, open a Program window by either selecting the **[View->Program Window...]** menu option or clicking on the Program Window button [▣] on the toolbar (if it is visible).

If you do not have any source files open already, HDI will prompt you for the relevant source file via the Open dialog box:



Look in: h8s

tutorial.c

tutorial.cpp

File name: tutorial

Files of type: C/C++ Source Files (*.c*)

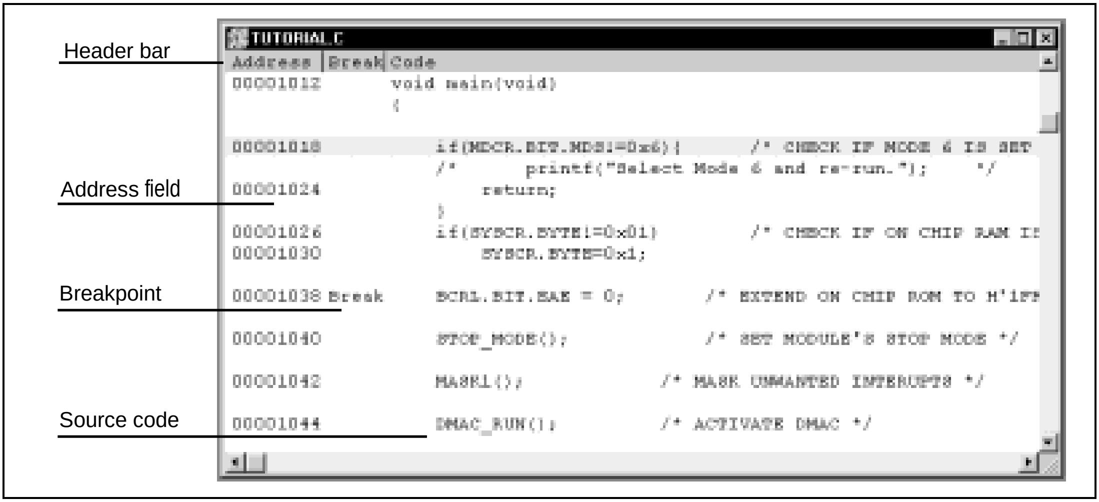
Address: 0

Open on: ○ Address ● Source

Viewing assembler code

Figure 4.1 Open Dialog Box

Select your source file and press **[Open]**, HDI opens a Program window:



```
Address   Break Code
00001012        void main(void)
                {

00001018            if(MDCR.BIT.MDS!=0x6){        /* CHECK IF MODE 6 IS SET
                    /*      printf("Select Mode 6 and re-run.");     */
00001024                return;
                    }
00001026            if(SYSCR.BYTE!=0x01)          /* CHECK IF ON CHIP RAM IS
00001030                SYSCR.BYTE=0x1;

00001038 Break      BCRL.BIT.EAE = 0;         /* EXTEND ON CHIP ROM TO H'1FF

00001040            STOP_MODE();              /* SET MODULE'S STOP MODE */

00001042            MASK1();              /* MASK UNWANTED INTERUPTS */

00001044            DMAC_RUN();           /* ACTIVATE DMAC */
```

Figure 4.2 Program Window (Source Display)

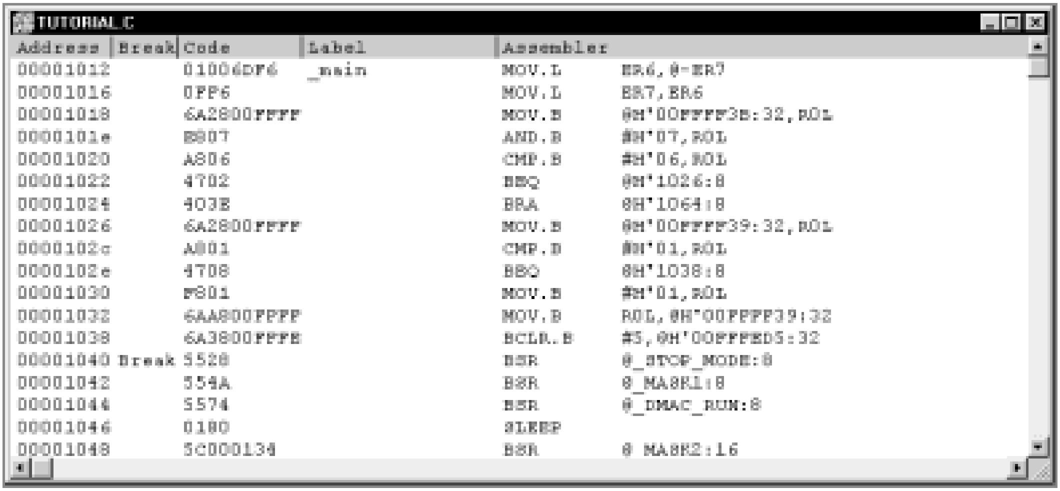
The Program window source display is divided into two areas; the header bar area and the main window area, and split vertically into three columns; Address, Break, and Code. The respective width of each column can be adjusted by dragging the dividing line between each column title in the header bar. 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.

By default the window will display the C/C++ or assembly-language source text of your program.

4.1.1 Viewing Assembly-Language Code

If you do not have a source file, but wish to view code at assembly-language level, select the Address radio button in the “Open on” group of the Open dialog box and enter the address (or symbol) in the Address field. Note that this is the only valid option if you have opened the Program window by the address and have no matching source file.

The Program window will show assembly-language mnemonics (with symbols when available) and the Code column will be replaced with three columns; Code - showing the machine code values, Label - showing labels and symbols, and Assembler - showing the disassembled mnemonics.

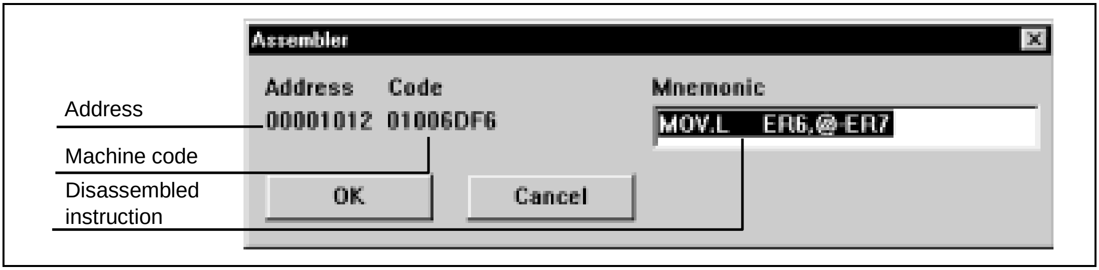


```
TUTORIAL.C
Address  Break Code         Label          Assembler
00001012       01006DF6     _main          MOV.L     ER6,@-ER7
00001016       0FF6                        MOV.L     ER7,ER6
00001018       6A2800FFFF                  MOV.B     @H'00FFFF3B:32,R0L
0000101e       E807                        AND.B     #H'07,R0L
00001020       A806                        CMP.B     #H'06,R0L
00001022       4702                        BEQ       @H'1026:8
00001024       403E                        BRA       @H'1064:8
00001026       6A2800FFFF                  MOV.B     @H'00FFFF39:32,R0L
0000102c       A801                        CMP.B     #H'01,R0L
0000102e       4708                        BEQ       @H'1038:8
00001030       F801                        MOV.B     #H'01,R0L
00001032       6AA800FFFF                  MOV.B     R0L,@H'00FFFF39:32
00001038       6A3800FFFE                  BCLR.B    #5,@H'00FFFED5:32
00001040 Break 5528                        BSR       @_STOP_MODE:8
00001042       554A                        BSR       @_MASK1:8
00001044       5574                        BSR       @_DMAC_RUN:8
00001046       0180                        SLEEP
00001048       5C000134                    BSR       @_MASK2:16
```

Figure 4.3 Program Window (Assembly-Language Display)

4.1.2 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 appear:



Assembler

Address: 00001012

Code: 01006DF6

Mnemonic: MOV.L ER6,@-ER7

Figure 4.4 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.

4.1.3 Displaying Source Code

To change the Program window display to show source code, invoke the pop-up menu by clicking the right mouse button (or alternatively pressing **SHIFT+F10** on the keyboard) and select **[Source]**.

4.1.4 Displaying Mixed Code

You can also display source code lines interspersed with the corresponding assembly-language code in what is called a Mixed display. To change the Program window display to show mixed source and assembly-language code, invoke the pop-up menu by clicking the right mouse button (or alternatively pressing **SHIFT+F10** on the keyboard) and select **[Mixed]**.

4.2 Looking at Symbols

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 symbol information. These symbols (or labels) are text names that represent an address in the program. When you have the Program window in assembly-language format or mixed format, you will see symbols in the Label field on the line of the corresponding address, and in the Assembler field as an instruction’s operand.

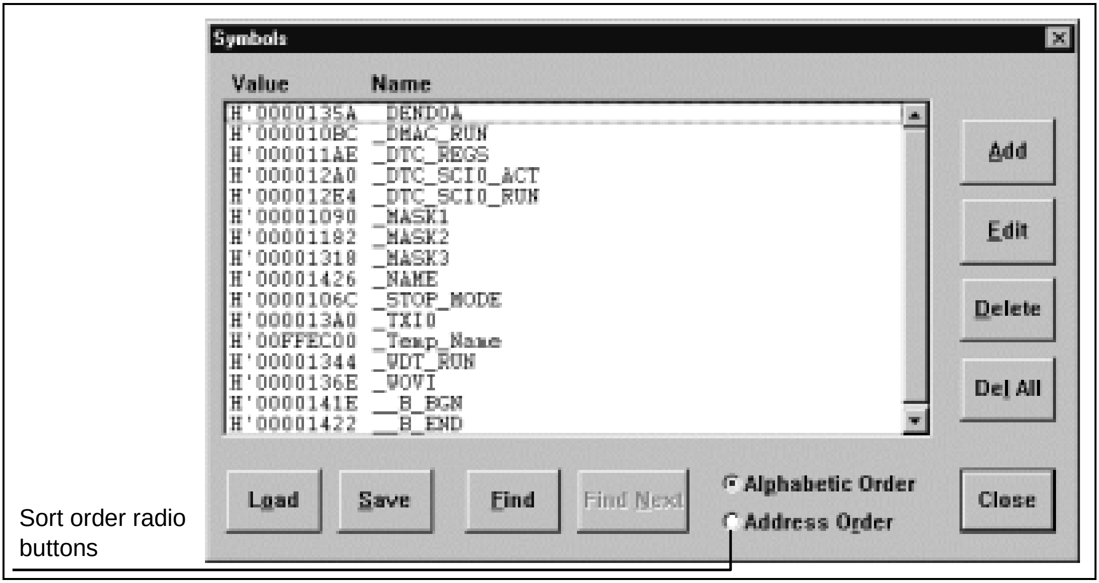
Note: An instruction’s operand is replaced with a symbol if the operand and symbol match. This is done on a simple match of the operand and symbols' values. If two or more symbols have the same value, then the symbol that comes first alphabetically will be displayed.

Wherever you can enter an address or value in HDI you can use a symbol instead.

4.2.1 Listing Symbols

To see a list of all the symbols defined in the current session open the Symbols dialog box by selecting the **[Tools->Symbols...]** menu option.

The Symbols dialog box shows a list of all the symbols defined in HDI.



Value	Name
H'0000135A	_DENDOA
H'000010BC	_DMAC_RUN
H'000011AE	_DTC_REGS
H'000012A0	_DTC_SCI0_ACT
H'000012E4	_DTC_SCI0_RUN
H'00001090	_MASK1
H'00001182	_MASK2
H'00001318	_MASK3
H'00001426	_NAME
H'0000106C	_STOP_MODE
H'000013A0	_TXI0
H'00FFEC00	_Temp_Name
H'00001344	_WDT_RUN
H'0000136E	_WOVI
H'0000141E	__B_BGN
H'00001422	__B_END

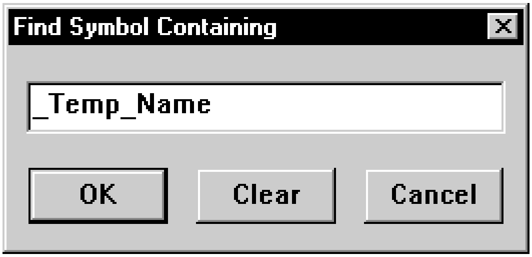
● Alphabetic Order
○ Address Order

Figure 4.5 Symbols Dialog Box

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

4.2.2 Finding a Symbol

To find a particular symbol, click on the **[Find]** button, the Find Symbol Containing dialog box is presented.



_Temp_Name

Figure 4.6 Find Symbol Containing Dialog Box

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

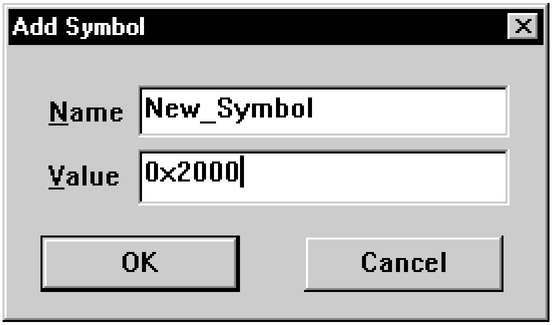
Note: Only the first 255 characters of a symbol are stored, therefore symbols should be unique in these first characters. Symbols are case sensitive.

4.2.3 Finding Again

To find the next symbol in the list containing the text that you entered in the Find Symbol Containing dialog box, click on the **[Find Next]** button.

4.2.4 Adding Symbols

To add a new symbol to the symbol list press the **[Add]** button.



Name: New_Symbol

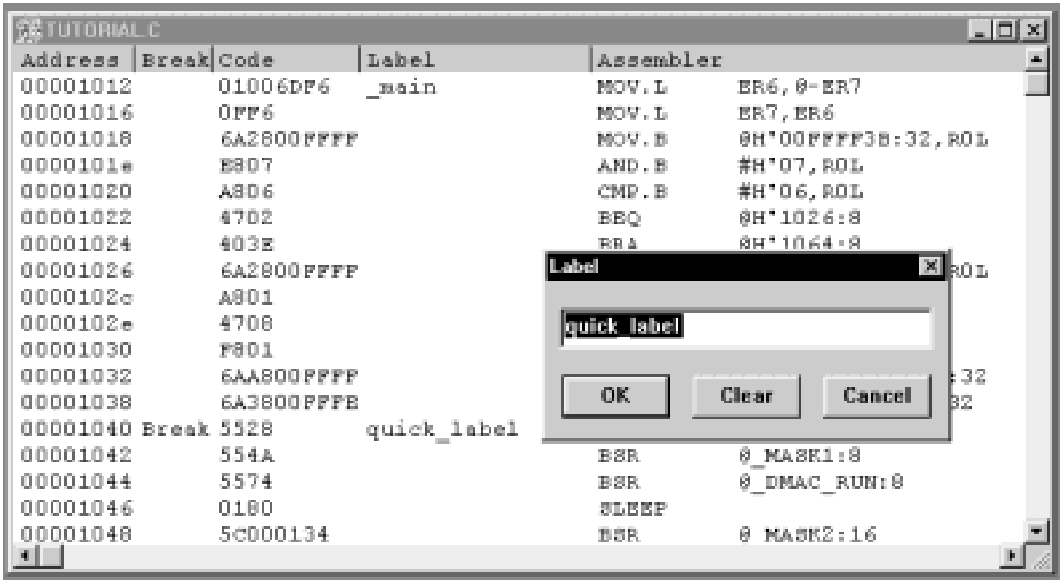
Value: 0x2000

Figure 4.7 Add Symbol Dialog Box

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

4.2.5 Adding a Symbol from the Program Window

You can quickly add a symbol from the Program window (when it is in assembly-language display mode), by double-clicking in the Label column at the address for which you want to assign the symbol. A Label dialog box appears for you to enter the symbol text.



```
Address  Break Code        Label              Assembler
00001012       01006DF6    _main              MOV.L     ER6,@-ER7
00001016       0FF6                           MOV.L     ER7,ER6
00001018       6A2800FFFF                     MOV.B     @H'00FFFF38:32,R0L
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
00001026       6A2800FFFF                                                 R0L
0000102c       A801
0000102e       4708
00001030       F801
00001032       6AA800FFFF                                                 :32
00001038       6A3800FFFE                                                 32
00001040 Break 5528        quick_label
00001042       554A                           BSR       @_MASK1:8
00001044       5574                           BSR       @_DMAC_RUN:8
00001046       0180                           SLEEP
00001048       5C000134                       BSR       @_MASK2:16
```

quick_label

Figure 4.8   Label Dialog Box

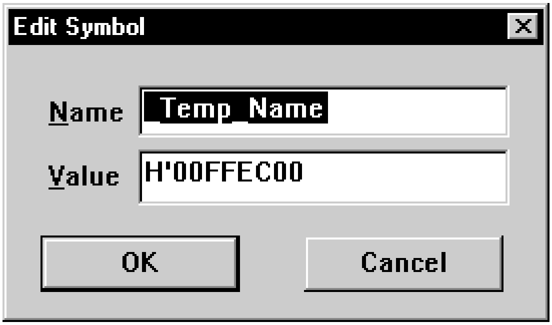
Enter the symbol name text and click **[OK]**, the symbol is added to the symbol list with the address value contained in the Address column of the corresponding line, and the Program window display is updated to show the symbol.

This method can also be used for quickly modifying the text of existing symbols. When you double-click on the symbol 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 symbol list. The Program window display is updated to show the new symbol.

Note: To use added or modified symbols again in later sessions, save the symbols in a file. For details, see *section 4.2.10, Saving a Symbol File*.

4.2.6 Modifying Symbols

Symbol names and values can be modified from the Symbols dialog box. Open this by selecting the **[Tools->Symbols...]** menu option. Select the symbol to modify by clicking on the symbol line in the list display. Clicking on the **[Edit]** button will open the Edit Symbol dialog box:



Edit Symbol

Name: _Temp_Name

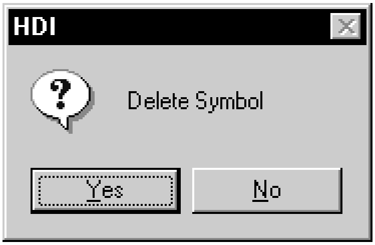
Value: H'00FFEC00

Figure 4.9 Edit Symbol Dialog Box

You can then edit the symbol name and value. Press **[OK]** to save the modified version in the symbol list. The list display is updated to show the new symbol. When an overloaded function or a class name is entered in the Value field, the Select Function dialog box appears for you to select a function. For details, refer to *section 10, Selecting Functions*.

4.2.7 Deleting Symbols

To delete a symbol from the symbol list, select the symbol to delete from the Symbols dialog box by clicking on the symbol line in the list display, and click on the **[Delete]** button. A confirmation message box appears:



HDI

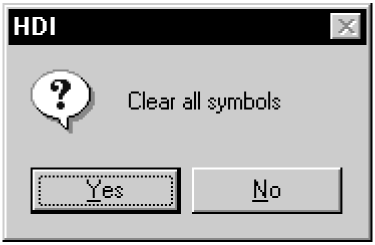
Delete Symbol

Figure 4.10 Message Box for Confirming Symbol Deletion

If you click on the **[Yes]** button the symbol is removed from the HDI system’s symbol table and the list display is updated.

4.2.8 Deleting All Symbols

To delete all the symbols from the symbol list, click on the **[Del All]** button. A confirmation message box appears:



Clear all symbols

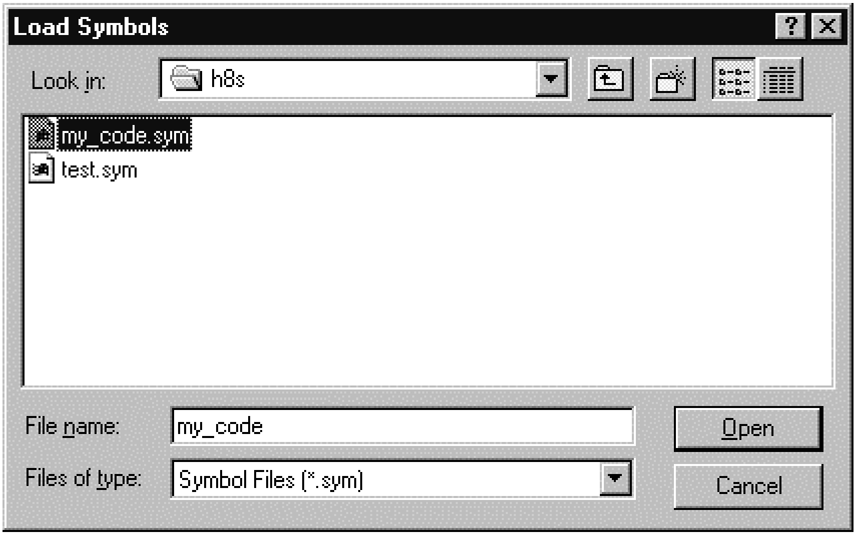
Figure 4.11 Message Box for Confirming All Symbol Deletion

If you click on the **[Yes]** button all the symbols are removed from the HDI system's symbol table and the list display will be cleared.

4.2.9 Loading a Symbol File

Although HDI will automatically extract symbols from a debug object file when it is loaded, there may be times when you want to load extra symbols. This could be the case if you are debugging code without debug information. Having to enter these symbols manually using the Add Symbols dialog box would be extremely tedious, but fortunately HDI allows you to load symbols from an external symbol file.

To load a symbol file into HDI, open the Symbols dialog box by selecting the **[Tools->Symbols...]** menu option and click the **[Load]** button. The Load Symbols dialog box appears:



Look in: h8s

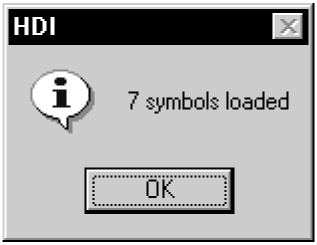
my_code.sym
test.sym

File name: my_code

Files of type: Symbol Files (*.sym)

Figure 4.12 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 loading is complete a message box shows how many symbols have been loaded:



7 symbols loaded

Figure 4.13 Number of Loaded Symbols

4.2.10 Saving a Symbol File

To save a symbol file from HDI, open the Symbols dialog box by selecting the **[Tools->Symbols...]** menu option and click the **[Save]** button. The Save Symbols dialog box appears. The dialog box operates like a standard Windows® save file dialog box. Enter the name for the file in the File name: field and click **[Open]** to start saving. The standard file extension for symbol files is “.sym”.

4.2.11 Symbol File Format

In order for HDI to be able to understand and decode the symbol file correctly, the file must be formatted in a specific way:

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
```

4.3 Looking at a Specific Address

When you are looking at your program in the Program 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 of the Program window, the Set Address dialog box appears:

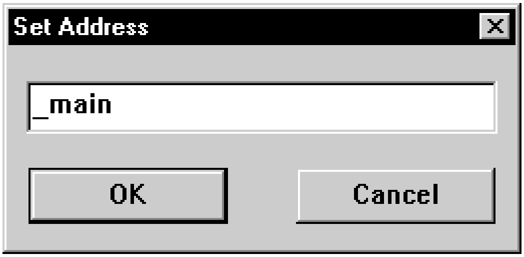


Figure 4.14 Set Address Dialog Box

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 Program 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 appears for you to select a function. For details, refer to *section 10, Selecting Functions*.

If the new address is in another source file, a new Program 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, the Program window shows assembly-language code.

If the new address is in a source file that already has a Program window open, that window is brought to the front and updated to show the code at the new address.

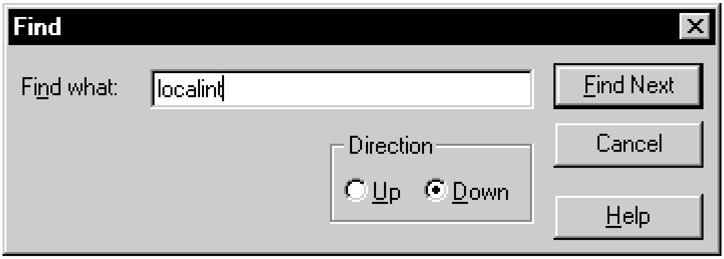
4.3.1 Looking at the Current Program Counter Address

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 invoke the Set Address dialog box and enter the expression "#PC", the Program 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 Program window using the find option. To do this invoke the pop-up menu by clicking the right mouse button (or alternatively pressing **SHIFT+F10** on the keyboard) and select **[Find…]**.

The Find dialog box is displayed:



Find what: localint

Direction: ○ Up ● Down

Figure 4.15 Find Dialog Box

Enter the text that you wish to find and click **[Find Next]** or press **ENTER**. The Program 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**.

4.5 Looking at a Text File

HDI will automatically show source files for your program if the debug object file provides it with source file information. However there will be occasions when you want to look at source or text files for which the compiler does not include this information e.g. include files, data tables or project documentation files. HDI provides this feature with the Text window.

4.5.1 Opening a Text Window

To open a Text window, select **[View->Text Window...]** menu option.

The Open dialog box appears:

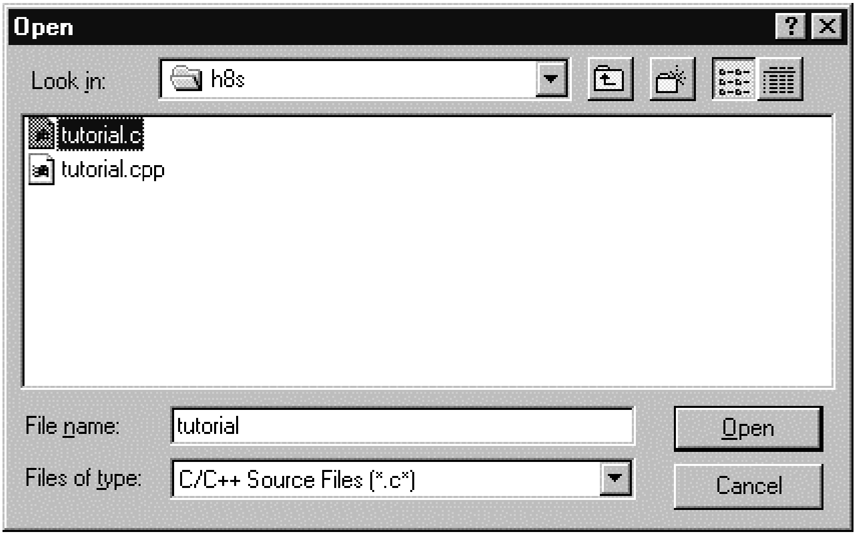
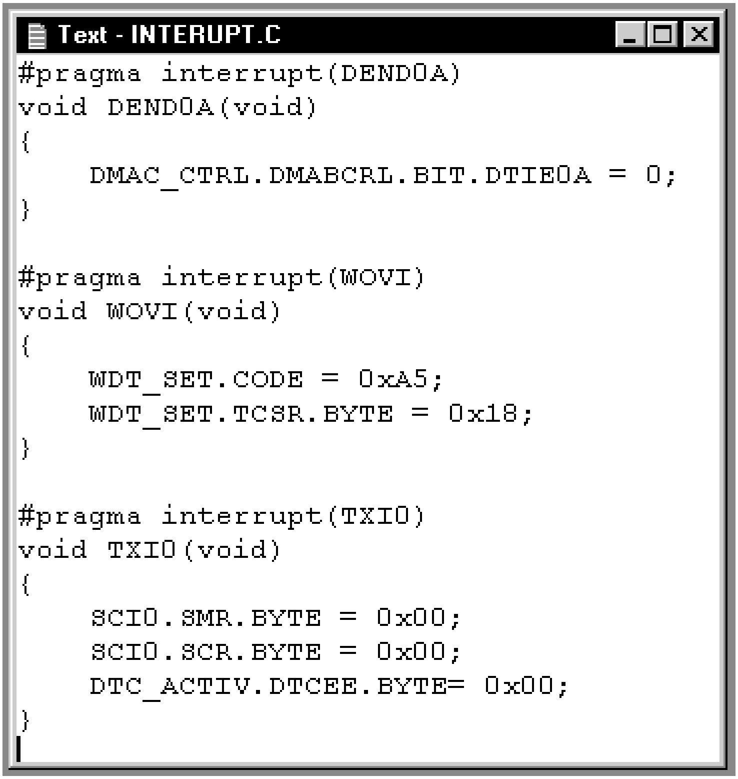


Figure 4.16   Open Dialog Box

The dialog box operates like a standard Windows® open file dialog box; select a text file and click **[Open]** to open it. A Text window will open and the file will be displayed.



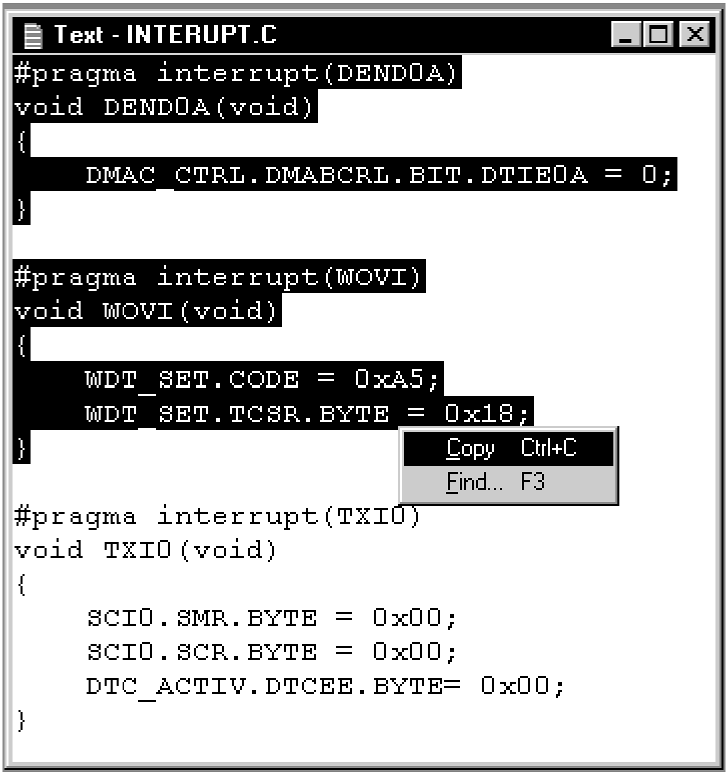
```
#pragma interrupt(DEND0A)
void DEND0A(void)
{
    DMAC_CTRL.DMABCRL.BIT.DTIE0A = 0;
}

#pragma interrupt(WOVI)
void WOVI(void)
{
    WDT_SET.CODE = 0xA5;
    WDT_SET.TCSR.BYTE = 0x18;
}

#pragma interrupt(TXI0)
void TXI0(void)
{
    SCI0.SMR.BYTE = 0x00;
    SCI0.SCR.BYTE = 0x00;
    DTC_ACTIV.DTCEE.BYTE= 0x00;
}
```

Figure 4.17 Text Window

You can move around in the display window using the scroll bars or via the keyboard using the cursor and **Page Up/Down** keys.

4.5.2 Copying Text

To copy text from a Text window into the clipboard, first select the text by dragging the mouse over it. Then invoke the local pop-up menu by clicking the right mouse button:



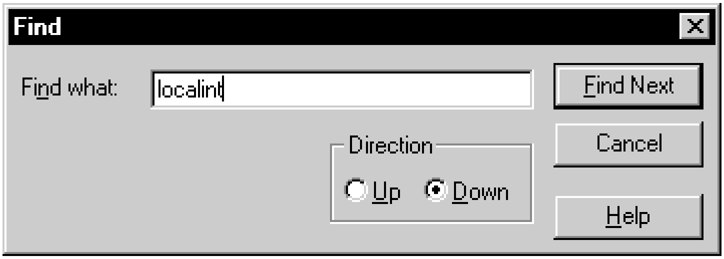
```
#pragma interrupt(DEND0A)
void DEND0A(void)
{
     DMAC_CTRL.DMABCRL.BIT.DTIE0A = 0;
}

#pragma interrupt(WOVI)
void WOVI(void)
{
     WDT_SET.CODE = 0xA5;
     WDT_SET.TCSR.BYTE = 0x18;
}

#pragma interrupt(TXI0)
void TXI0(void)
{
     SCI0.SMR.BYTE = 0x00;
     SCI0.SCR.BYTE = 0x00;
     DTC_ACTIV.DTCEE.BYTE= 0x00;
}
```

Figure 4.18 Copying Text

Click on **[Copy]** and the selected text is copied to the clipboard.

4.5.3 Finding Text

You can search for a particular text string in the Text window using the find option. Invoke the local pop-up menu by clicking the right mouse button, click on **[Find...]** and you will be presented with the Find dialog box.



Find what: localint

Direction ○ Up ● Down

Figure 4.19 Find Dialog Box

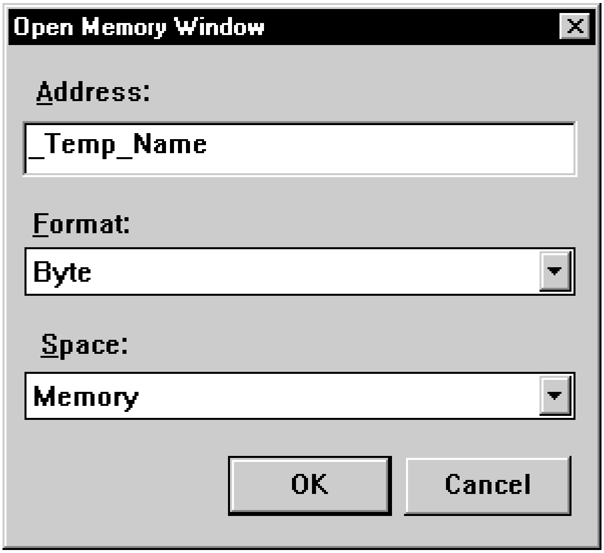
Enter the text that you wish to search for in the edit box and click on **[Find Next]**. HDI searches through the Text window and stops at the first occurrence of the text that you specified. To find the next occurrence, click **[Find Next]** again.

Section 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, open a Memory window, by selecting the **[View->Memory Window]** menu option, or clicking the Memory Window toolbar button [] if it is visible. You will be presented with an Open Memory Window dialog box:



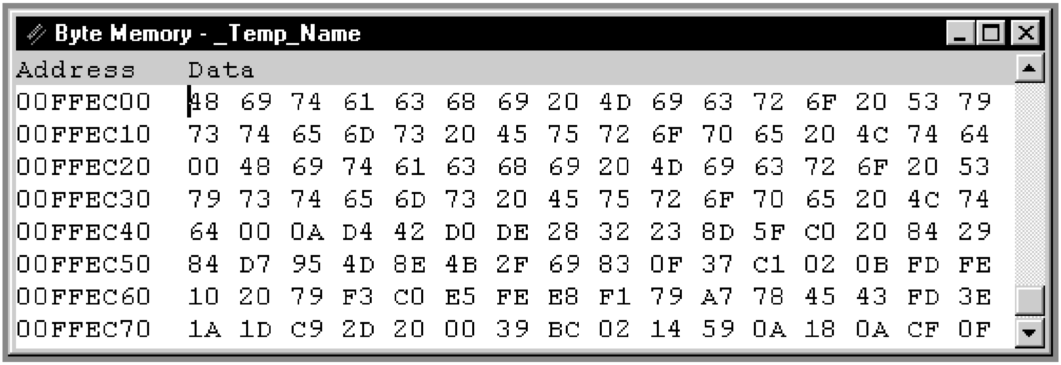
Address: _Temp_Name

Format: Byte

Space: Memory

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 disappears and a Memory window opens:



```
Address    Data
00FFEC00   48 69 74 61 63 68 69 20 4D 69 63 72 6F 20 53 79
00FFEC10   73 74 65 6D 73 20 45 75 72 6F 70 65 20 4C 74 64
00FFEC20   00 48 69 74 61 63 68 69 20 4D 69 63 72 6F 20 53
00FFEC30   79 73 74 65 6D 73 20 45 75 72 6F 70 65 20 4C 74
00FFEC40   64 00 0A D4 42 D0 DE 28 32 23 8D 5F C0 20 84 29
00FFEC50   84 D7 95 4D 8E 4B 2F 69 83 0F 37 C1 02 0B FD FE
00FFEC60   10 20 79 F3 C0 E5 FE E8 F1 79 A7 78 45 43 FD 3E
00FFEC70   1A 1D C9 2D 20 00 39 BC 02 14 59 0A 18 0A CF 0F
```

Figure 5.2 Memory Window (Bytes)

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

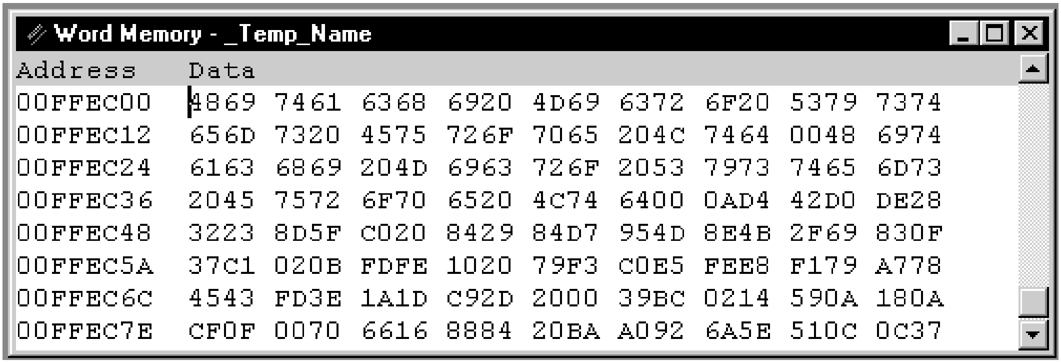
5.1.1 Displaying Memory as Bytes

To display memory as bytes, invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**).

Select the **[Byte]** menu option and the display will be updated to show the area of memory as individual bytes, as shown above.

5.1.2 Displaying Memory as Words

To display memory as words, invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**). Select the **[Word]** menu option and the display will be updated to show the area of memory as 16 bit words:

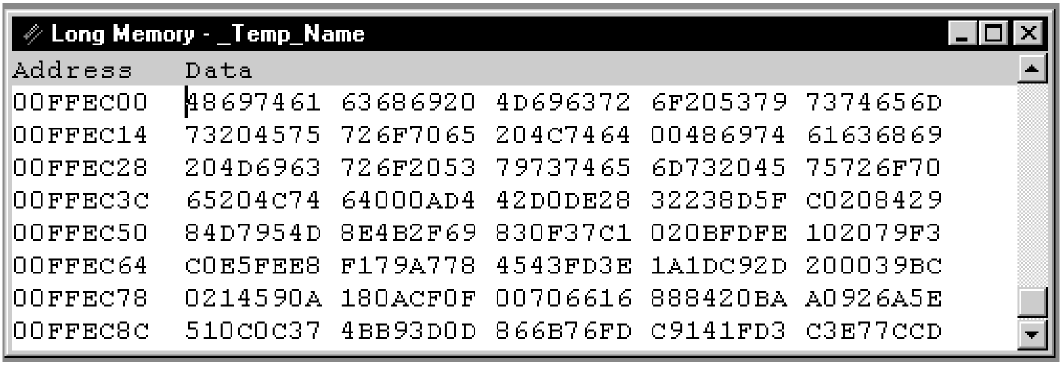


```
Address    Data
00FFEC00   4869 7461 6368 6920 4D69 6372 6F20 5379 7374
00FFEC12   656D 7320 4575 726F 7065 204C 7464 0048 6974
00FFEC24   6163 6869 204D 6963 726F 2053 7973 7465 6D73
00FFEC36   2045 7572 6F70 6520 4C74 6400 0AD4 42D0 DE28
00FFEC48   3223 8D5F C020 8429 84D7 954D 8E4B 2F69 830F
00FFEC5A   37C1 020B FDFE 1020 79F3 C0E5 FEE8 F179 A778
00FFEC6C   4543 FD3E 1A1D C92D 2000 39BC 0214 590A 180A
00FFEC7E   CF0F 0070 6616 8884 20BA A092 6A5E 510C 0C37
```

Figure 5.3 Memory Window (Words)

5.1.3 Displaying Memory as Long words

To display memory as long words, invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**). Select the **[Long]** menu option and the display will be updated to show the area of memory as 32 bit long words:

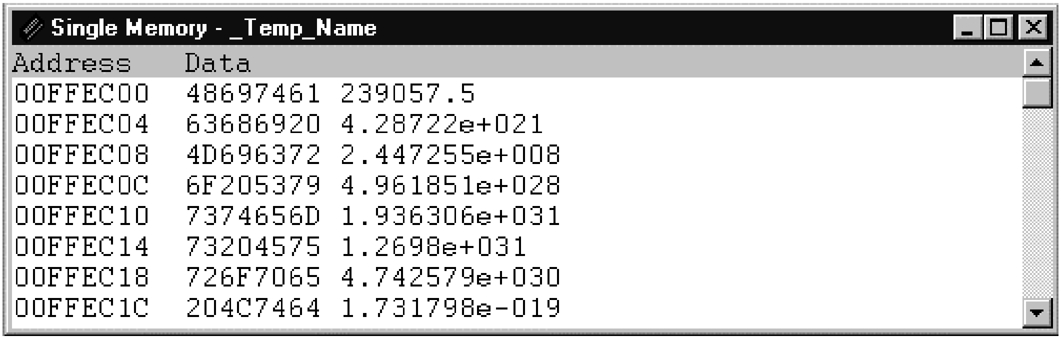


```
Address     Data
00FFEC00    48697461 63686920 4D696372 6F205379 7374656D
00FFEC14    73204575 726F7065 204C7464 00486974 61636869
00FFEC28    204D6963 726F2053 79737465 6D732045 75726F70
00FFEC3C    65204C74 64000AD4 42D0DE28 32238D5F C0208429
00FFEC50    84D7954D 8E4B2F69 830F37C1 020BFDFE 102079F3
00FFEC64    C0E5FEE8 F179A778 4543FD3E 1A1DC92D 200039BC
00FFEC78    0214590A 180ACF0F 00706616 888420BA A0926A5E
00FFEC8C    510C0C37 4BB93D0D 866B76FD C9141FD3 C3E77CCD
```

Figure 5.4 Memory Window (Long words)

5.1.4 Displaying Memory as Single-Precision Floating Point

To display memory as single-precision floating-point data, invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**). Select the **[Single float]** menu option and the display will be updated to show the area of memory as single-precision floating-point data:

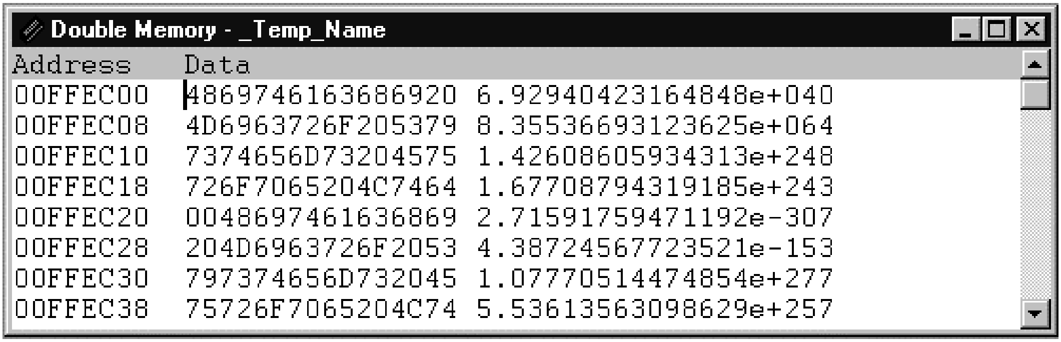


```
Address     Data
00FFEC00    48697461 239057.5
00FFEC04    63686920 4.28722e+021
00FFEC08    4D696372 2.447255e+008
00FFEC0C    6F205379 4.961851e+028
00FFEC10    7374656D 1.936306e+031
00FFEC14    73204575 1.2698e+031
00FFEC18    726F7065 4.742579e+030
00FFEC1C    204C7464 1.731798e-019
```

Figure 5.5 Memory Window (Single-Precision Floating Point)

5.1.5 Displaying Memory as Double-Precision Floating Point

To display memory as double-precision floating-point data, invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**). Select the **[Double float]** menu option and the display will be updated to show the area of memory as double-precision floating-point data:

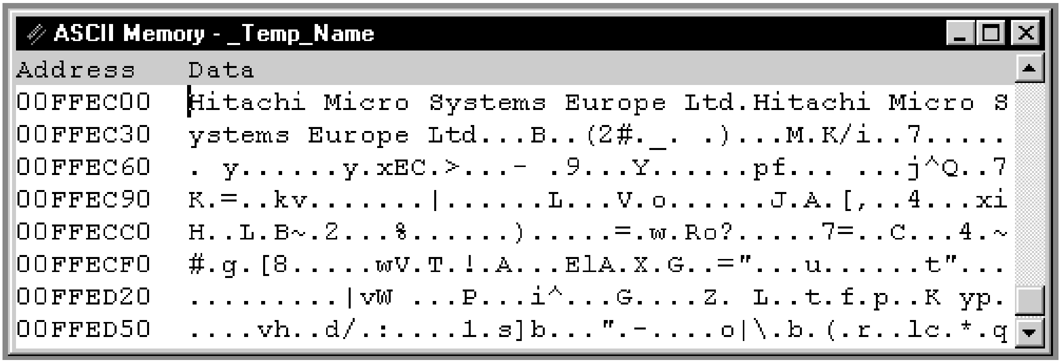


```
Address   Data
00FFEC00  4869746163686920 6.92940423164848e+040
00FFEC08  4D6963726F205379 8.35536693123625e+064
00FFEC10  7374656D73204575 1.42608605934313e+248
00FFEC18  726F7065204C7464 1.67708794319185e+243
00FFEC20  0048697461636869 2.71591759471192e-307
00FFEC28  204D6963726F2053 4.38724567723521e-153
00FFEC30  797374656D732045 1.07770514474854e+277
00FFEC38  75726F7065204C74 5.53613563098629e+257
```

Figure 5.6 Memory Window (Double-Precision Floating Point)

5.1.6 Displaying Memory as ASCII

To display memory as ASCII characters, invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**). Select the **[ASCII]** menu option and the display will be updated to show the area of memory as ASCII characters:



```
Address   Data
00FFEC00  Hitachi Micro Systems Europe Ltd.Hitachi Micro S
00FFEC30  ystems Europe Ltd...B..(2#._. .)...M.K/i..7.....
00FFEC60  . y......y.xEC.>...- .9...Y.....pf... ...j^Q..7
00FFEC90  K.=..kv.......|......L...V.o......J.A.[,..4...xi
00FFECC0  H..L.B~.2...%......).....=.w.Ro?.....7=..C...4.~
00FFECF0  #.g.[8.....wV.T.!.A...ElA.X.G..="...u......t"...
00FFED20  .........|vW ...P...i^...G....Z. L..t.f.p..K yp.
00FFED50  ....vh..d/.:....1.s]b...".-....o|\.b.(.r..lc.*.q
```

Figure 5.7 Memory Window (ASCII)

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 invoked either from the pop-up menu or by double clicking in the Address column. Invoke the pop-up menu by clicking the right mouse button, (or pressing **SHIFT+F10**) and click on **[Set Address]**:

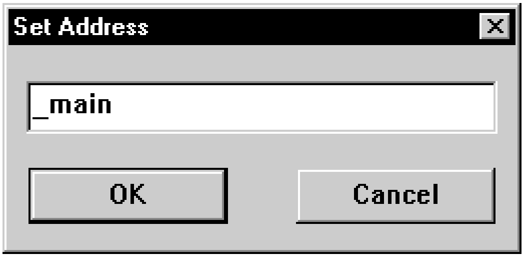


Figure 5.8 Set Address Dialog Box

The Set Address dialog box appears; enter the new address value. Click **[OK]** or press **ENTER**, the dialog box disappears 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 appears for you to select a function. For details, refer to *section 10, Selecting Functions*.

5.2 Modifying Memory Contents

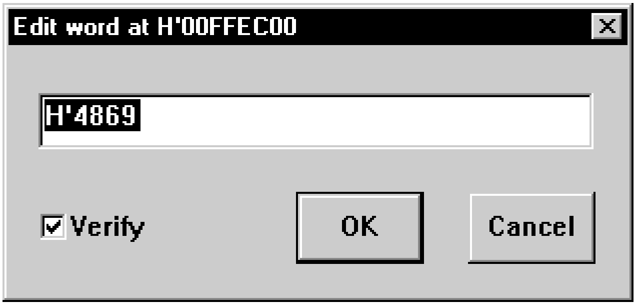
There are two ways that you can change the contents of memory at an address; the quick edit method that allows you to enter values by typing directly into the window, but is limited to hexadecimal values only, and the full edit method that requires you to enter values via a dialog box, but allows you to enter values in any simple or complex 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. 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 appears:



Edit word at H'00FFEC00

H'4869

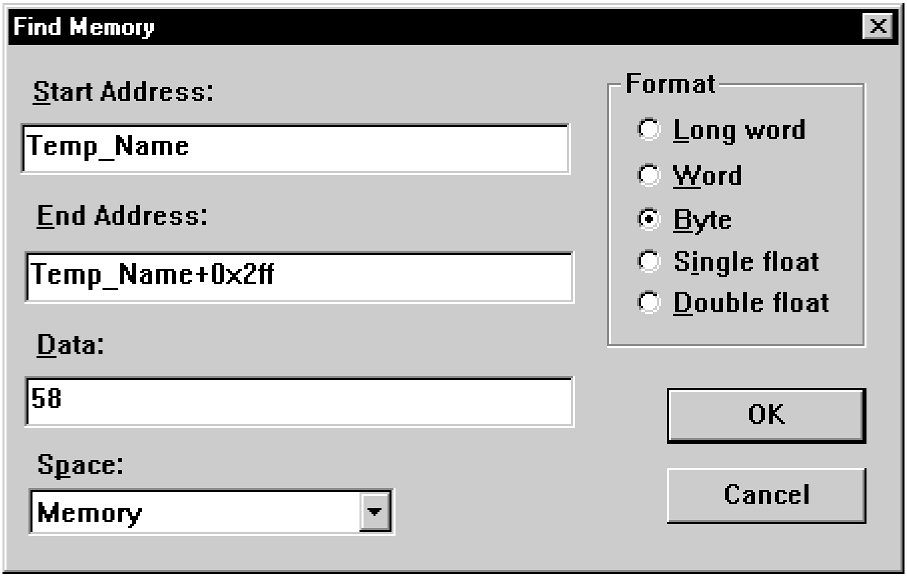
☑ Verify

Figure 5.9 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.

5.3 Finding a Value in Memory

You can search for a value in memory using the Find feature. To find the value, invoke the pop-up menu by clicking the right mouse button, (or pressing **SHIFT+F10**) and click on **[Find]**. The Find Memory dialog box appears:



Find Memory

Start Address: Temp_Name

End Address: Temp_Name+0x2ff

Data: 58

Space: Memory

Format

○ Long word

○ Word

● Byte

○ Single float

○ Double float

Figure 5.10 Find Memory Dialog Box

Enter the start and end addresses of the range in which to search and the data value to search for. Select the search format Long word/Word/Byte/Single float/Double float and click **[OK]** or press **ENTER**. The dialog box closes and HDI searches the range for the specified data. If the data is found, a message showing the address at which the data has been found is displayed on the Status bar:

Memory pattern found at H'5.

Figure 5.11 Address Where Data Found

otherwise the following message will be displayed:

Pattern not found.

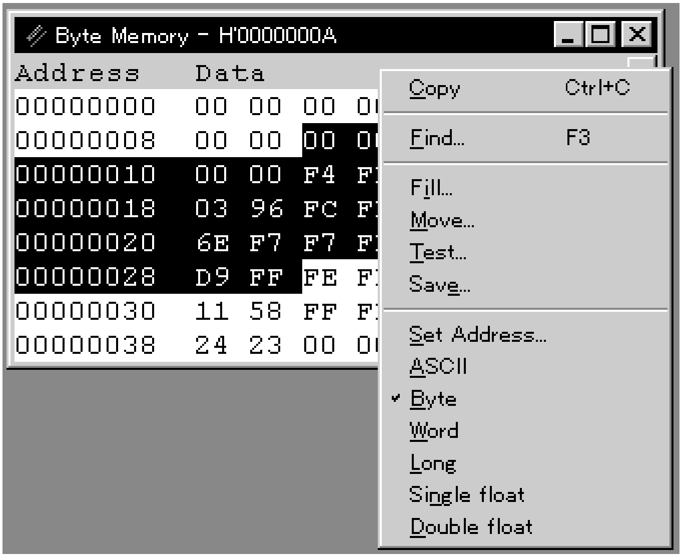
Figure 5.12 Address Where Data Not Found

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 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.



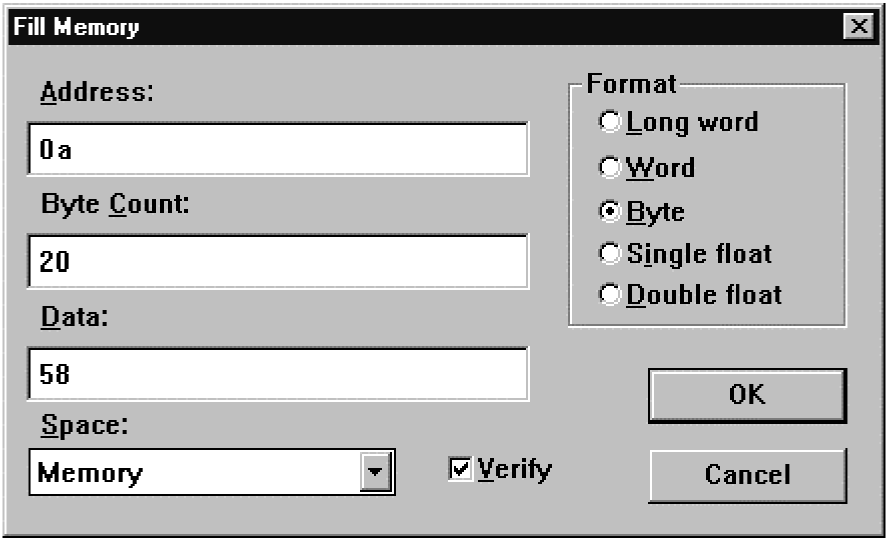
```
Address    Data
00000000   00  00  00  0(
00000008   00  00  00  0(
00000010   00  00  F4  F
00000018   03  96  FC  F
00000020   6E  F7  F7  F
00000028   D9  FF  FE  F
00000030   11  58  FF  F
00000038   24  23  00  0(
```

Figure 5.13   Selecting a Memory Range

If the memory address range is larger than or outside the Memory window, then you can enter the start addresses and byte count in the respective fields of Fill Memory dialog box.

5.4.2 Filling the Range

To fill the range, invoke the pop-up menu by clicking the right mouse button, (or pressing **SHIFT+F10**) and click on **[Fill]**. The Fill Memory dialog box appears:



Fill Memory

Address: 0a

Byte Count: 20

Data: 58

Space: Memory

Format

○ Long word
○ Word
● Byte
○ Single float
○ Double float

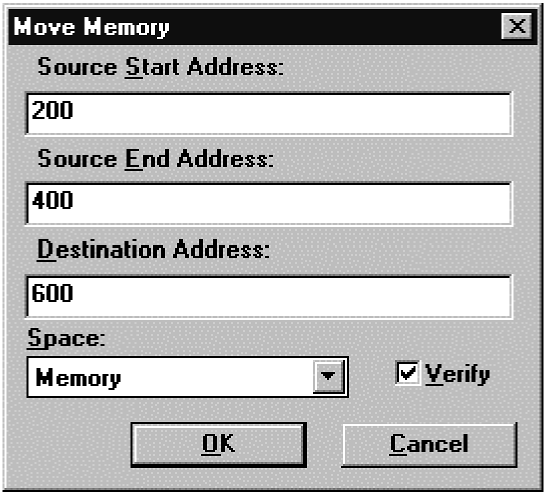
☑ Verify

Figure 5.14 Fill Memory Dialog Box

When an address range has been selected in the Memory window, the specified start address and byte count will be displayed in the Address and Byte Count fields. Select the format by clicking on the relevant radio button in the Format group and enter the data value in the Data field. Click the **[OK]** button or press **ENTER**, the dialog box closes and the new value is written into the memory range.

5.5 Moving an Area of Memory

You can move an area of memory in the address space using the memory move feature. Select a memory range (see *section 5.4.1, Selecting a memory range*), invoke the pop-up menu and click on **[Move]**. The Move Memory dialog box appears:



Source Start Address: 200

Source End Address: 400

Destination Address: 600

Space: Memory

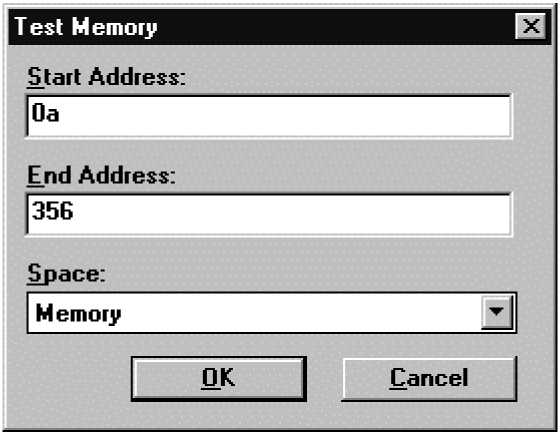
☑ Verify

Figure 5.15 Move Memory Dialog Box

The source start address and end address specified in the Memory window will be displayed in the Source Start Address and Source End Address fields. Enter the destination start address in the Destination Address 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

You can test an area of memory in the address space using the memory test feature. Select a memory range (see *section 5.4.1*, *Selecting a memory range*), invoke the pop-up menu and click on **[Test]**. The Test Memory dialog box appears:



Start Address: 0a

End Address: 356

Space: Memory

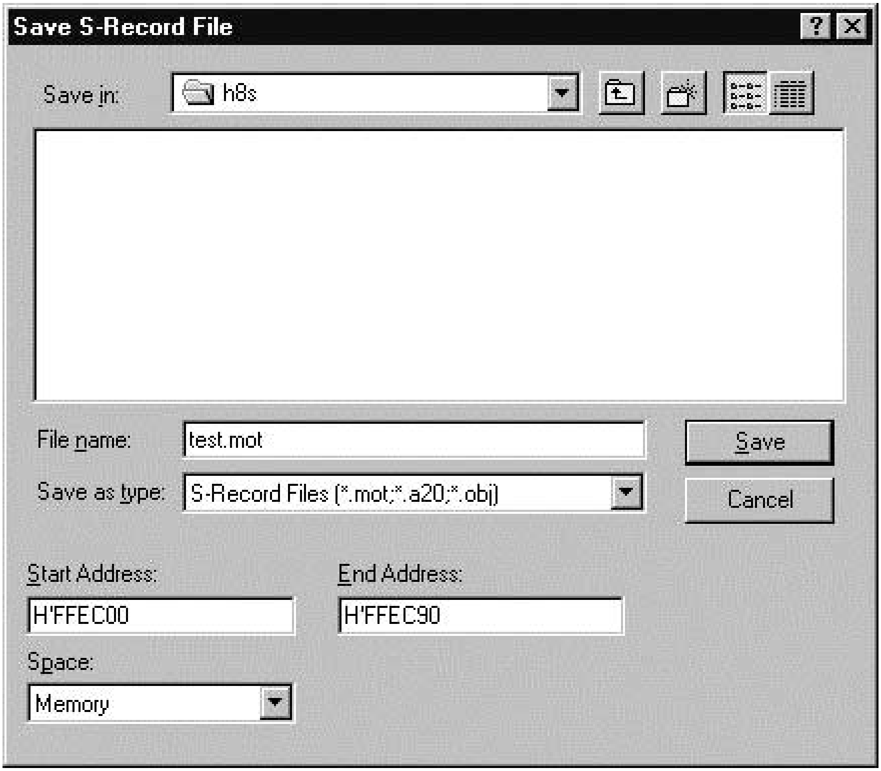
Figure 5.16 Test Memory Dialog Box

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

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.

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 S-Record File dialog box by selecting the **[File->Save Memory...]** menu option:



Save in: h8s

File name: test.mot

Save as type: S-Record Files (*.mot;*.a20;*.obj)

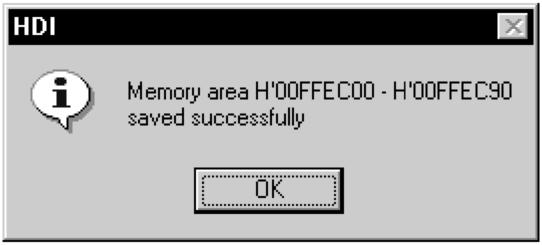
Start Address: H'FFEC00

End Address: H'FFEC90

Space: Memory

Figure 5.17 Save S-Record File Dialog Box

Enter the start and end addresses of the memory block that you wish to save and a file name. Directory and drive navigation is possible using the standard Windows® file open dialog box controls, to the right of the file list. 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 appears:

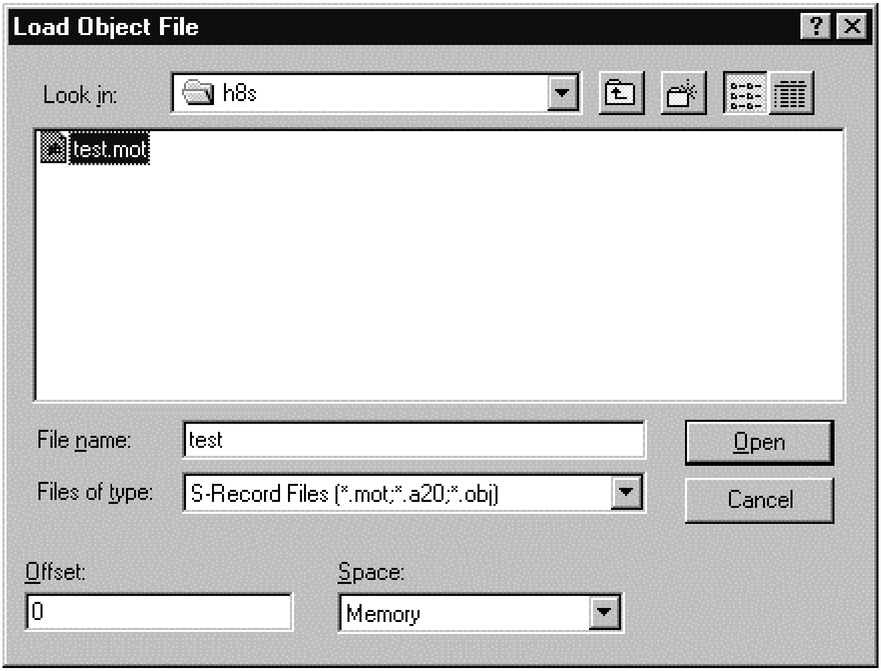


Memory area H'00FFEC00 - H'00FFEC90 saved successfully

Figure 5.18 Message Box for Confirming File Save Completion

5.8 Loading an Area of Memory

You can load a program to an area of memory from a disk file using the load program feature. Since programs are loaded as Motorola S-Record files this feature can be used to load any data values into memory at the addresses specified by the records in the file, e.g. a previously saved memory block. Open the Load Object File dialog box by selecting the **[File->Load Program...]** menu option:



Look in: h8s

test.mot

File name: test

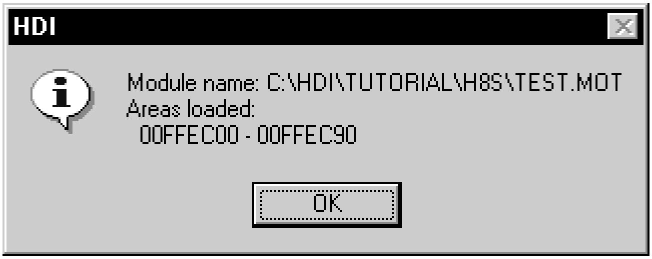
Files of type: S-Record Files (*.mot;*.a20;*.obj)

Offset: 0

Space: Memory

Figure 5.19 Load Object File 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 file is loaded into memory. When the file load is complete a confirmation message box appears:



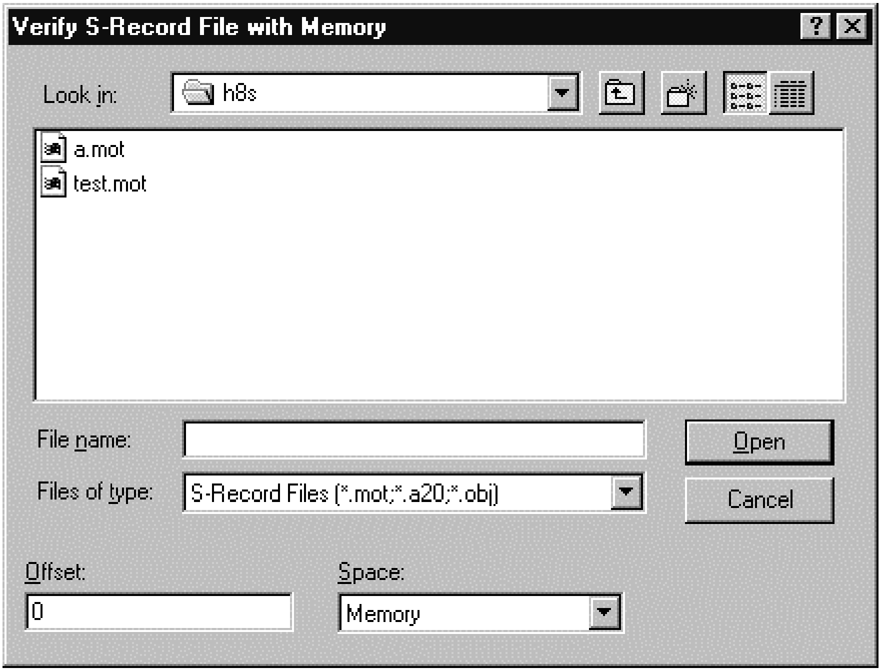
HDI

Module name: C:\HDI\TUTORIAL\H8S\TEST.MOT
Areas loaded:
00FFEC00 - 00FFEC90

Figure 5.20 Message Box for Confirming File Load Completion

5.9 Verifying an Area of Memory

You can verify an area of memory in the address space against a disk file using the verify memory feature. This can be useful for checking the integrity of data currently in memory compared to a previously saved block of memory in a file. Open the Verify S-Record File with Memory dialog box by selecting the **[File->Verify Memory]** menu option:



Verify S-Record File with Memory

Look in: h8s

a.mot
test.mot

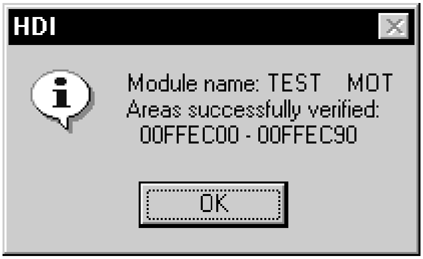
File name:
Files of type: S-Record Files (*.mot;*.a20;*.obj)

Offset: 0

Space: Memory

Figure 5.21 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 **[OK]** button or press **ENTER**, the dialog box closes and the file is verified. When the file verification is complete a confirmation message box appears:



HDI

Module name: TEST   MOT
Areas successfully verified:
 00FFEC00 - 00FFEC90

Figure 5.22   Message Box for Confirming Verification Completion

Section 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

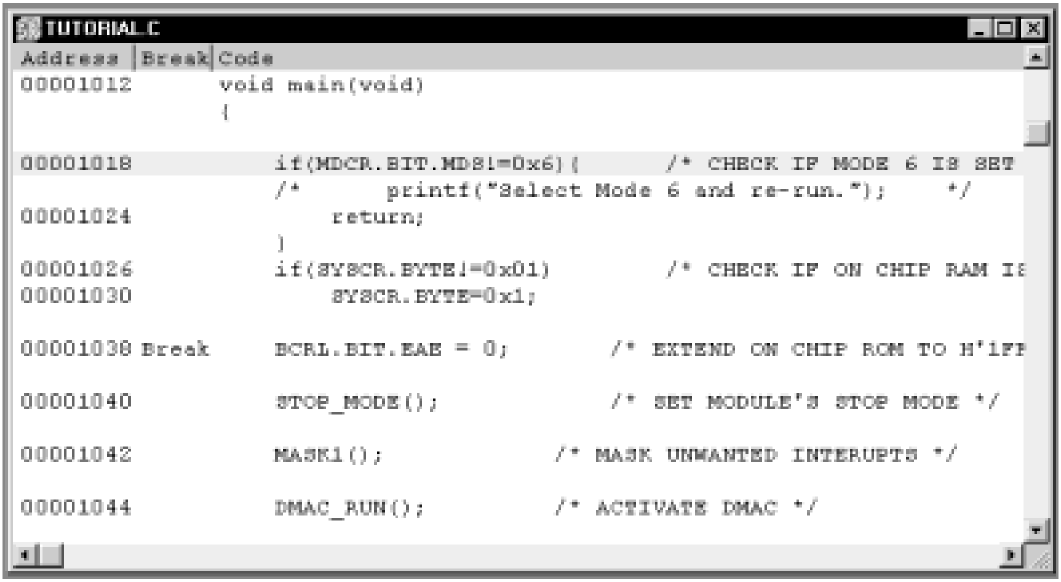
To reset your user system and run your program from the Reset Vector address, click the Go Reset toolbar button [], if it is visible, or select the **[Run->Go Reset]** menu option.

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

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 Program window that corresponds 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.



```
Address  |Break|Code
00001012        void main(void)
                {

00001018            if(MDCR.BIT.MDS!=0x6){        /* CHECK IF MODE 6 IS SET
                    /*      printf("Select Mode 6 and re-run.");     */
00001024                return;
                    }
00001026            if(SYSCR.BYTE!=0x01)          /* CHECK IF ON CHIP RAM IS
00001030                SYSCR.BYTE=0x1;

00001038 Break      BCRL.BIT.EAE = 0;         /* EXTEND ON CHIP ROM TO H'1FF

00001040            STOP_MODE();              /* SET MODULE'S STOP MODE */

00001042            MASK1();              /* MASK UNWANTED INTERUPTS */

00001044            DMAC_RUN();           /* ACTIVATE DMAC */
```

Figure 6.1   Highlighted Line Corresponding to PC Address

To continue running from the current PC address click the Continue toolbar button [ ], if it is visible, or select the **[Run->Go]** menu option.

6.3 Running to the Cursor

Sometimes as you are going through your application you may want to run only a small section of code, that would require many single steps to execute. In this case it would be useful to be able to run to a particular point. You can do this using the Go To Cursor feature.

Using Go To Cursor

1. Make sure that the Program window is open showing the address at which you wish to stop.
2. Position the cursor on the address at which you wish to stop by either clicking in the Address field or using the cursor keys.
3. Invoke the pop-up menu by clicking the right mouse button, (or pressing **SHIFT+F10**).
4. Select the **[Go To Cursor]** menu option. The debugging platform will run your code 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, code execution can be stopped by pressing **Esc**, selecting the **[Run->Halt program]** menu option, or clicking on the 'Stop' 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 Program 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. If the Program window display is in source mode, then a step operation will step a single source line. If the Program window display is in assembly-language or mixed mode, 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 [], if it is visible, or select 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 [], if it is visible, or select the **[Run->Step Over]** menu option.

6.6 Stepping Out of a Function

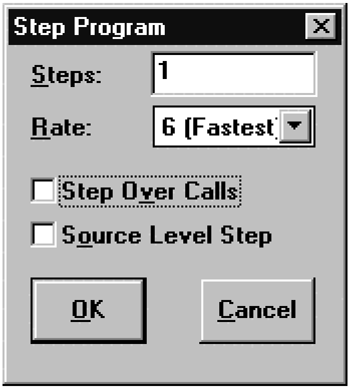
During debugging, there are occasions when you may have entered a function, finished stepping through the instructions that you want to examine and would like to return to the calling function without tediously stepping through all the remaining code in the function. Or alternatively (and perhaps more 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 [], if it is visible, or select 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. It is invoked by selecting the **[Run->Step...]** menu option.

The Step Program dialog box is displayed:



Step Program

Steps: 1

Rate: 6 (Fastest

□ Step Over Calls

□ Source Level Step

Figure 6.2 Step Program Dialog Box

Enter the number of steps in the Steps field and select whether you want to step over function calls by the Step Over Calls 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.

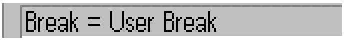
Section 7 Stopping Your Program

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 [STOP] (a red STOP sign), and when the program has stopped it is disabled [STOP] (the STOP sign is grayed out). To stop the program click on the Halt toolbar button, if it is visible, or select the **[Run->Halt Program]** menu option.

Your program’s execution is halted, with the following message displayed on the status bar. HDI will then update any open windows.



Break = User Break

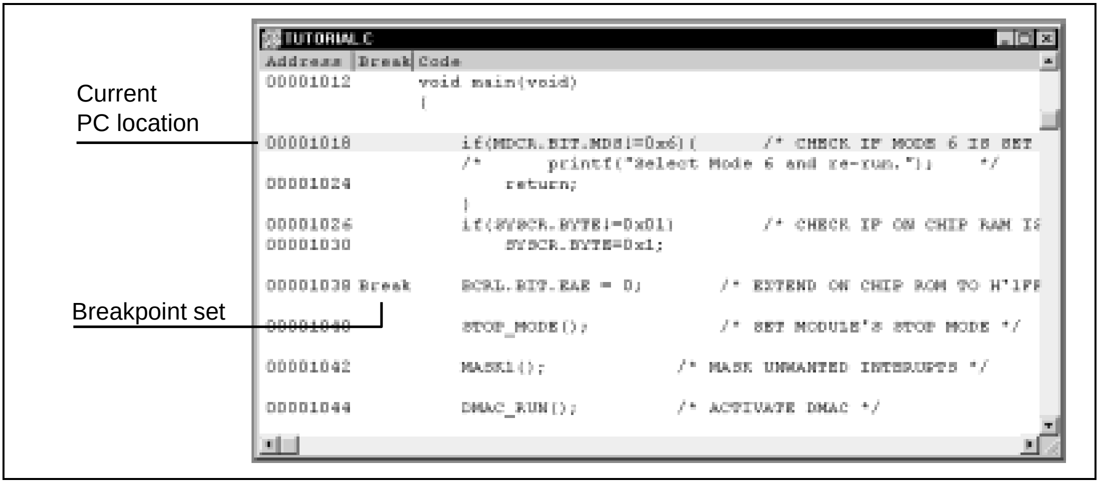
Figure 7.1 Execution Halted by Clicking Halt Button

7.2 Program 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

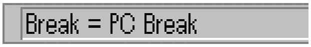
1. Make sure that the Program window is open at the place you want to set a program (PC) breakpoint.
2. Double-click in the Break column of the line at which you want the program to stop.
3. You will see the word ‘Break’ appear in the column to indicate that a program (PC) breakpoint has been set.



```
Address  Break Code
00001012       void main(void)
               {

00001018           if(MDCR.BIT.MDS!=0x6){       /* CHECK IF MODE 6 IS SET
                   /*      printf("Select Mode 6 and re-run.");     */
00001024               return;
                   }
00001026           if(SYSCR.BYTE!=0x01)         /* CHECK IF ON CHIP RAM IS
00001030               SYSCR.BYTE=0x1;

00001038 Break     BCRL.BIT.EAE = 0;          /* EXTEND ON CHIP ROM TO H'1FF

00001040           STOP_MODE();               /* SET MODULE'S STOP MODE */

00001042           MASK1();                /* MASK UNWANTED INTERRUPTS */

00001044           DMAC_RUN();             /* ACTIVATE DMAC */
```

Figure 7.2 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 Break

Figure 7.3 Break at a Program Breakpoint

displayed on the status bar, and the Program 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.

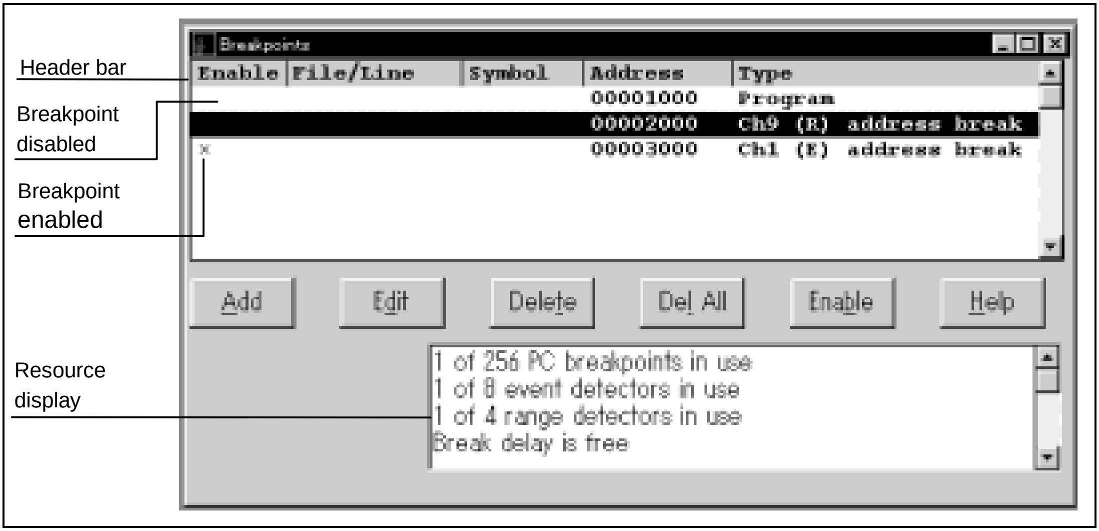
7.2.1 Clearing Program Breakpoints (PC Breakpoints)

To clear a program (PC) breakpoint, double-click on the word ‘Break’ in the Break column of the line at which the program (PC) breakpoint is set. The display will update and the word ‘Break’ disappears.

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 select the **[View->Breakpoint Window]** menu option or click the Breakpoint Window toolbar button [▣], if it visible.

A Breakpoints window opens.



Enable	File/Line	Symbol	Address	Type
			00001000	Program
			00002000	Ch9 (R) address break
×			00003000	Ch1 (E) address break

```
1 of 256 PC breakpoints in use
1 of 8 event detectors in use
1 of 4 range detectors in use
Break delay is free
```

Figure 7.4 Breakpoints Window

The window is divided into two main areas; a list of the breakpoints set in the system, and a display of breakpoint resources. 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 in one of three ways:

- Click the **[Add]** button.
- Invoke the pop-up menu by clicking the right mouse button and select the **[Add]** menu option.
- Invoke the pop-up menu by pressing **SHIFT+F10** and select the **[Add]** menu option.

In this manual, directions to click a button also mean selecting the local pop-up menu option having the same name as the button.

Breakpoint/Event Properties dialog box will appear in which you can select the type and parameters of the breakpoint that you wish to set.

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 click **[Edit]** button.

Breakpoint/Event Properties dialog box will appear in which you can select the type and parameters of the breakpoint that you wish to set.

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 click **[Delete]** button.

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 click **[Del All]** button.

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 click **[Disable]** button.

The menu disappears and the breakpoint list updates to show that the breakpoint is no longer enabled in the Enable column.

7.4.2 Enabling a Breakpoint

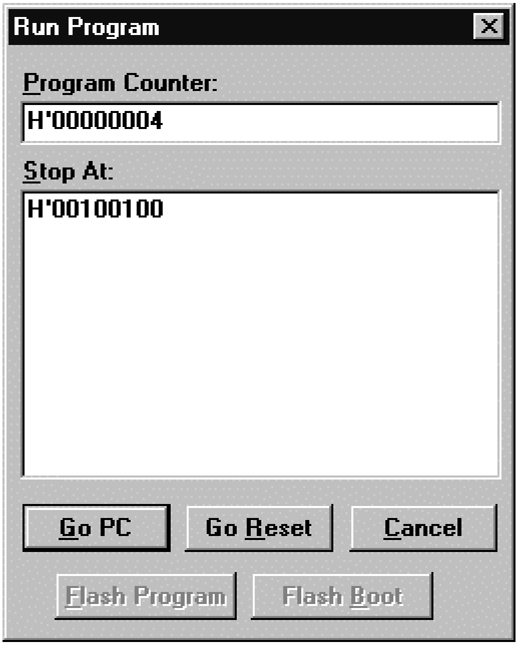
When you want to re-enable a breakpoint in the Breakpoints window list, select the breakpoint in the list by clicking on the line corresponding to it and click **[Enable]** button.

The menu disappears and the breakpoint list updates to show that the breakpoint is again enabled, by showing a check mark in the Enable column.

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 address. For example you may want to perform something like the Go To Cursor operation, but the destination may be outside the Program window or you may want to stop at several addresses. To do this you can use HDI’s temporary breakpoint feature to run with up to ten temporary breakpoints set that are cleared when you break. Temporary breakpoints are set in the Run Program dialog box, which is opened by selecting the **[Run->Run...]** menu option.

The Run Program dialog box appears:



Program Counter: H'00000004

Stop At: H'00100100

Figure 7.5 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 appears 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.

The program will stop if it reaches a temporary breakpoint, a normal breakpoint or if you halt it manually. When the program halts the temporary breakpoints that you specified in the Stop At field are cleared. When you select the Run Program dialog box again, the temporary breakpoints are listed and will be set again if you click **[Go PC]** or **[Go Reset]**. To remove a temporary breakpoint from the list, select it by clicking and dragging the mouse across it and press DELETE. If you click **[Cancel]** any changes that you have made in the Run Program dialog box will be lost.

7.6 Hardware Breakpoints(Event)

Note: The 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*.

Section 8 Looking at Variables

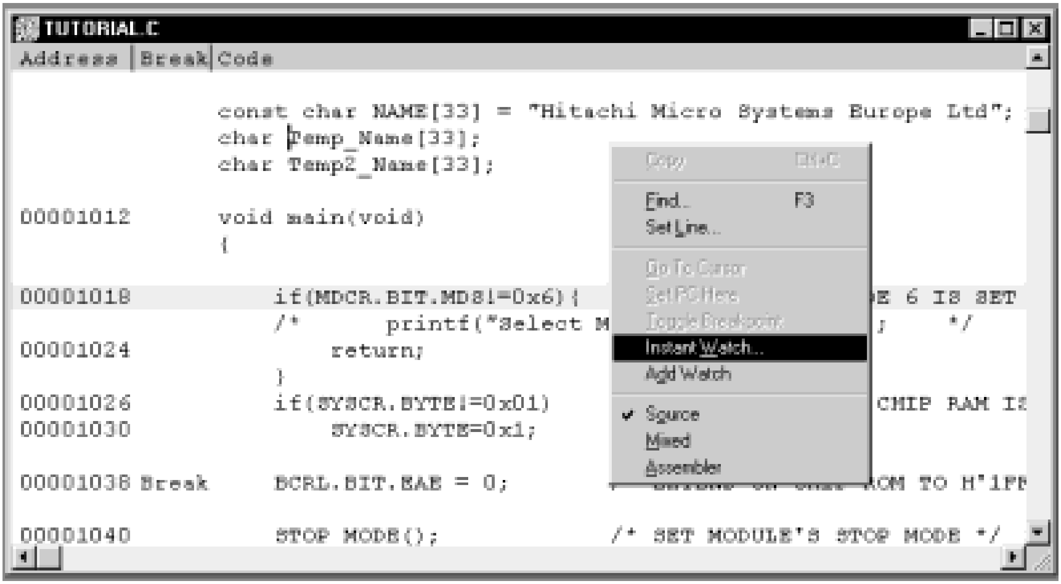
This section describes how to look at the variables and data objects that your program uses. It shows you how to view variables, set up watch items and look at the contents of the CPU's general and on-chip peripheral registers.

8.1 Instant Watch

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

To use Instant Watch:

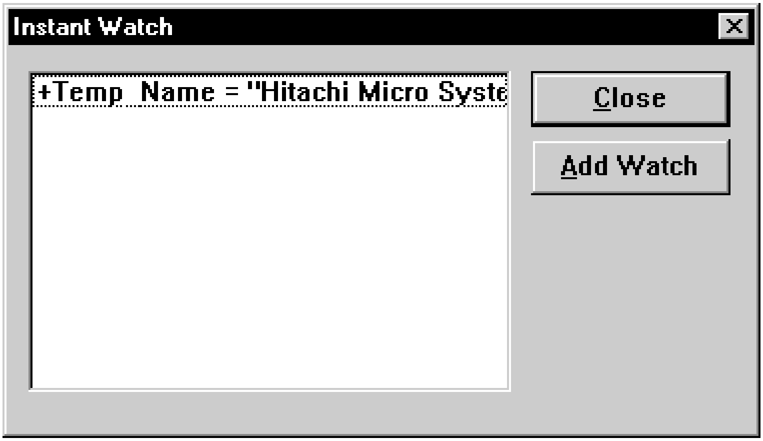
1. Open the Program window showing the variable that you want to examine.
2. Click on the variable. You should see a cursor on the variable.
3. Invoke the pop-up menu by clicking the right mouse button, (or pressing **SHIFT+F10**), and click on **[Instant Watch]**.



```
                 const char NAME[33] = "Hitachi Micro Systems Europe Ltd";
                 char Temp_Name[33];
                 char Temp2_Name[33];

00001012         void main(void)
                 {

00001018             if(MDCR.BIT.MDS!=0x6){                         E 6 IS SET
                     /*      printf("Select M                       ;     */
00001024                 return;
                     }
00001026             if(SYSCR.BYTE!=0x01)                           CHIP RAM IS
00001030                 SYSCR.BYTE=0x1;

00001038 Break       BCRL.BIT.EAE = 0;                              ROM TO H'1FF

00001040             STOP MODE();              /* SET MODULE'S STOP MODE */
```

Figure 8.1 Selecting Instant Watch

An Instant Watch dialog box appears showing the variable name and its value:



+Temp_Name = "Hitachi Micro Syste

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.

8.2 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 select the **[View->Watch Window]** 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.

8.2.1 Adding a Watch

There are two ways to add Watch items to the Watch window; the quick method accessed from the Program 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.

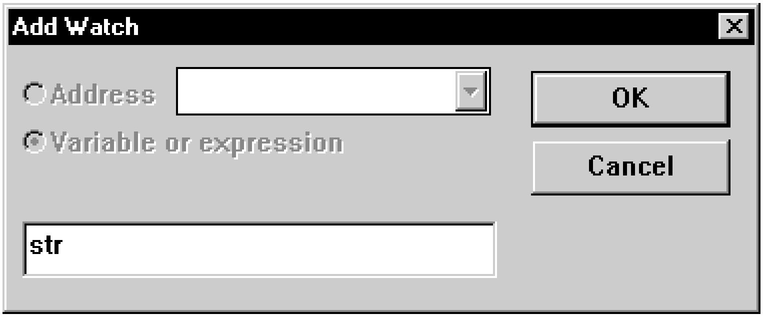
To use Add Watch:

1. Open the Program window showing the variable that you want to examine.
2. Click on the variable. You should see a cursor on the variable.
3. Invoke the Program window pop-up menu by clicking the right mouse button, (or pressing **SHIFT+F10**) and click on **[Add Watch]**.
4. The variable is added as a watch item and the Watch window updates.

Full Method

To add an item to the Watch window, invoke the local pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**) in the Watch window and click on the **[Add Watch]** option.

The Add Watch dialog box appears:



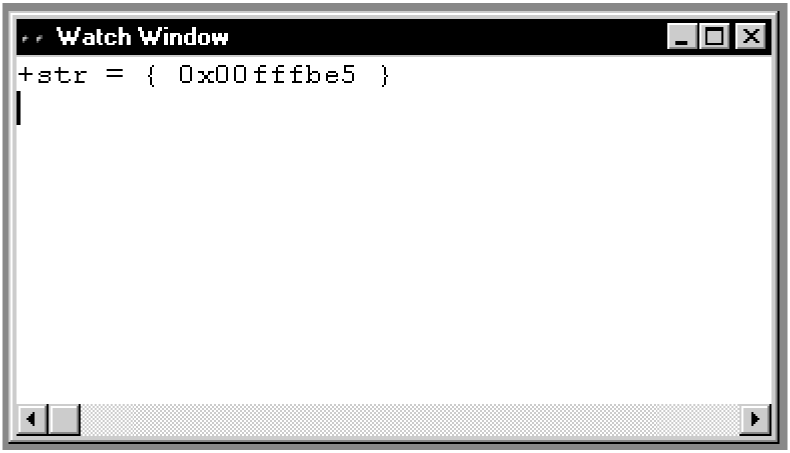
○ Address

● Variable or expression

str

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.



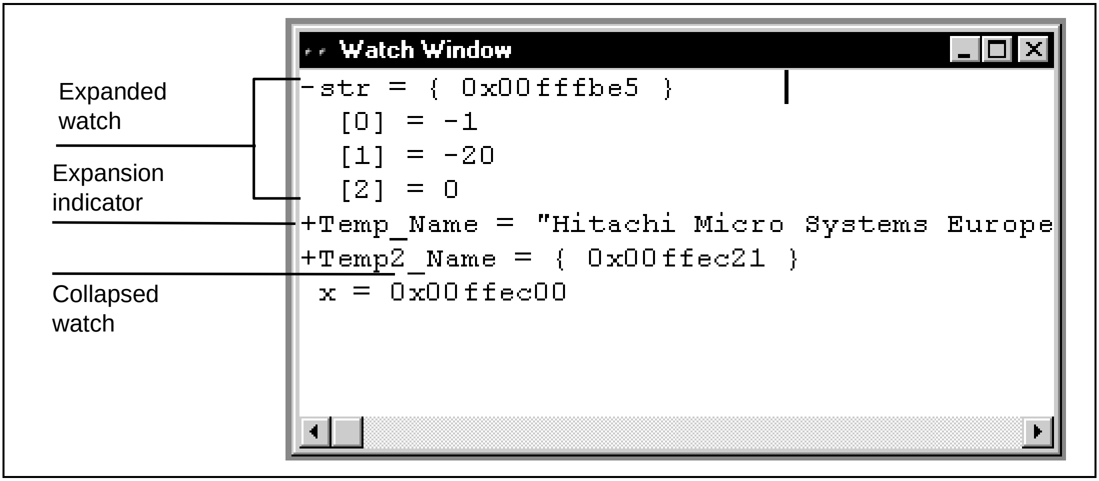
```
+str = { 0x00fffbe5 }
```

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.

8.2.2 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 stop, 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.



```
-str = { 0x00fffbe5 }
  [0] = -1
  [1] = -20
  [2] = 0
+Temp_Name = "Hitachi Micro Systems Europe
+Temp2_Name = { 0x00ffec21 }
 x = 0x00ffec00
```

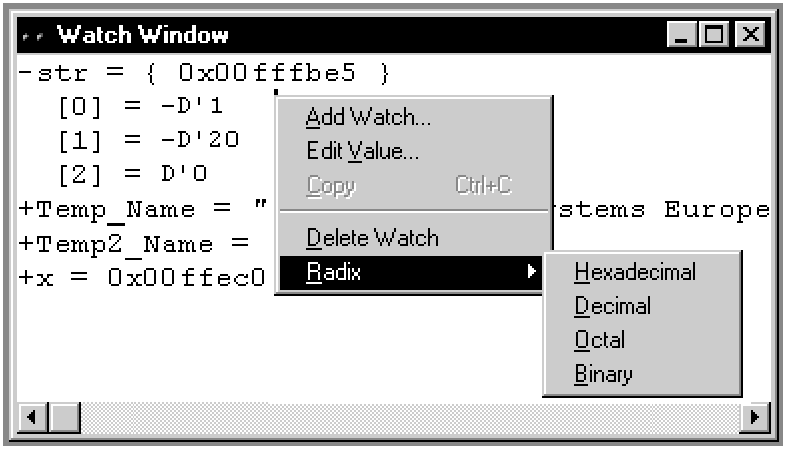
Figure 8.5 Expanding a Watch

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

8.2.3 Modifying Radix for Watch Item Display

To change the radix for watch item display, click on the watch item you wish to change. Then invoke the pop-up menu by clicking the right mouse button, (or pressing **SHIFT+F10**) and click on the **[Radix]** option.

The submenu to specify the radix appears:



```
-str = { 0x00fffbe5 }
   [0] = -D'1
   [1] = -D'20
   [2] = D'0
+Temp_Name = "               stems Europe
+Temp2_Name =
+x = 0x00ffec0
```

Figure 8.6   Modifying Radix for Watch Item Display

Click on the radix you wish to be used for display, and the radix of the selected watch item is changed to that.

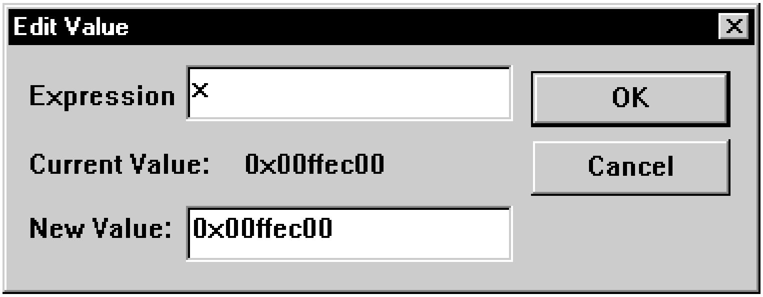
8.2.4 Changing a Watch Item’s Value

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. Invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**).
3. Click on the **[Edit Value]** menu option.

The Edit Value dialog box appears:



Expression x

Current Value: 0x00ffec00

New Value: 0x00ffec00

Figure 8.7   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.

8.2.5 Deleting a Watch

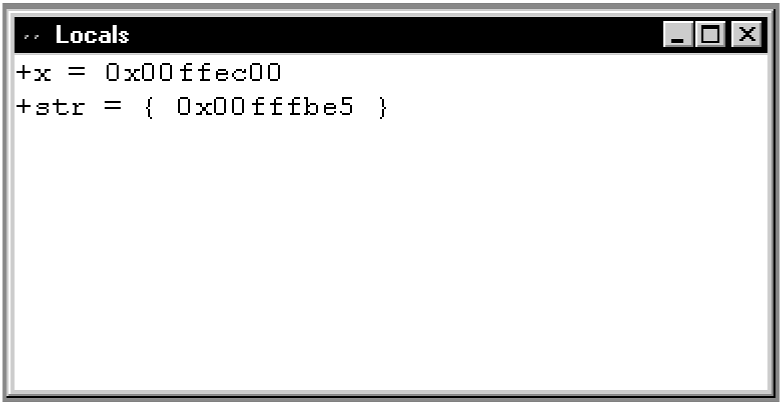
To delete a watch item, select it by clicking on it. Invoke the pop-up menu by clicking the right mouse button (or pressing **SHIFT+F10**) and click on the **[Delete Watch]** option. The item is deleted and the Watch window is updated.

Note: The watches that you have set in the watch window can be saved in a session file. See *section 11, Configuring the User Interface*.

8.3 Looking at Local Variables

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

The Locals window appears:



```
+x = 0x00ffec00
+str = { 0x00fffbe5 }
```

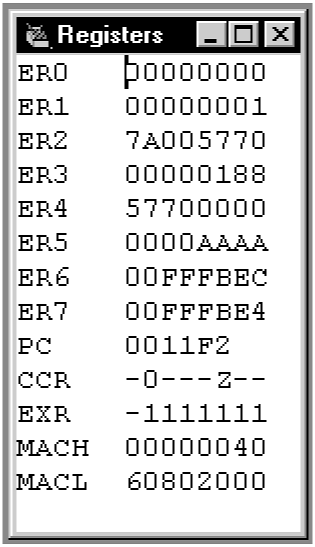
Figure 8.8 Locals Window

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.

8.4 Looking at Registers

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



```
ER0    00000000
ER1    00000001
ER2    7A005770
ER3    00000188
ER4    57700000
ER5    0000AAAA
ER6    00FFFBEC
ER7    00FFFBE4
PC     0011F2
CCR    -0---Z--
EXR    -1111111
MACH   00000040
MACL   60802000
```

Figure 8.9 Registers Window

To open a Registers window select the **[View->Register Window]** menu option or click the Register Window toolbar button [] if it is visible. A Registers window opens showing all of the CPU’s general registers and their values, displayed in hexadecimal.

8.4.1 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, and the full edit method that requires you to enter values via a dialog box, but allows you to enter values in any base and complex expressions.

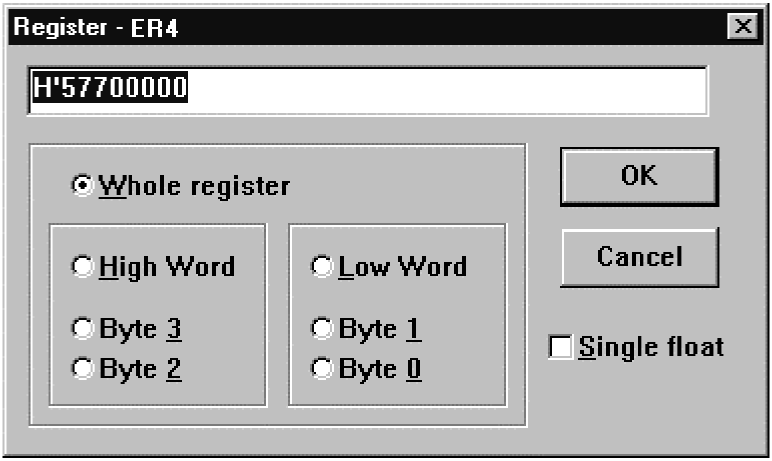
Quick Edit

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 four ways:

- Double-click the register you want to change.
- Select the register you want to change, and press **ENTER**.
- Select the register you want to change, invoke the pop-up menu by clicking the right mouse button, and click on the **[Edit]** menu option.
- Select the register you want to change, invoke the pop-up menu by pressing **SHIFT+F10**, and click on the **[Edit]** menu option.


Register - ER4

H'57700000

● Whole register

○ High Word
○ Byte 3
○ Byte 2

○ Low Word
○ Byte 1
○ Byte 0

□ Single float

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 or just part of the register by selecting the Whole register/High Word/Low Word/Byte 0-3 radio buttons. You can enter floating-point data by checking the Single float check box.

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.

8.4.2 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 Program or Memory windows. You can do this by specifying the register name prefixed by the “#” character, e.g.: #R1, #PC, #R6L, or #ER3.

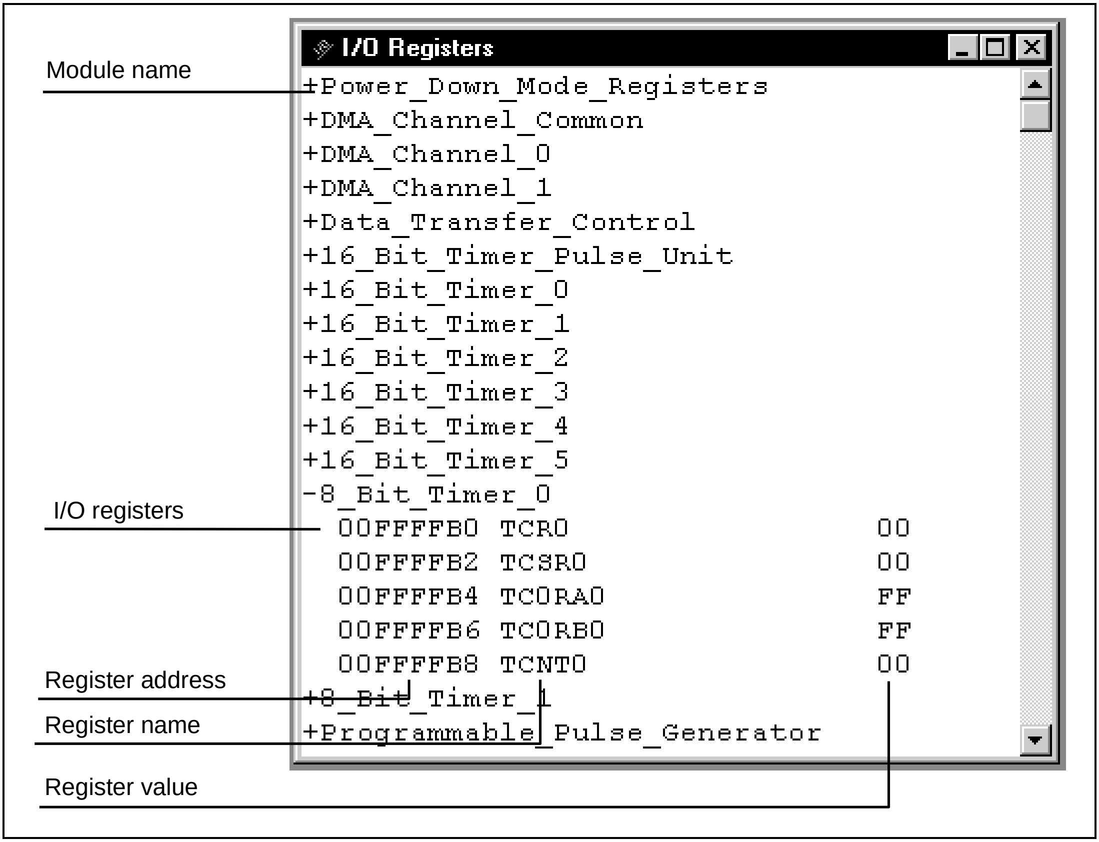
8.5 Looking at I/O

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 into the microcomputer’s address space.

Since the setting up and use of these on-chip peripheral registers is usually very important in an embedded microcomputer 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.

8.5.1 Opening an I/O Registers Window

To open an I/O Registers window select the **[View->I/O Register Window]** menu option or click the I/O Register Window toolbar button [▣] if it is visible. The I/O register information is organized by modules, corresponding to the on-chip peripheral modules. When an I/O Registers window is first opened, only a list of module names is displayed.



```
+Power_Down_Mode_Registers
+DMA_Channel_Common
+DMA_Channel_0
+DMA_Channel_1
+Data_Transfer_Control
+16_Bit_Timer_Pulse_Unit
+16_Bit_Timer_0
+16_Bit_Timer_1
+16_Bit_Timer_2
+16_Bit_Timer_3
+16_Bit_Timer_4
+16_Bit_Timer_5
-8_Bit_Timer_0
  00FFFFB0  TCR0                  00
  00FFFFB2  TCSR0                 00
  00FFFFB4  TCORA0                FF
  00FFFFB6  TCORB0                FF
  00FFFFB8  TCNT0                 00
+8_Bit_Timer_1
+Programmable_Pulse_Generator
```

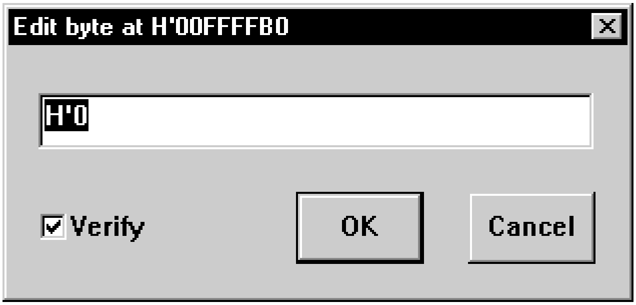
Figure 8.11   I/O Registers Window

8.5.2 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.

8.5.3 Modifying I/O Register Contents

To edit the value in an I/O register you can double click or press **ENTER** on the register to open a dialog box to modify the register contents:


Edit byte at H'00FFFFB0

H'0

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 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 Code window) open on the I/O area can have the same effect.

8.5.4 I/O Register Files

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.

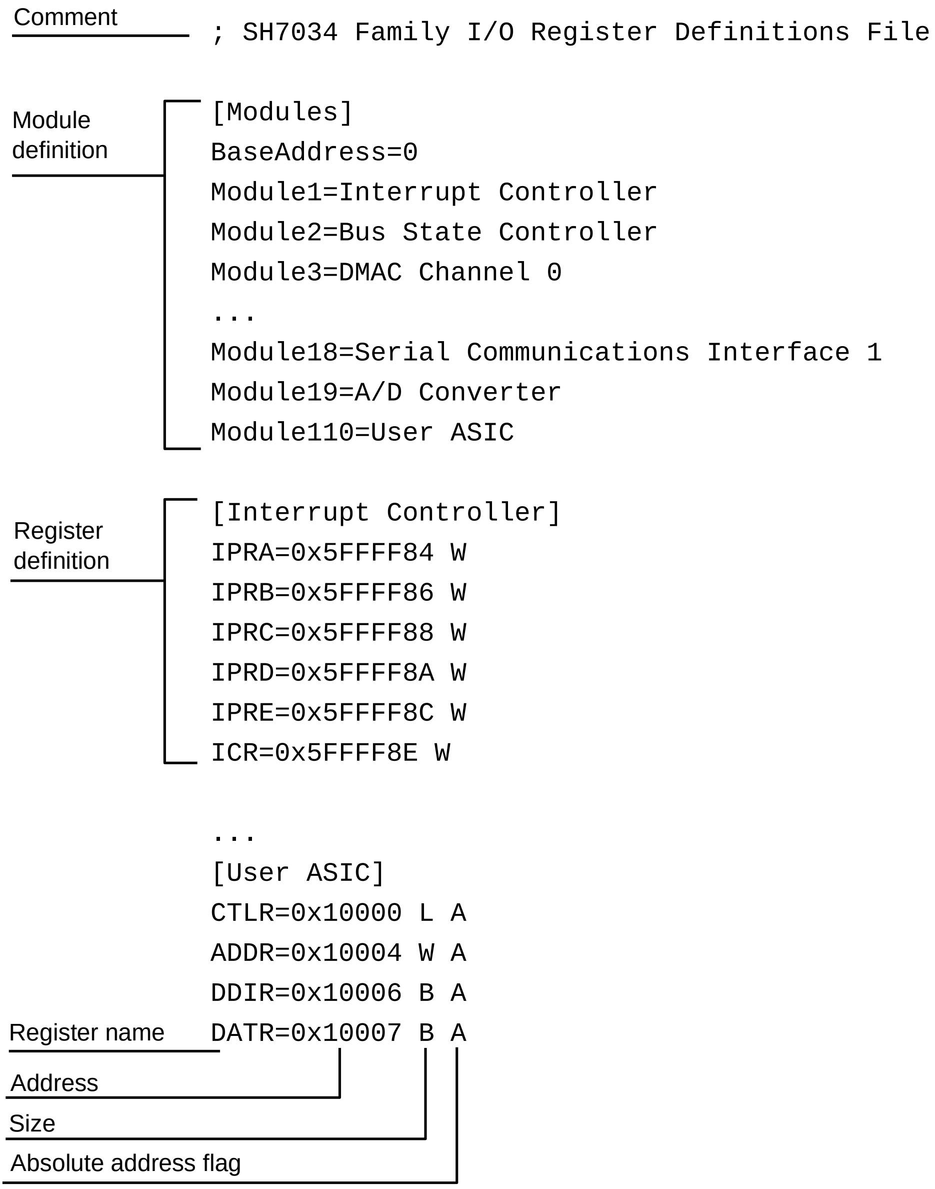
The ‘BaseAddress’ definition is for devices where the location of I/O registers moves in the address space depending on the CPU mode. In this case, the ‘BaseAddress’ value is the base address of the I/O registers in one specific mode and the addresses used in the register definitions are the address 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.

The first field in a [Register] definition entry is the register name followed by the “=” character. The second field is the register address value followed by two optional flags:

1. <size> which may be B, W or L for byte, word, or long word (default is byte).
2. <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.

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

Example:



```
; SH7034 Family I/O Register Definitions File

[Modules]
BaseAddress=0
Module1=Interrupt Controller
Module2=Bus State Controller
Module3=DMAC Channel 0
...
Module18=Serial Communications Interface 1
Module19=A/D Converter
Module110=User ASIC

[Interrupt Controller]
IPRA=0x5FFFF84 W
IPRB=0x5FFFF86 W
IPRC=0x5FFFF88 W
IPRD=0x5FFFF8A W
IPRE=0x5FFFF8C W
ICR=0x5FFFF8E W

...
[User ASIC]
CTLR=0x10000 L A
ADDR=0x10004 W A
DDIR=0x10006 B A
DATR=0x10007 B A
```

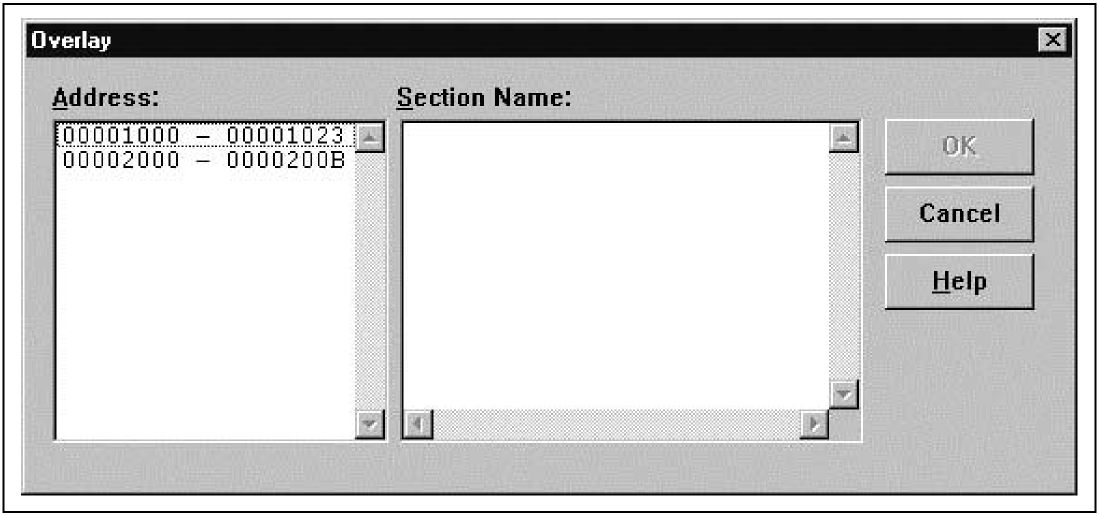
Section 9 Overlay Function

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 selecting the **[Setup->Overlay]** menu option.

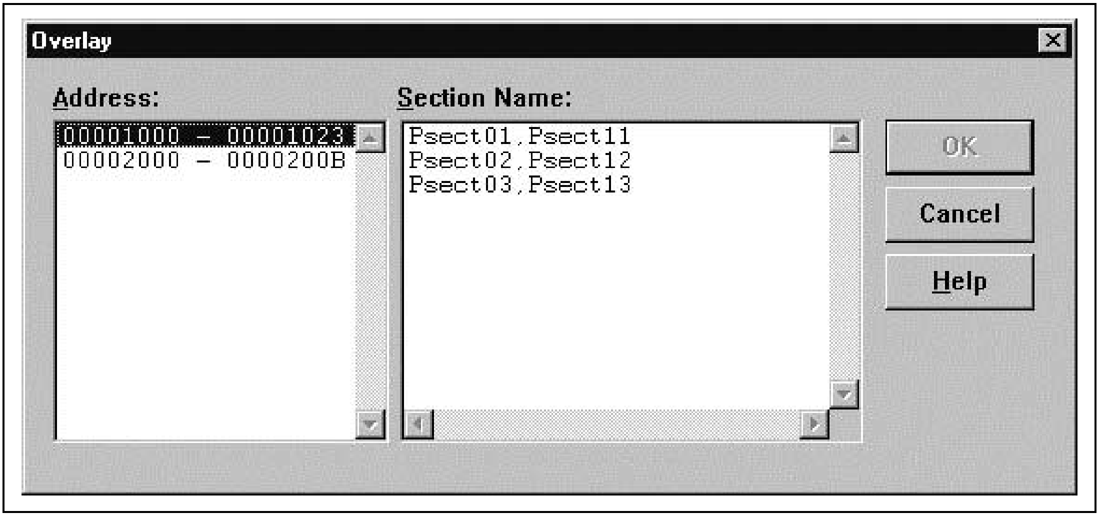


00001000 – 00001023
00002000 – 0000200B

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.



00001000 - 00001023
00002000 - 0000200B

Psect01,Psect11
Psect02,Psect12
Psect03,Psect13

Figure 9.2 Overlay Dialog Box (Address Range Selected)

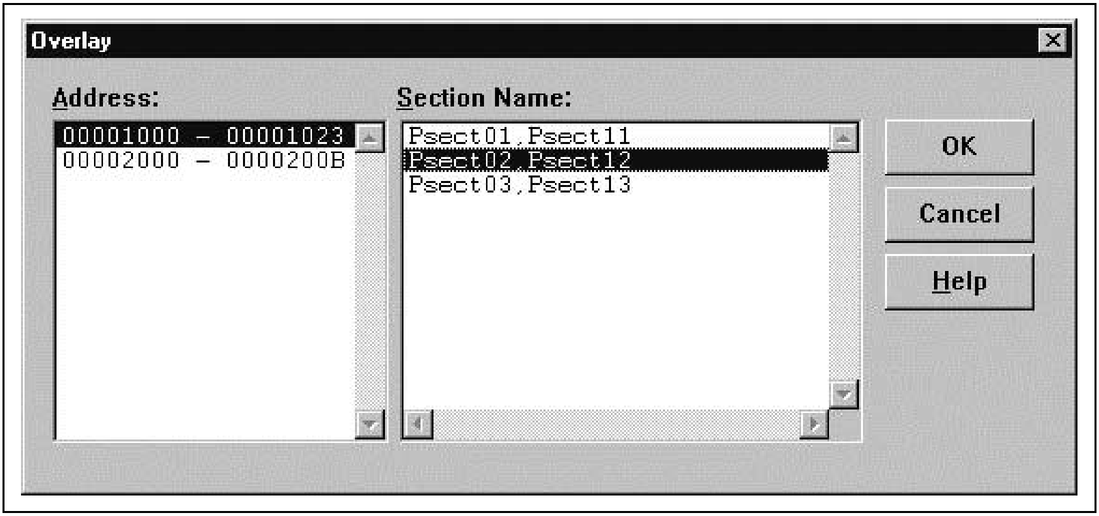
The Section Name list box displays the section groups assigned to the selected address range.

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.



00001000 – 00001023
00002000 – 0000200B

Psect01,Psect11
Psect02,Psect12
Psect03,Psect13

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.

Section 10 Selecting Functions

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

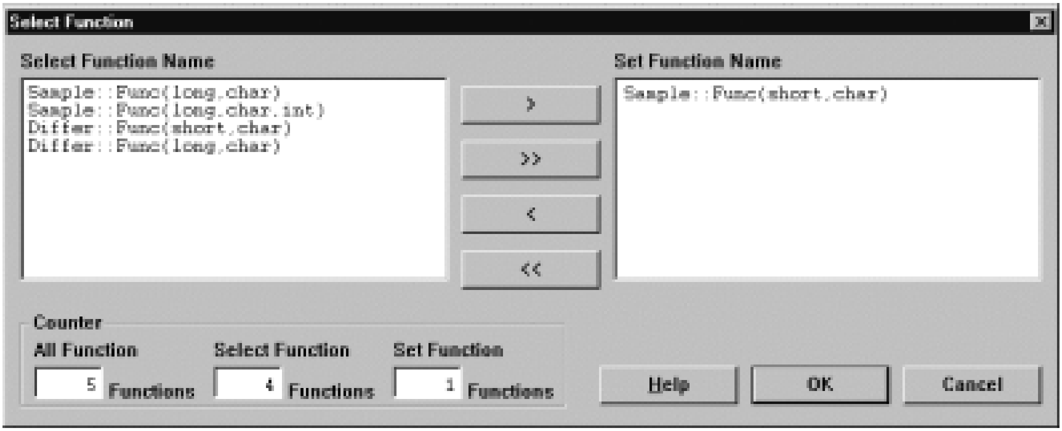
10.1 Displaying Functions

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



Select Function

Select Function Name

```
Sample::Func(long,char)
Sample::Func(long,char,int)
Differ::Func(short,char)
Differ::Func(long,char)
```

Set Function Name

```
Sample::Func(short,char)
```

Counter

All Function: 5 Functions

Select Function: 4 Functions

Set Function: 1 Functions

Figure 10.1 Select Function Dialog Box

This dialog box has three areas.

- Select Function Name list box

  Displays the same-name 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 same-name 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, multiple functions can be selected.

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.

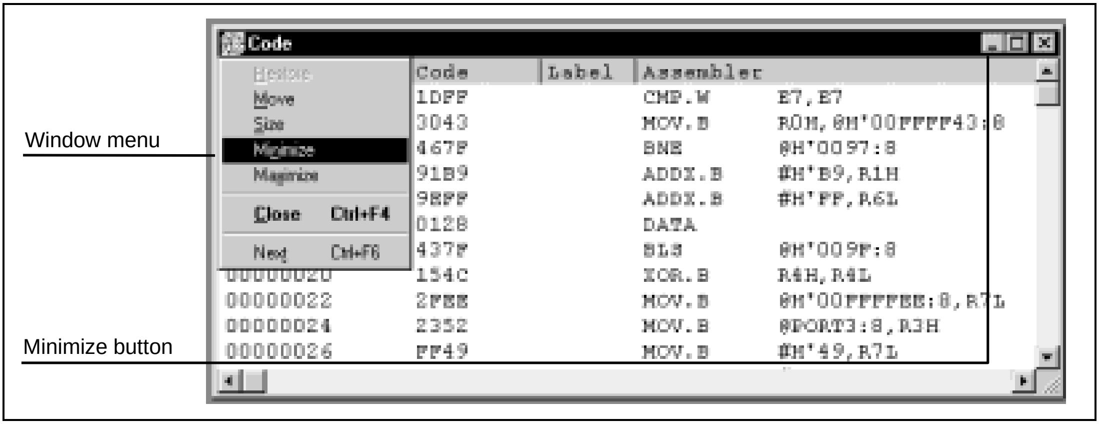
Section 11 Configuring the User Interface

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 select the **[->Minimize]** window menu option.



```
                Code       Label     Assembler
                1DFF                 CMP.W      E7,E7
                3043                 MOV.B      R0H,@H'00FFFF43:8
                467F                 BNE        @H'0097:8
                91B9                 ADDX.B     #H'B9,R1H
                9EFF                 ADDX.B     #H'FF,R6L
                0128                 DATA
                437F                 BLS        @H'009F:8
00000020        154C                 XOR.B      R4H,R4L
00000022        2FEE                 MOV.B      @H'00FFFFEE:8,R7L
00000024        2352                 MOV.B      @PORT3:8,R3H
00000026        FF49                 MOV.B      #H'49,R7L
```

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 Code window example the icon is:

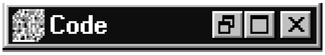


Figure 11.2 Code Window Icon

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 click once to invoke the pop-up menu and select **[Restore]**.

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, select the **[Window->Arrange Icons]** menu option.

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

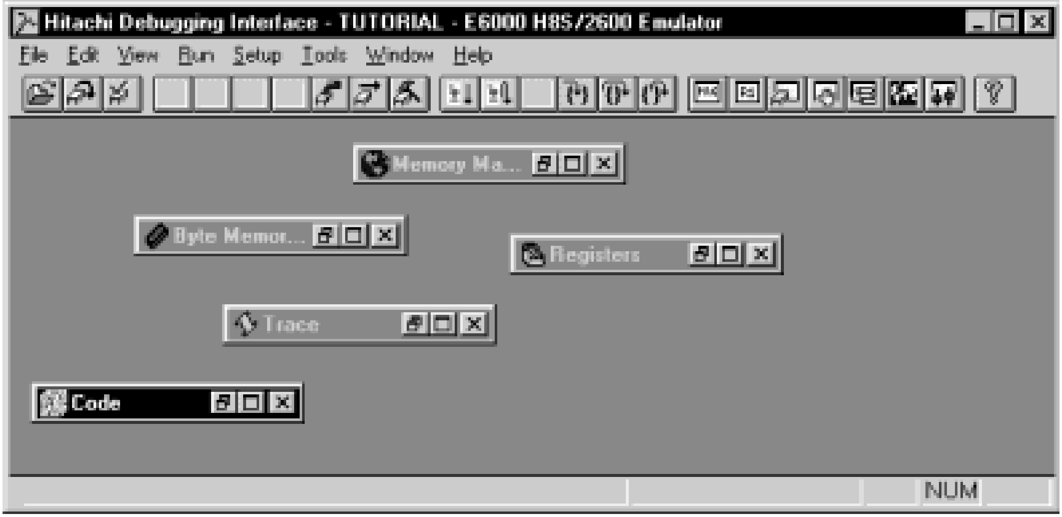

Hitachi Debugging Interface - TUTORIAL - E6000 H8S/2600 Emulator
File Edit View Run Setup Tools Window Help
Memory Ma...
Byte Memor...
Registers
Trace
Code
NUM

Figure 11.3 Icons Before Arrangement

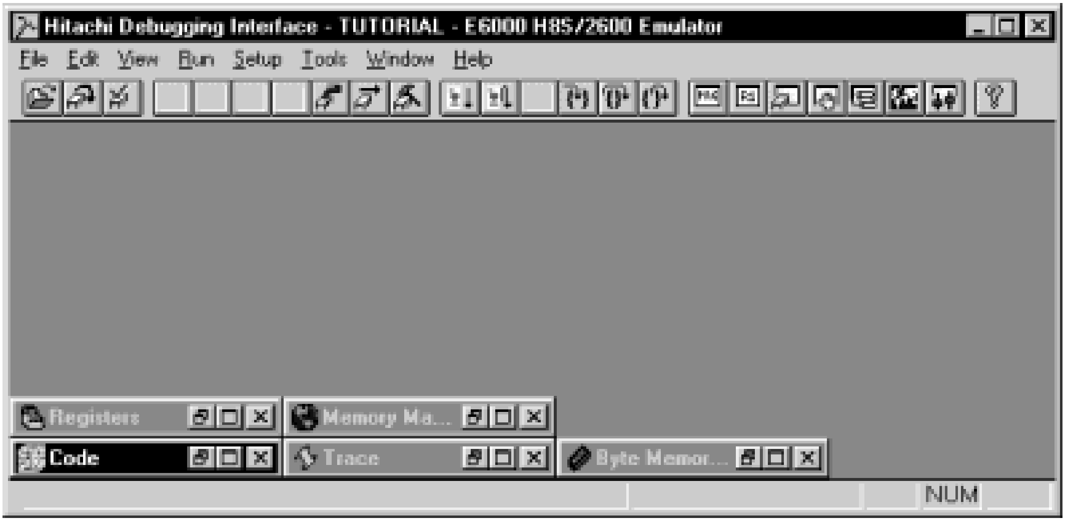

Hitachi Debugging Interface - TUTORIAL - E6000 H8S/2600 Emulator
File Edit View Run Setup Tools Window Help
Registers
Memory Ma...
Code
Trace
Byte Memor...
NUM

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. To invoke this select 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. To invoke this select 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, select **[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, invoke the **[Window]** menu. Click on the window that you wish to select from the list of windows (open and minimized) at the bottom of the menu. The currently selected window has a check mark next to it in the window list. In the following example, the Code window is the currently selected window:

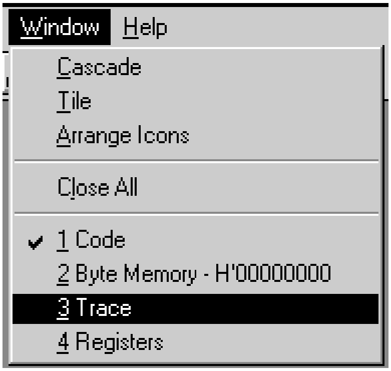


Figure 11.5 Selecting a Window

The window that you select (the Trace window in the above example) 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, select the **[View->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, select the **[View->Status Bar]** menu option again. The Status bar will be enabled and added to the HDI application window display.

11.4 Enabling/Disabling the Toolbar

You can select whether or not the Toolbar is displayed at the top of the HDI application window; by default it will be displayed. To disable display of the Toolbar, select the **[View->Toolbar]** menu option.

The Toolbar will be disabled and removed from the HDI application window display. To re-enable the Toolbar display, select the **[View->Toolbar]** menu option again. The Toolbar will be enabled and added to the HDI application window display.

11.5 Customizing the Toolbar

You can customize the selection and arrangement of buttons displayed on the toolbar. To change the display invoke the **[Setup->Customise->Toolbar]** menu option.

The Customise Toolbar dialog box appears:

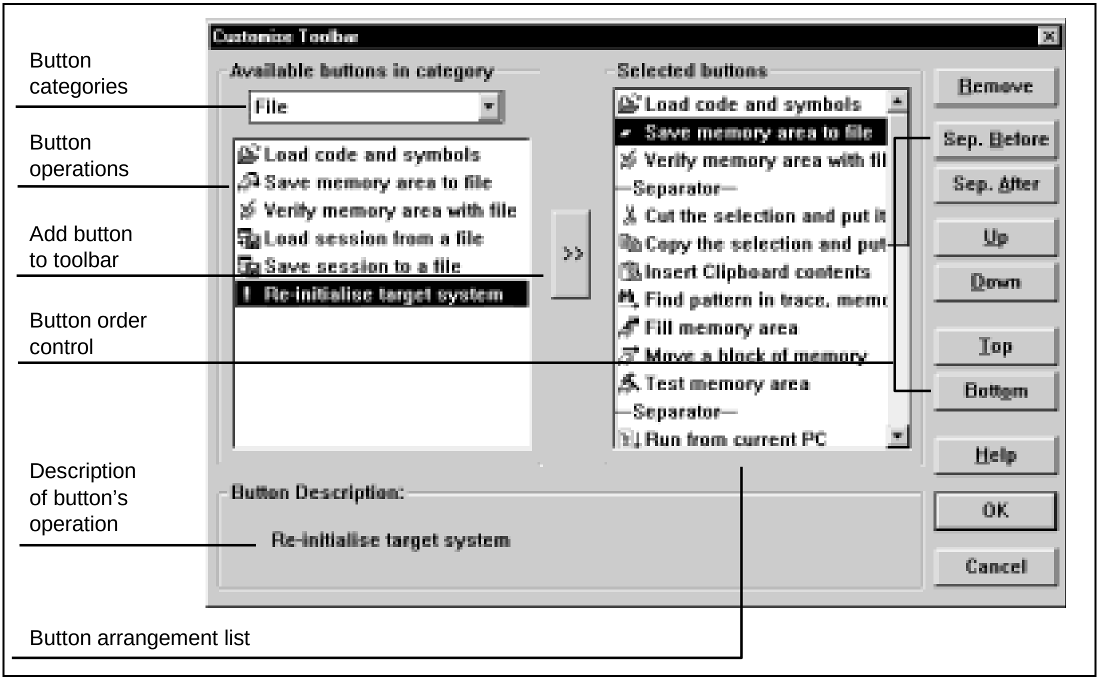


Figure 11.6 Customise Toolbar Dialog Box

11.5.1 Button Categories

At the top left of the dialog box is a list of button categories, which are; File, Edit, View, Run, Setup, Tools, Window, and Help. For each category a list of possible button controlled operations is listed below the category. 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.

11.5.2 Adding a Button to the Toolbar

To add a button to the toolbar:

1. Select the button category from the button category list.
2. Select the button item from the operation list.
3. Click the add **[>>]** button.

The button is added to the list. If an existing button item is selected in the Selected Buttons list, the new button is added after the selected button in the list. If no button is selected the new button is added to the bottom of the list.

11.5.3 Positioning a Button in the Toolbar

To move a button position in the toolbar order:

1. Select the button to move in the Selected Button list.
2. Click the button order controls to move the button; **[Up]**, **[Down]**, **[Top]**, **[Bottom]** of the list.

You can add a separator in the list to separate blocks of buttons by clicking on the **[Sep. Before]** or **[Sep. After]** buttons.

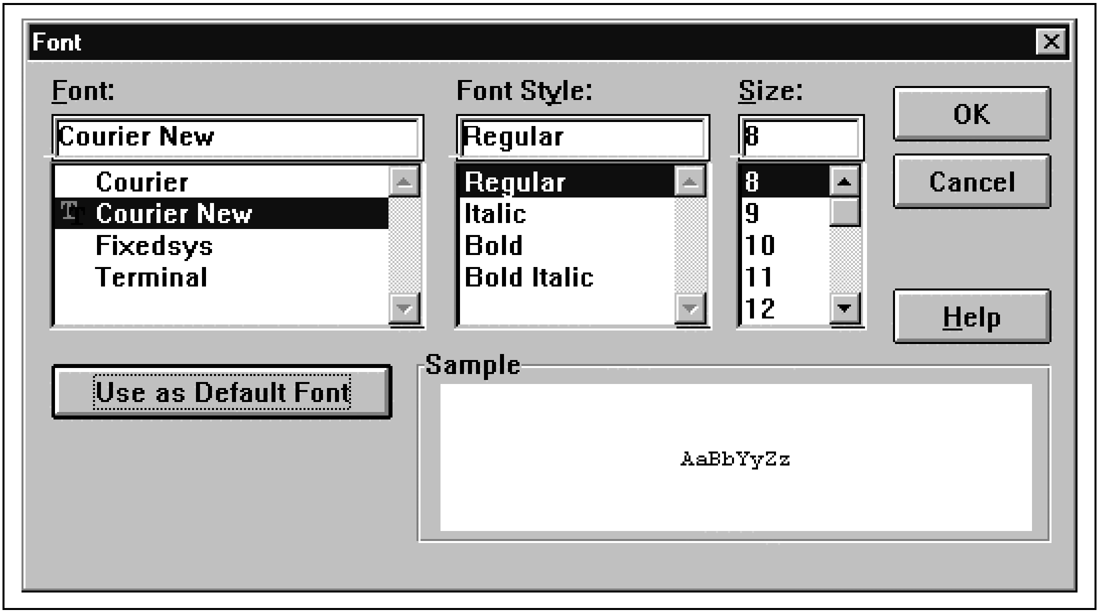
11.5.4 Removing a Button from the Toolbar

To remove a button from the toolbar, select the button in the Selected Buttons list and click **[Remove]**. The button is removed from the list and the list is updated.

11.6 Customizing the Fonts

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

To change the display invoke the **[Setup->Customise->Font]** menu option. This will launch the Font configuration dialog box:



Font

Font: Courier New

Font Style: Regular

Size: 8

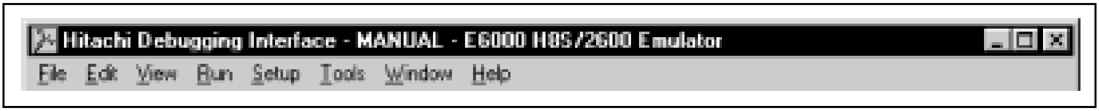
Figure 11.7 Font Dialog Box

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. The command button 'Use as Default Font' will save the current font settings as the font used when opening any new windows that do not already have their own font settings.

11.7 Saving a Session

If you have downloaded user code 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.

To save the current setting, select the **[File->Save Session As...]** menu option. This will launch a standard Windows® file dialog box prompting you for a file name. Two files are saved, an HDI session file (*.hds) and a target session file (*.hdt) - the former includes the HDI interface settings, e.g. all the open windows and their positions, while the latter includes the settings specific to the debugging platform/user system, e.g. the name of the debugging platform and its configuration. The session name is then displayed as the second entry in HDI's title bar, e.g. "MANUAL":



Hitachi Debugging Interface - MANUAL - E6000 H8S/2600 Emulator

Figure 11.8 Session Name Display

If you are already using a named session, it can be updated by selecting the **[File->Save Session]** menu option. This will overwrite the current session files with the latest settings.

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 4.2.10, Saving a Symbol File* and *section 5.7, Saving an Area of Memory*.

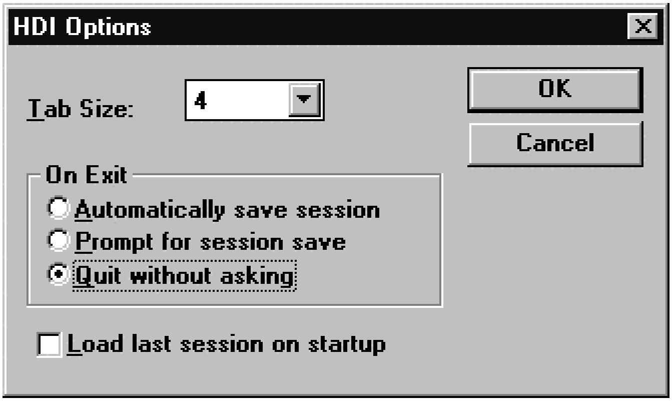
11.8 Loading a Session

To reload a saved session, select the **[File->Load Session]** menu option. This will launch a standard Windows® file dialog box prompting you for an HDI session file name (*.hds) - the associated target session file (*.hdt) will be automatically loaded at the same time.

Any currently open windows will be closed, and the connection to the debugging platform initialized. If user code 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. Selecting the **[Setup->Options]** menu option will launch the HDI Options dialog box:



Tab Size: 4

On Exit

○ Automatically save session
○ Prompt for session save
● Quit without asking

□ Load last session on startup

Figure 11.9 HDI Options 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.

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 session save - 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.

11.10 Setting the Default Radix

HDI can display numbers in several formats. Normally you would fill in information fields by using 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, use 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:

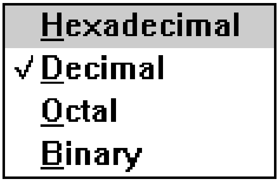


Figure 11.10 Setting Radix

Section 12 Menus

This document uses the standard Microsoft® menu naming convention:

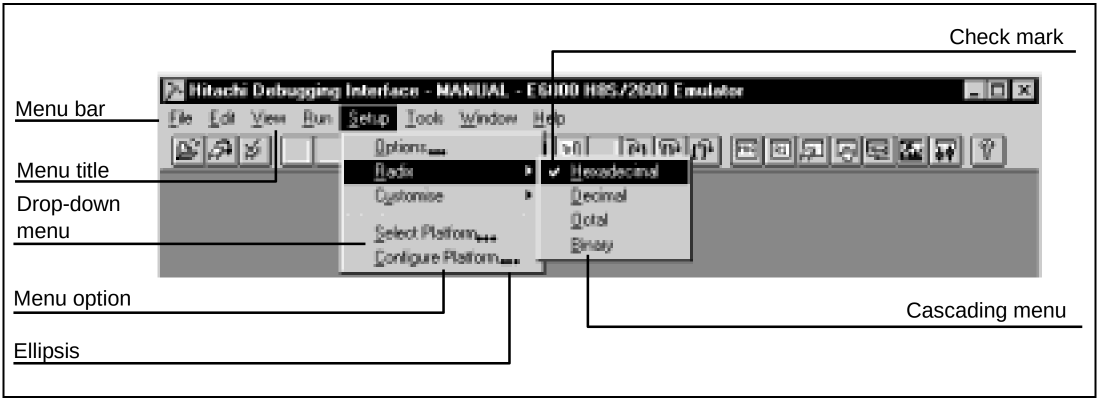


Figure 12.1 Menus

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

Ellipsis indicate 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 data files.

12.1.1 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.

12.1.2 Save Memory...

Launches the Save S-Record File 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.

12.1.3 Verify Memory...

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.

12.1.4 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.

12.1.5 Load Session...

Launches the Open dialog box allowing the user to load a session from a selected session file (*.hds extension). A session file contains the details of the current debugging platform, 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.

12.1.6 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 details of the current debugging platform, 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.

12.1.7 Initialise

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 12.4.11, Reset CPU*)

12.1.8 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 12.5.1, 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 who’s contents can be modified (shown disabled).

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 (shown disabled).

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 (shown disabled).

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 (shown disabled).

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 Set Line

Only available in the Program window source format. Launches the Set Line dialog box allowing the user to enter an integer (i.e. a line number of the source program) - the view will then scroll to display this line at the top of the window.

12.2.6 Fill Memory...

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

12.2.7 Move Memory...

Launches the Move 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.

12.2.8 Test Memory...

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 OR DATA WILL BE ERASED.

12.2.9 Update Memory

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

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 Toolbar

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

12.3.2 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.

12.3.3 Breakpoint Window

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

12.3.4 Command Line Window

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.

12.3.5 I/O Register Window

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

12.3.6 Local Variable Window

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 Mapping Window

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.

12.3.8 Memory Window...

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

12.3.9 Performance Analysis Window

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

12.3.10 Program Window...

Launches the Open dialog box allowing the user to enter a file name of the program (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. To view a file in these circumstances use the **[View->Text Window]** menu option.

12.3.11 Register Window

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

12.3.12 Status Window

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

12.3.13 Text Window...

Launches the Open dialog box allowing the user to enter the name of a text file that you wish to view.

12.3.14 Trace Window

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

12.3.15 Watch Window

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 code in the debugging platform.

12.4.1 Go

Starts executing the user code at the current PC.

12.4.2 Go Reset

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

12.4.3 Go To Cursor

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

12.4.4 Run...

Launches the Run Program dialog box allowing the user to enter temp-orary breakpoints before executing the user code.

12.4.5 Step In

Executes a block of user code 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 12.4.8*, *Step...*). If a subroutine call is reached, then the subroutine will be entered and the view is updated to include its code.

12.4.6 Step Over

Executes a block of user code 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 12.4.8*, *Step...*). If a subroutine call is reached, then the subroutine will not be entered and sufficient user code will be executed to set the current PC position to the next line in the current view.

12.4.7 Step Out

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

12.4.8 Step...

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

12.4.9 Halt Program

Stops the execution of the user code and returns control to the user.

12.4.10 Set PC to Cursor

Sets the PC to the address indicated by the current text cursor (not mouse cursor).

12.4.11 Reset CPU

Resets the user system hardware and sets the PC to the *Reset Vector* address. (See also *section 12.1.7, Initialise*)

12.5 Setup

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

12.5.1 Options...

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

12.5.2 Radix

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.

12.5.3 Customise

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

Toolbar launches the Customise Toolbar dialog box.

Font launches the Font dialog box, allowing a fixed width font to be selected.

12.5.4 Select Platform...

Launches the Select Platform dialog box allowing the user to select a new debugging platform. The item is grayed if only one target DLL is present in the same directory as the HDI program file (and that debugging platform will be selected automatically when HDI is loaded).

12.5.5 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.

12.5.6 Overlay...

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

12.6 Tools

The Tools menu selects and launches additional applications. The following three menu options are always displayed, and the user may define their own tools (e.g. a find in file utility) which will be displayed below the standard items.

12.6.1 Symbols...

Launches the Symbols dialog box allowing the user to manipulate the current program's symbols (labels).

12.6.2 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 radix.

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.

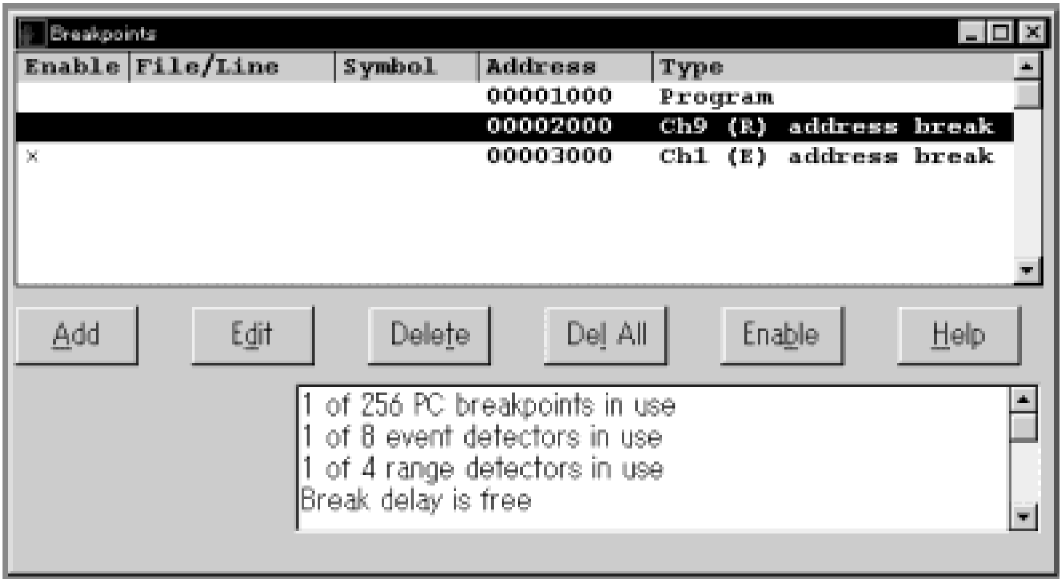
12.8.4 About HDI

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

Section 13 Windows

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

13.1 Breakpoints



Enable	File/Line	Symbol	Address	Type
			00001000	Program
			00002000	Ch9 (R) address break
×			00003000	Ch1 (E) address break

1 of 256 PC breakpoints in use
1 of 8 event detectors in use
1 of 4 range detectors in use
Break delay is free

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*.

The functionality of the command buttons is identical to the pop-up menu options shown below; while global enabling of breakpoints is only available using the check box in the window.

A pop-up menu containing the following options is available by right-clicking within the window.

13.1.1 Help

Launches the help file at the index for controlling breakpoints.

13.1.2 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.

13.1.3 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.

13.1.4 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 13.1.6, Disable/Enable*).

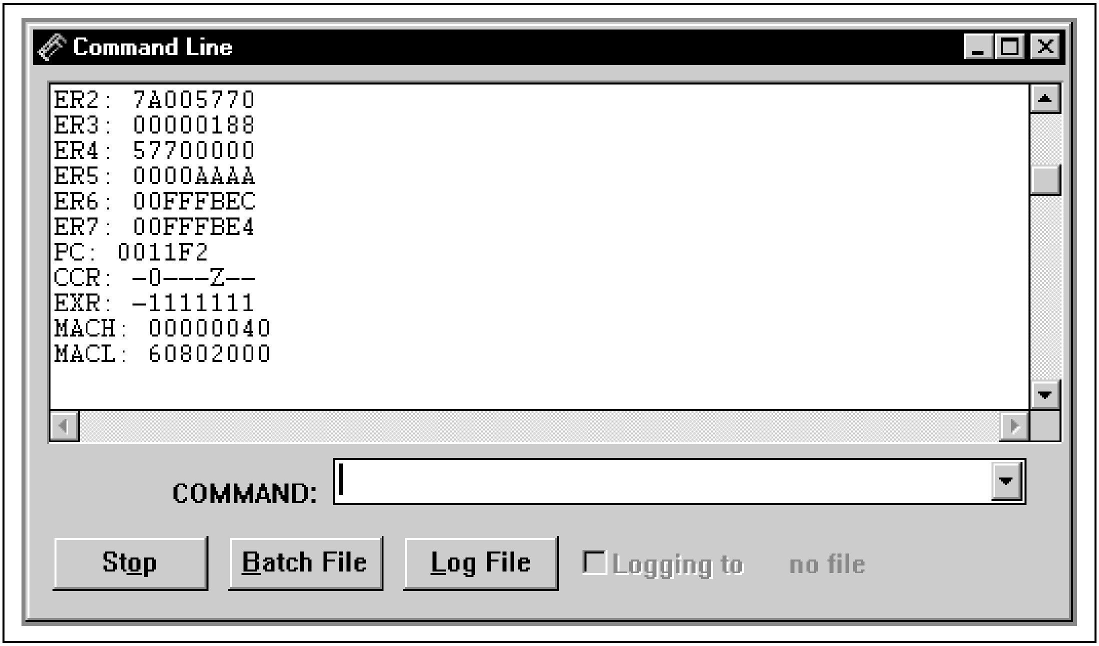
13.1.5 Delete All

Removes all breakpoints from the list.

13.1.6 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 an ‘x’ is shown in the Enable column for the breakpoint).

13.2 Command Line



```
ER2: 7A005770
ER3: 00000188
ER4: 57700000
ER5: 0000AAAA
ER6: 00FFFBEC
ER7: 00FFFBE4
PC: 0011F2
CCR: -0---Z--
EXR: -1111111
MACH: 00000040
MACL: 60802000
```

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. For information about the available commands, refer to the on-line help.

The functionality of the command buttons is identical to the pop-up menu options shown below.

13.2.1 Stop

Stops an executing batch file and returns control to the user.

13.2.2 Batch File

Launches the Run Batch File dialog box, allowing the user to enter the name of an HDI command file (*.hdc).

13.2.3 Log File

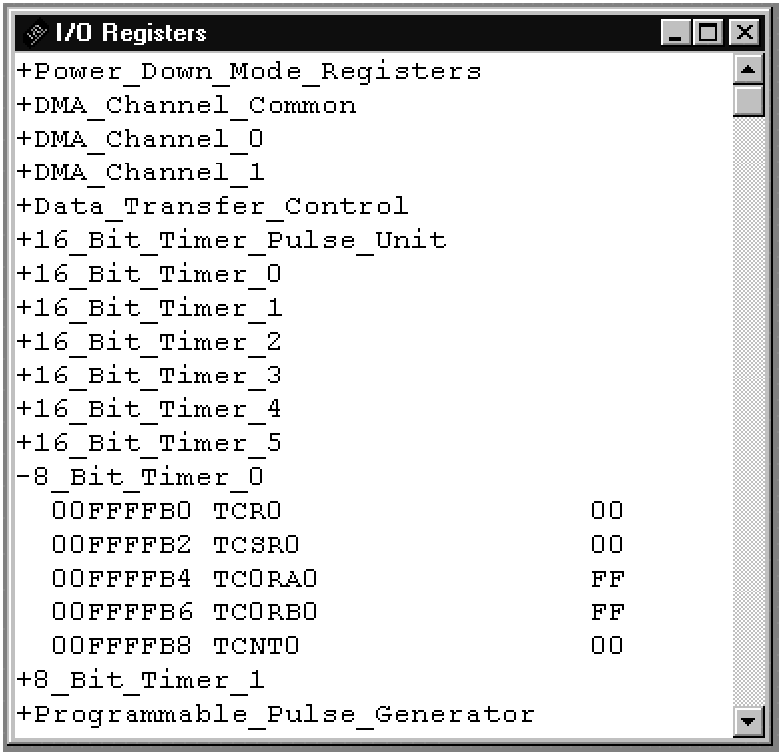
Launches the Open Log File dialog box, allowing the user to enter the name of an HDI log file (*.log). The logging option is automatically set and the name of the file shown on the window, e.g. “ Logging to MANUAL.LOG”.

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, a check mark is shown to the left of the menu text (and the check box in the window is set). 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 I/O Registers



```
+Power_Down_Mode_Registers
+DMA_Channel_Common
+DMA_Channel_0
+DMA_Channel_1
+Data_Transfer_Control
+16_Bit_Timer_Pulse_Unit
+16_Bit_Timer_0
+16_Bit_Timer_1
+16_Bit_Timer_2
+16_Bit_Timer_3
+16_Bit_Timer_4
+16_Bit_Timer_5
-8_Bit_Timer_0
  00FFFFB0 TCR0                 00
  00FFFFB2 TCSR0                00
  00FFFFB4 TCORA0               FF
  00FFFFB6 TCORB0               FF
  00FFFFB8 TCNT0                00
+8_Bit_Timer_1
+Programmable_Pulse_Generator
```

Figure 13.3 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 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.

A pop-up menu containing the following options is available by right-clicking within the window:

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 Edit...

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

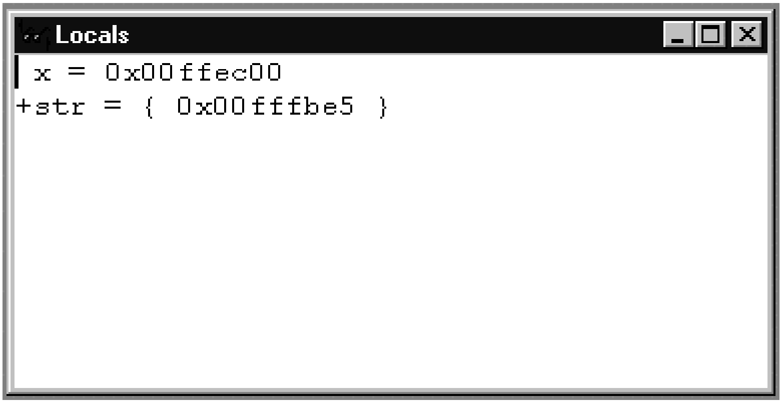
13.3.3 Expand/Collapse

Expands/collapses the selected module.

13.3.4 Help

Launches the help file.

13.4 Locals



```
 x = 0x00ffec00
+str = { 0x00fffbe5 }
```

Figure 13.4 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 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. For more information on the display of information, refer to section 8.2.2, Expanding a Watch.

A pop-up 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.

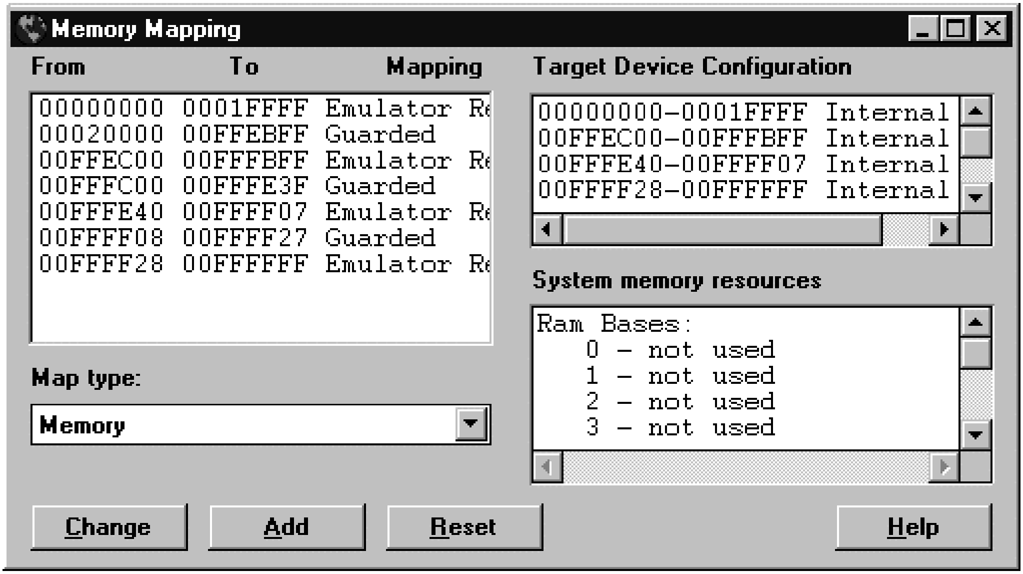
13.4.2 Edit Value

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

13.4.3 Radix

Changes the radix for the selected local variable display.

13.5 Memory Mapping



```
00000000  0001FFFF  Emulator Re
00020000  00FFEBFF  Guarded
00FFEC00  00FFFBFF  Emulator Re
00FFFC00  00FFFE3F  Guarded
00FFFE40  00FFFF07  Emulator Re
00FFFF08  00FFFF27  Guarded
00FFFF28  00FFFFFF  Emulator Re
```

```
00000000-0001FFFF Internal
00FFEC00-00FFFBFF Internal
00FFFE40-00FFFF07 Internal
00FFFF28-00FFFFFF Internal
```

```
Ram Bases:
   0 - not used
   1 - not used
   2 - not used
   3 - not used
```

Map type: Memory

Figure 13.5 Memory Mapping Window

Allows the user to view and modify the debugging platform's memory map and to view its memory configuration and resources. In some debugging platforms, the Memory Map dialog box will open.

The functionality of the command buttons is basically the same as the pop-up menu options shown below, but the pop-up menu options depend on the debugging platform.

13.5.1 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.

13.5.2 Change

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.

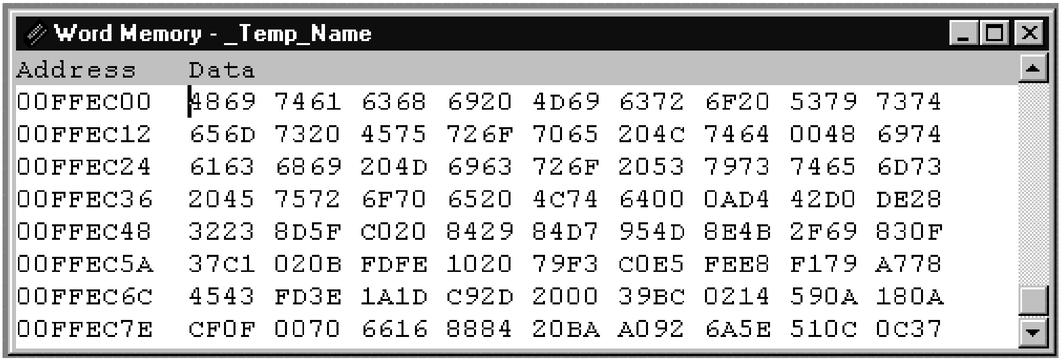
13.5.3 Reset

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

13.5.4 Help

Launches the help file.

13.6 Memory



```
Address   Data
00FFEC00  4869 7461 6368 6920 4D69 6372 6F20 5379 7374
00FFEC12  656D 7320 4575 726F 7065 204C 7464 0048 6974
00FFEC24  6163 6869 204D 6963 726F 2053 7973 7465 6D73
00FFEC36  2045 7572 6F70 6520 4C74 6400 0AD4 42D0 DE28
00FFEC48  3223 8D5F C020 8429 84D7 954D 8E4B 2F69 830F
00FFEC5A  37C1 020B FDFE 1020 79F3 C0E5 FEE8 F179 A778
00FFEC6C  4543 FD3E 1A1D C92D 2000 39BC 0214 590A 180A
00FFEC7E  CF0F 0070 6616 8884 20BA A092 6A5E 510C 0C37
```

Figure 13.6 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 an complex expression. If the data at that address cannot be modified (i.e. within ROM) 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 pop-up menu containing the following options is available by right-clicking within the window:

13.6.1 Copy

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

13.6.2 Find

Launches the Find 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.

13.6.3 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 Find option.

13.6.4 Move

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 Find option.

13.6.5 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 Find option.

13.6.6 Save

Launches the Save S-Record File 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 Find option.

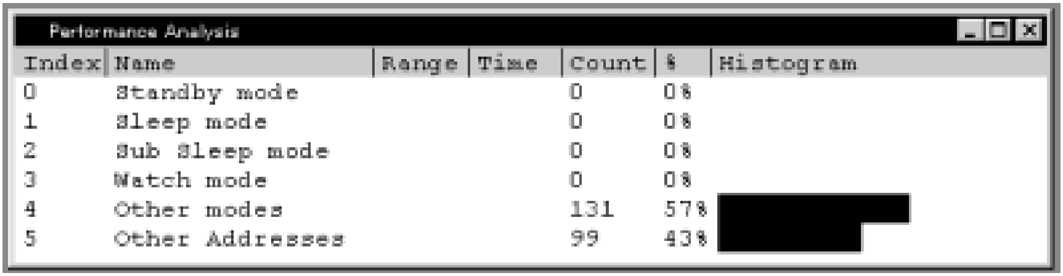
13.6.7 Set Address

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 appears for you to select a function.

13.6.8 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.

13.7 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 13.7 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 pop-up menu containing the following options is available by right-clicking within the view area:

13.7.1 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.

13.7.2 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.

13.7.3 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.

13.7.4 Reset Counts/Times

Clears the current performance analysis data.

13.7.5 Delete All Ranges

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

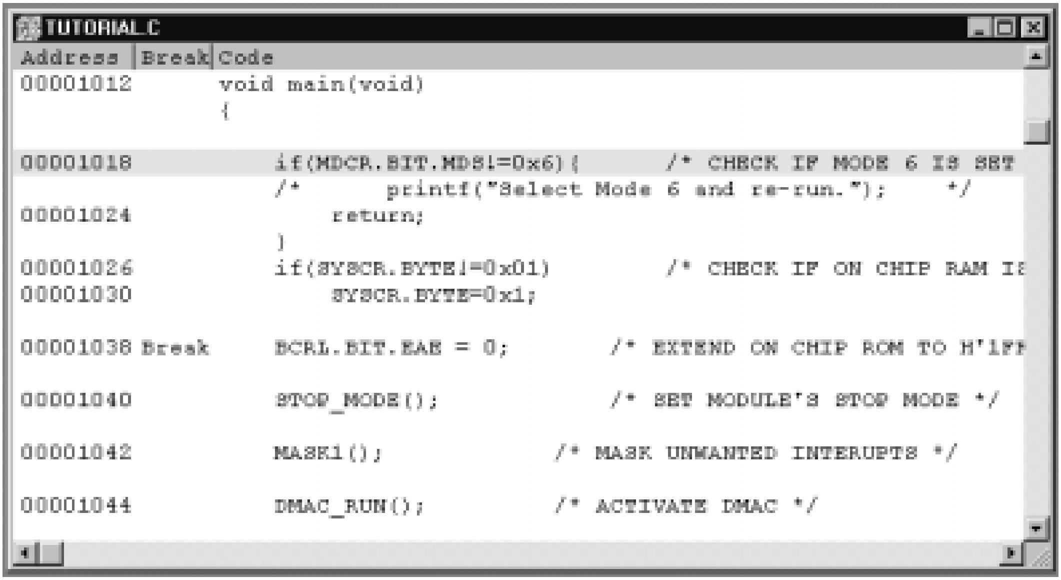
13.7.6 Analysis Enabled

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

13.8 Program

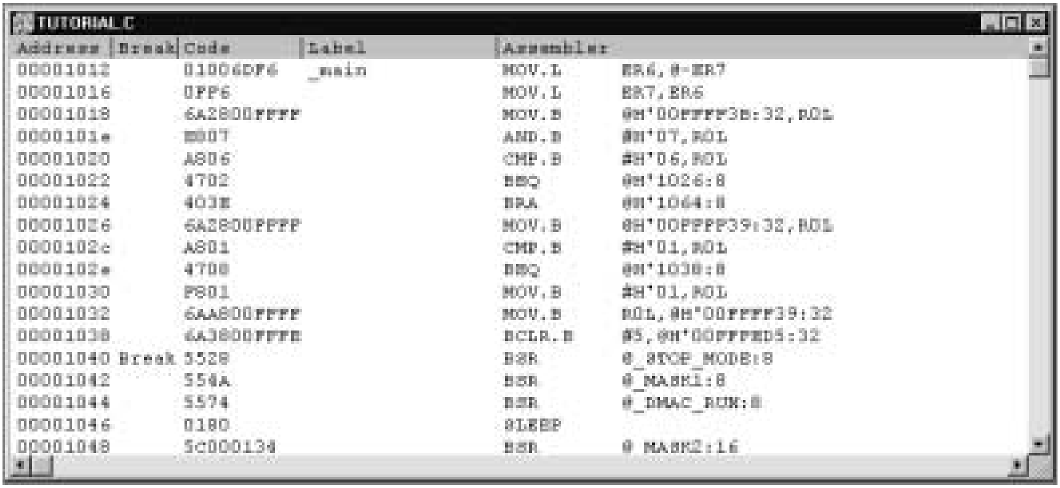
The format of this window can be set to one of three different formats - source, mixed, and assembly-language.

- The source format can be used to view any source file that was included within the absolute (*.abs) file’s debug information - this may be C/C++, assembly-language, etc.
- The mixed format has a similar layout to the source format, but also displays the instructions associated with each line of code. This is useful when stepping through a program at assembly level.
- The assembly-language format has a different layout to the other two, with an additional column Label which displays the symbol 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. (If there is no associated source file, the window’s title is “Code”.)



```
Address  |Break|Code
00001012        void main(void)
                {

00001018            if(MDCR.BIT.MDS!=0x6){        /* CHECK IF MODE 6 IS SET
                    /*       printf("Select Mode 6 and re-run.");      */
00001024                return;
                    }
00001026            if(SYSCR.BYTE!=0x01)          /* CHECK IF ON CHIP RAM IS
00001030                SYSCR.BYTE=0x1;

00001038 Break      BCRL.BIT.EAE = 0;         /* EXTEND ON CHIP ROM TO H'1FF

00001040            STOP_MODE();              /* SET MODULE'S STOP MODE */

00001042            MASK1();              /* MASK UNWANTED INTERUPTS */

00001044            DMAC_RUN();           /* ACTIVATE DMAC */
```

Figure 13.8 Source Format



```
Address  |Break|Code        |Label          |Assembler
00001012       01006DF6     _main           MOV.L      ER6,@-ER7
00001016       0FF6                         MOV.L      ER7,ER6
00001018       6A2800FFFF                   MOV.B      @H'00FFFF3B:32,R0L
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
00001026       6A2800FFFF                   MOV.B      @H'00FFFF39:32,R0L
0000102c       A801                         CMP.B      #H'01,R0L
0000102e       4708                         BEQ        @H'1038:8
00001030       F801                         MOV.B      #H'01,R0L
00001032       6AA800FFFF                   MOV.B      R0L,@H'00FFFF39:32
00001038       6A3800FFFE                   BCLR.B     #5,@H'00FFFED5:32
00001040 Break 5528                         BSR        @_STOP_MODE:8
00001042       554A                         BSR        @_MASK1:8
00001044       5574                         BSR        @_DMAC_RUN:8
00001046       0180                         SLEEP
00001048       5C000134                     BSR        @_MASK2:16
```

Figure 13.9 Assembly-Language Format

All formats support column-specific double-click actions:

- 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 window will be opened in assembly-language format and entitled ‘Code’. When an overloaded function or a class name is entered, the Select Function dialog box appears for you to select a function.

- Break - Sets/clears a program (PC) breakpoint at that address.
- Code - (Assembly-language format only.) 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 - (Assembly-language format only.) Launches the Label dialog box, allowing the user to enter a new label, or to clear or edit the name of an existing label.
- Assembler - (Assembly-language format only.) Launches the Assembler dialog box allowing the user to modify the instruction at that address. Note that changes to the instruction do not modify the source file, and any changes will be lost at the end of the session.

A pop-up menu containing the following options is available by right-clicking within the window:

13.8.1 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.

13.8.2 Find

Launches the Find dialog box, allowing the user to search the source file for a string. It is only available in the source view format.

13.8.3 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. It is only available in the pure source view format.

13.8.4 Go To Cursor

Commences to execute the user code 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.

13.8.5 Set PC Here

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

13.8.6 Toggle Breakpoint

Sets/clears a software breakpoint at the address indicated by the text cursor (not the mouse cursor). This is equivalent to double-clicking within the Break column. (Additional break point types may be available depending on the selected debugging platform - in this case, the list will be cycled through.)

13.8.7 Instant Watch

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

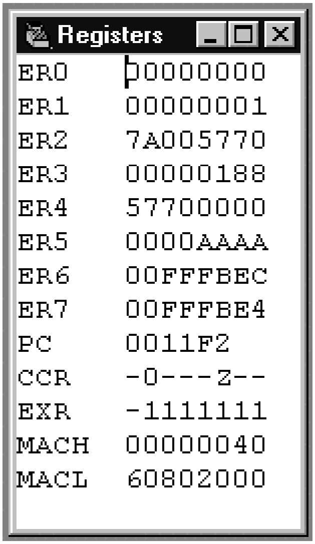
13.8.8 Add Watch

Only available in the source and mixed formats. 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.

13.8.9 Source/Mixed/Assembler

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

13.9 Registers



```
ER0     00000000
ER1     00000001
ER2     7A005770
ER3     00000188
ER4     57700000
ER5     0000AAAA
ER6     00FFFBEC
ER7     00FFFBE4
PC      0011F2
CCR     -0---Z--
EXR     -1111111
MACH    00000040
MACL    60802000
```

Figure 13.10 Registers Window

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

A pop-up menu containing the following options is available by right-clicking within the window:

13.9.1 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.

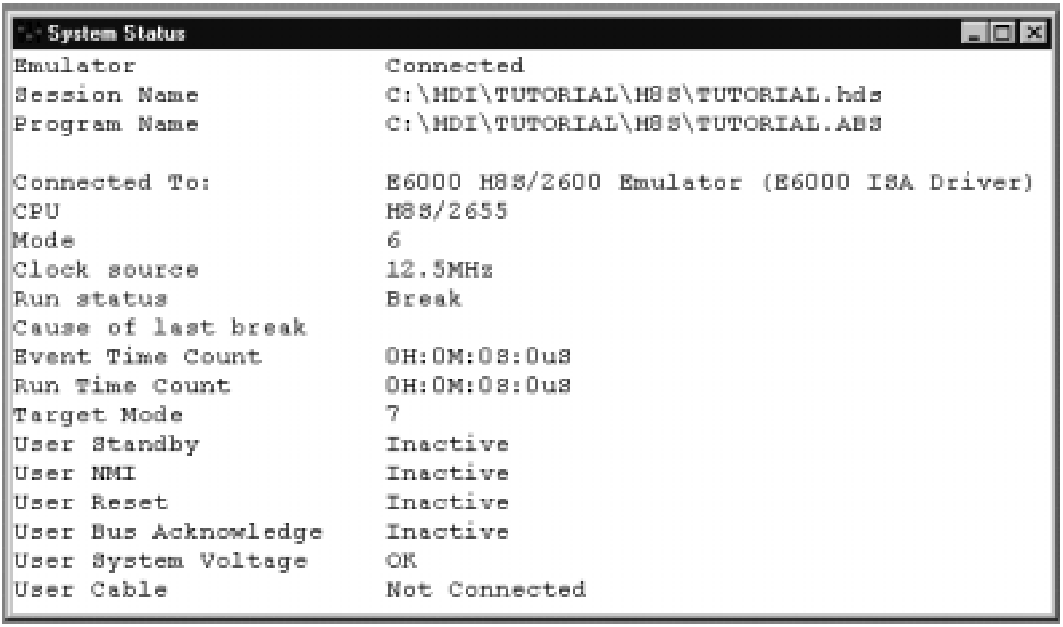
13.9.2 Edit

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

13.9.3 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.

13.10 System Status



```
Emulator                  Connected
Session Name              C:\MDI\TUTORIAL\H8S\TUTORIAL.hds
Program Name              C:\MDI\TUTORIAL\H8S\TUTORIAL.ABS

Connected To:             E6000 H8S/2600 Emulator (E6000 ISA Driver)
CPU                       H8S/2655
Mode                      6
Clock source              12.5MHz
Run status                Break
Cause of last break
Event Time Count          0H:0M:0S:0uS
Run Time Count            0H:0M:0S:0uS
Target Mode               7
User Standby              Inactive
User NMI                  Inactive
User Reset                Inactive
User Bus Acknowledge      Inactive
User System Voltage       OK
User Cable                Not Connected
```

Figure 13.11 System Status Window

Allows the user to view the current status of the debugging platform. The text consists of two sections - a standard section (from Emulator to Cause of last break) and a debugging platform dependent section. See the supplied *Debugging Platform User's Manual* for more information about the latter section.

A pop-up menu containing the following options is available by right-clicking within the window:

13.10.1 Update

Updates the displayed data.

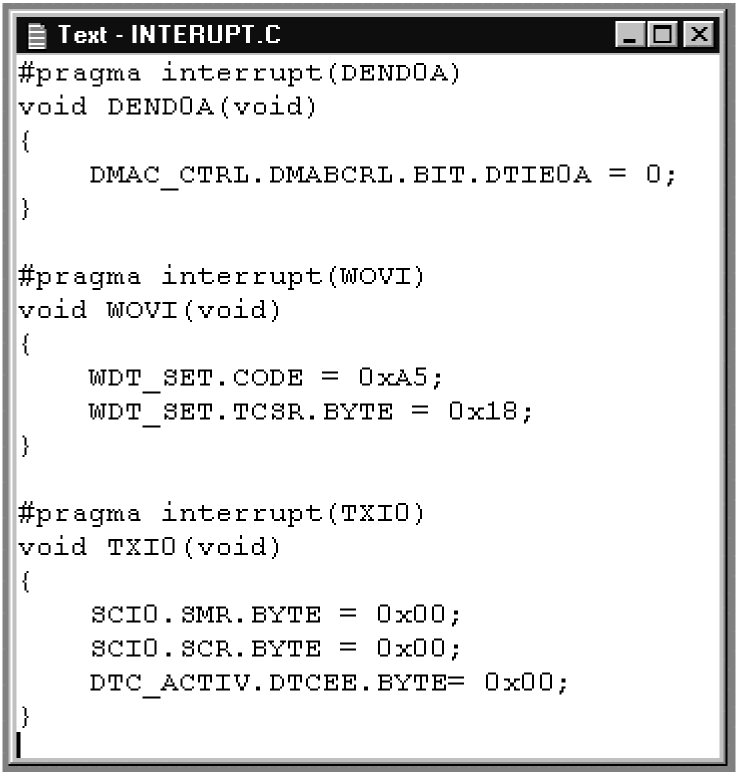
13.10.2 Copy

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.

13.10.3 Configure...

Launches a setup dialog box specific to the selected debugging platform. Refer to the supplied *Debugging Platform User's Manual* for more detail about the options available in the dialog box.

13.11 Text



```
#pragma interrupt(DEND0A)
void DEND0A(void)
{
    DMAC_CTRL.DMABCRL.BIT.DTIE0A = 0;
}

#pragma interrupt(WOVI)
void WOVI(void)
{
    WDT_SET.CODE = 0xA5;
    WDT_SET.TCSR.BYTE = 0x18;
}

#pragma interrupt(TXI0)
void TXI0(void)
{
    SCI0.SMR.BYTE = 0x00;
    SCI0.SCR.BYTE = 0x00;
    DTC_ACTIV.DTCEE.BYTE= 0x00;
}
```

Figure 13.12 Text Window

Allows the user to view a text file, e.g. a log file or source code without debugging information available. The window is read-only.

A pop-up menu containing the following options is available by right-clicking within the window:

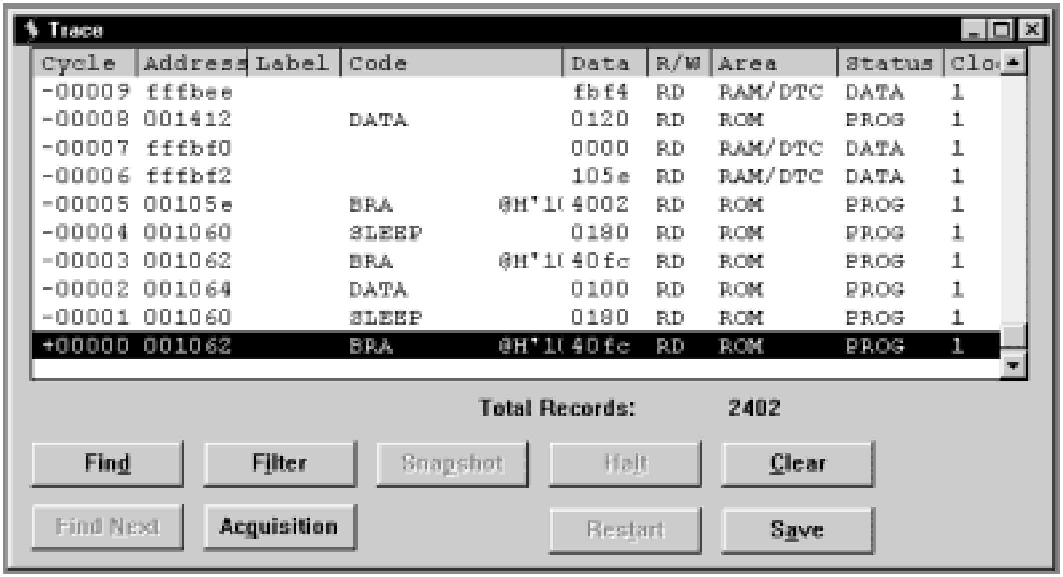
13.11.1 Copy

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 text file for a string.

13.12 Trace



Trace

Cycle	Address	Label	Code	Data	R/W	Area	Status	Clo
-00009	fffbee			fbf4	RD	RAM/DTC	DATA	1
-00008	001412		DATA	0120	RD	ROM	PROG	1
-00007	fffbf0			0000	RD	RAM/DTC	DATA	1
-00006	fffbf2			105e	RD	RAM/DTC	DATA	1
-00005	00105e		BRA @H'1(	4002	RD	ROM	PROG	1
-00004	001060		SLEEP	0180	RD	ROM	PROG	1
-00003	001062		BRA @H'1(	40fc	RD	ROM	PROG	1
-00002	001064		DATA	0100	RD	ROM	PROG	1
-00001	001060		SLEEP	0180	RD	ROM	PROG	1
+00000	001062		BRA @H'1(	40fc	RD	ROM	PROG	1

Total Records: 2402

Figure 13.13 Trace Window

Allows the user to view the sequence of instructions leading up to the debugging platform’s current status.

The functionality of the command buttons is identical to the pop-up menu options shown below.

13.12.1 Find

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

13.12.2 Find Next

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

13.12.3 Filter

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

13.12.4 Acquisition

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

13.12.5 Halt

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

13.12.6 Restart

Starts tracing data.

13.12.7 Snapshot

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

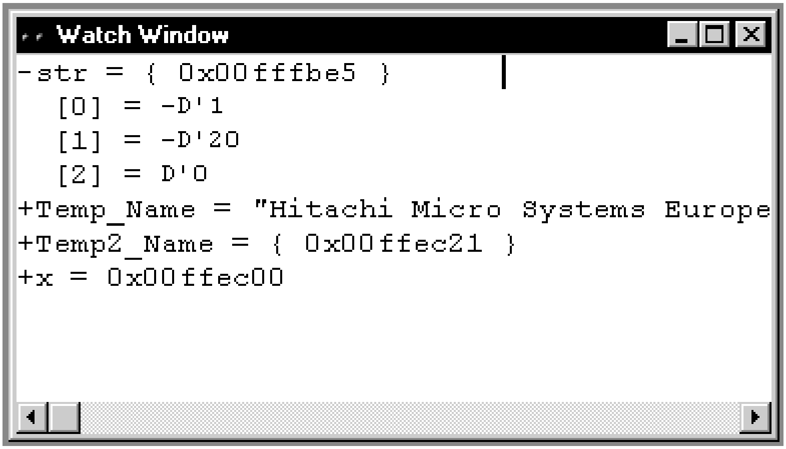
13.12.8 Clear

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.

13.12.9 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 Watch



```
-str = { 0x00fffbe5 }
  [0] = -D'1
  [1] = -D'20
  [2] = D'0
+Temp_Name = "Hitachi Micro Systems Europe
+Temp2_Name = { 0x00ffec21 }
+x = 0x00ffec00
```

Figure 13.14 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 code 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.

A pop-up menu containing the following options is available by right-clicking within the windows:

13.13.1 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.

13.13.2 Delete Watch

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

13.13.3 Add Watch

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

13.13.4 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.

13.13.5 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.

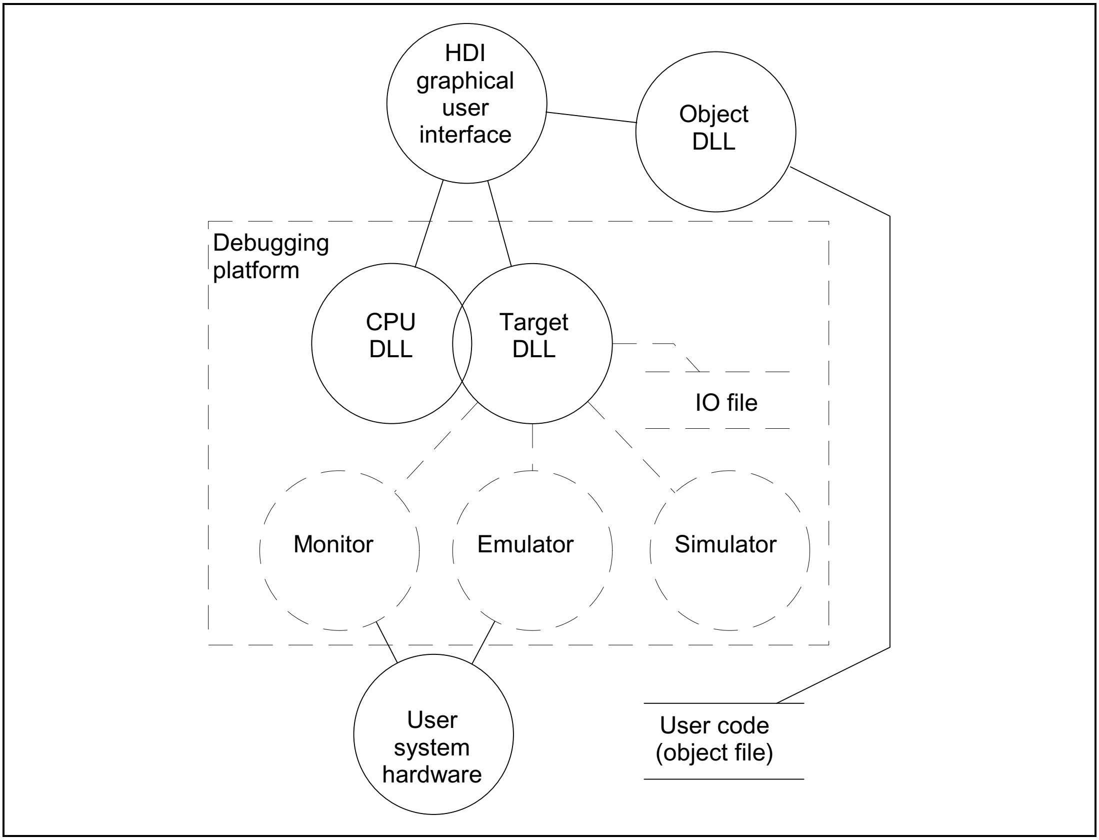


Figure A.1   HDI System Modules

In normal operation, the user code 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 code. It also provides the interface between the module DLLs and the host file system, i.e., the PC.

Object DLL

When creating the user code, 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 code 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 Program 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 <expression>

Parameter	Type	Description
<expression>	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 <address> [<length>]

Parameter	Type	Description
<address>	Numeric	Start address
<length>	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'00000000;00000000;00000001;01101110
EV #R2H * #R2L	Result:	H'121 D'289 O'441 B'00000000;00000000;00000001;00100001

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 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	Copies 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 (inclusive)

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>Sets step rate (1-6), 6 = fastest

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.UDC.
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

Long name	Short name	Description
MEMORY_DISPLAY	MD	Displays memory contents
MEMORY_EDIT	ME	Modifies memory contents
MEMORY_FILL	MF	Fills a memory area
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.)

Hitachi Debugging Interface User’s Manual

Publication Date: 1st Edition, November 1997
2nd Edition, February 1999
Published by: Electronic Devices Sales & Marketing Group
Semiconductor & Integrated Circuits Group
Hitachi, Ltd.
Edited by: Technical Documentation Group
UL Media Co., Ltd.