ARM Software Development Toolkit Version 2.0

Windows Toolkit Guide

Document Number: ARM DUI 0022B
Issued: June 1995

EUROPE
Advanced RISC Machines Limited
Fulbourn Road
Cherry Hinton
Cambridge CB1 4JN
Telephone: +44 1223 400400
Facsimile: +44 1223 400410
Email: info@armltd.co.uk

JAPAN
Advanced RISC Machines K.K.
KSP West Bldg, 3F 300D, 3-2-1 Sakado,
Takatsu-ku, Kawasaki-shi
Kanagawa, 213 Japan
Telephone: +81 44 850 1301
Facsimile: +81 44 850 1308
Email: info@armltd.co.uk

USA
ARM USA
Suite 5, 985 University Avenue
Los Gatos
California 95030
Telephone: +1 408 399 5199
Facsimile: +1 408 399 8854
Email: info@arm.com

Proprietary Notice

Change Log

Issue	Date	By	Change
A draft1	Mar 95	AW	Created
A	Apr 95	AW	Review comments incorporated. Standard windows terminology used. Beta release.
B	Jun 95	AP	Added remote debug chapter and extra worked example. First formal release.

TOC

Contents

1 Introduction **1-1**

1.1 About this Manual 1-2
1.2 About the Toolkit 1-3
1.3 Feedback 1-5
1.4 Overview of the Windows Toolkit 1-6
1.5 Options Within the Windows Toolkit 1-7
1.6 Using Online Help 1-8

2 Project Manager **2-1**

2.1 Overview 2-2
2.2 Entering the Project Manager 2-3
2.3 Creating or Opening a Project 2-4
2.4 Managing Project Files 2-5
2.5 Setting Options 2-7
2.6 Building a Project 2-14
2.7 Dependencies within a Project 2-15
2.8 Entering the Debugger from the Project Manager 2-15

Contents

3	**Debugger**		**3-1**
	3.1	Overview	3-2
	3.2	Accessing the Debugger	3-3
	3.3	Loading and Displaying the Image	3-3
	3.4	Executing the Image	3-4
	3.5	Breakpoints and Watchpoints	3-6
	3.6	Viewing Variables	3-9
	3.7	Setting Expressions	3-10
	3.8	Viewing Registers	3-10
	3.9	Viewing a Backtrace	3-11
4	**Remote Debugging**		**4-1**
	4.1	Overview	4-2
	4.2	ARM6 PIE Card	4-3
	4.3	ARM7 PIE Card	4-6
	4.4	EmbeddedICE	4-8
5	**Worked Example**		**5-1**
	5.1	Example 1: Dhrystone 2.1 Benchmark	5-2
	5.2	Example 2: Software Development Example	5-10
A	**Project Manager Options**		**A-1**
	A.1	File Menu	A-2
	A.2	Edit Menu	A-3
	A.3	View Menu	A-4
	A.4	Project Menu	A-5
	A.5	Options Menu	A-7
	A.6	Windows Menu	A-14
	A.7	Help Menu	A-15
B	**Debugger Options**		**B-1**
	B.1	File Menu	B-2
	B.2	Edit Menu	B-3
	B.3	Search Menu	B-4
	B.4	View Menu	B-5
	B.5	Execute Menu	B-8
	B.6	Options Menu	B-9
	B.7	Item Menu	B-11
	B.8	Window Menu	B-12
	B.9	Help Menu	B-13

1 Introduction

This chapter introduces the Windows Toolkit.

1.1	About this Manual	1-2
1.2	About the Toolkit	1-3
1.3	Feedback	1-5
1.4	Overview of the Windows Toolkit	1-6
1.5	Options Within the Windows Toolkit	1-7
1.6	Using Online Help	1-8

Introduction

1.1 About this Manual

1.1.1 Assumptions

This manual assumes that the reader has some knowledge of using a Windowing environment. It does not require prior knowledge of any of the ARM products.

For further information about the command-line tools referred to in this manual, see the Software Development Toolkit Reference Manual (ARM DUI0020).

1.1.2 Conventions

The following typographical conventions are used in this manual:

`typewriter`	Denotes text that may be entered at the keyboard: commands, file and program names and assembler and C code.
`typewriter-italic`	Shows text which must be substituted for user-supplied information: this is most often used in syntax descriptions.
Oblique	Highlights important notes and ARM-specific terminology.
bold	Denotes menu options and dialog box field names.

1.2 About the Toolkit

1.2.1 Programming tools

The following gives a brief description of the tools used by the Windows Toolkit.

armcc	The ARM C compiler, which compiles C source into 32-bit ARM code.
tcc	The Thumb C compiler, which compiles C source into 16-bit Thumb code.
armasm	The ARM assembler compiles ARM Assembly Language into ARM Object Format object code.
tasm	The Thumb assembler compiles both ARM and Thumb Assembly Language into ARM Object Format object code.
armlink	The ARM linker combines the contents of one or more object files (the output of the compiler or assembler) with selected parts of one or more object libraries, to produce an executable program.
armsd	The ARM symbolic debugger is used to debug programs assembled or compiled using the ARM assembler and the ARM C compiler.
decaof	The ARM-Thumb object-file decoder/disassembler decodes ARM Object Format files such as those produced by the ARM assembler or ARM C compiler.

1.2.2 Hardware definitions

The following hardware definitions apply for this manual:

ARM6 PIE card	A compact card which serves as a target board for the development of a RISC processor-based embedded system. It is based around the ARM60 processor which has been designed specifically for embedded applications.
ARM7 PIE card	A compact card which serves as a target board for the development of a RISC processor-based embedded system. It is based around the ARM70DI processor and supports both serial and JTAG connections to a host.
EmbeddedICE	The EmbeddedICE unit converts data from a serial/parallel port to data for a JTAG port.

1.2.3 Filenaming conventions

The following filenaming conventions are used in the Windows Toolkit:

`.s`	ARM/Thumb assembler source file
`.c`	C source file
`.cpp`	C++ source file
`.o`	Object file (either ARM or Thumb)
`.h`	C header file
`.32l`	Library designed for 32-bit ARM instructions in little endian mode
`.32b`	Library designed for 32-bit ARM instructions in big endian mode
`.16l`	Library designed for 16-bit Thumb instructions in little endian mode
`.16b`	Library designed for 16-bit Thumb instructions in big endian mode

1.3 Feedback

1.3.1 Feedback on the Windows Toolkit

If you have feedback on the Windows Toolkit, please contact either your supplier, or ARM Ltd. You can send feedback via email to: tools200@armltd.co.uk.

In order to help us to provide a rapid and useful response, please give:

- details of which release of the Windows Toolkit you are using
- details of which platform you are running on
- a small stand-alone sample of code which reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used (including any command-line options)
- sample output illustrating the problem
- the version string of the tool (including the version number and date)

1.3.2 Feedback on this manual

If you have feedback on this manual, please send it via email to: documentation@armltd.co.uk, giving:

- the manual's title and revision
- the page number(s) to which your comments refer
- a concise explanation of the problem

General suggestions for additions and improvements are also welcome.

1.4 Overview of the Windows Toolkit

The Windows Toolkit consists of two applications:

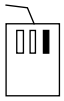
Project Manager	Allows you to build your source code into image files or libraries. You can perform all of your code writing within the Project Manager.
Debugger	Allows you to debug your source files.

This manual covers the following topics within these applications:

- creating projects
- building and linking projects
- editing source files
- setting compiler, assembler, linker and project options
- displaying an executable image
- running an executable image
- setting breakpoints and watchpoints
- viewing variables, expressions and registers
- using the online help
- worked examples

1.5 Options Within the Windows Toolkit

You can choose many of the Windows Toolkit options in several ways:

Pulldown menus	These menus contain the Windows Toolkit options. You can choose any of the Windows Toolkit operations using one or more of these options.
Toolbar	This menu bar contains the more commonly used options. All of the options found here can also be found on the pulldown menus. Where toolbar options are available, this is indicated in the appendices.
 Context menu	This is a context-sensitive menu which can be opened by clicking the right mouse button. Many of the commands you need to use appear on this menu when available.
Function keys	Some of the menu operations are also available using function keys. Available function key shortcuts are indicated in the appendices.

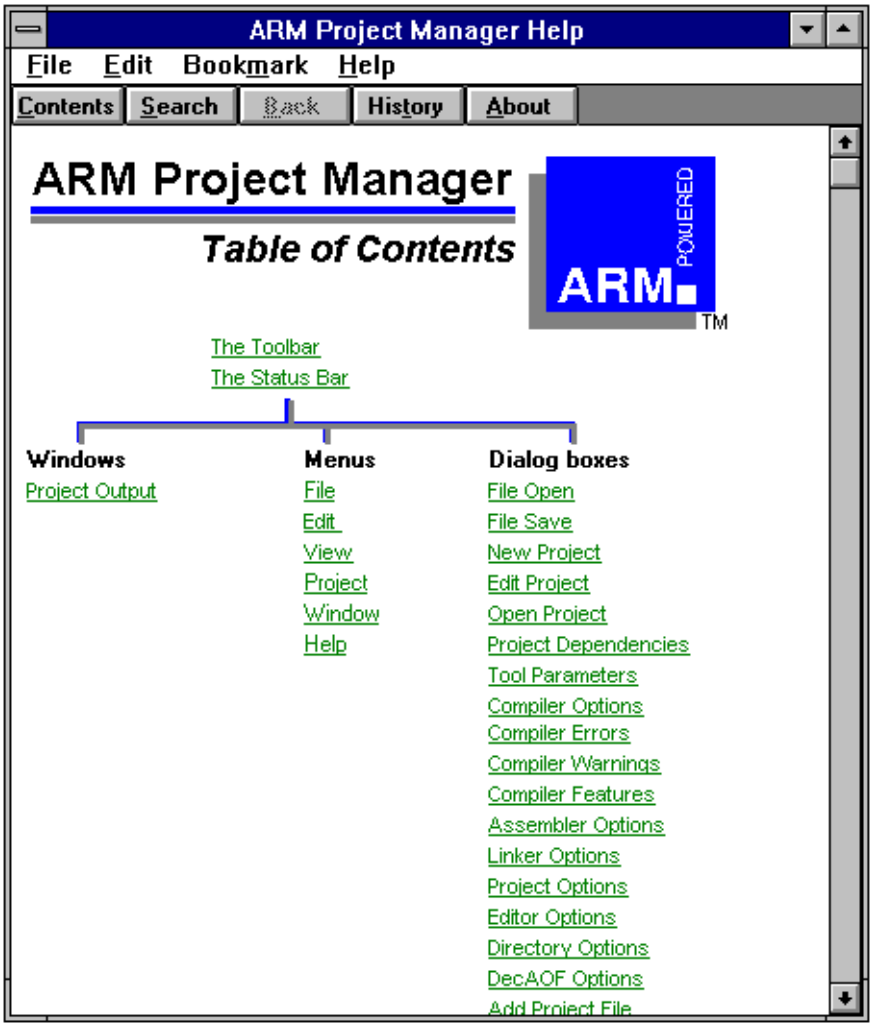
Introduction

1.6 Using Online Help

There is an extensive online help system in both the Project Manager and the Debugger.

To access the online help, select the **Index** option from the Help menu. You can then select the subject area for which you require help. Alternatively, press **F1** to access help on the area you are currently working with. You can also access this context sensitive help by clicking on the **Help** button in a dialog box.

For more information about how to use the online help, choose **Using Help** from the Help menu.



2 Project Manager

This chapter describes how to create, build and link a project using the Project Manager.

2.1	Overview	2-2
2.2	Entering the Project Manager	2-3
2.3	Creating or Opening a Project	2-4
2.4	Managing Project Files	2-5
2.5	Setting Options	2-7
2.6	Building a Project	2-14
2.7	Dependencies within a Project	2-15
2.8	Entering the Debugger from the Project Manager	2-15

Project Manager

2.1 Overview

The ARM Project Manager is an easy-to-use graphical front-end for the ARM command-line tools. It allows you to build a single target (usually an executable image for loading into the Debugger) by storing a list of files that constitute a project.

2.1.1 Project file types

A project consists of a `.apj` file which references a collection of files. You can include the following types of file in a project:

- source files (`.c`, `.cpp`, `.s`)
- object files (`.o`)
- libraries (`.lib`, `.16l`, `.16b`, `.32l`, `.32b`)

Note: *Header files (`.h`) are automatically included in the project by the Project Manager.*

2.1.2 Operations

You can perform the following operations using the Project Manager:

- create a new project (see *2.3 Creating or Opening a Project* on page 2-4)
- open an existing project (see *2.3 Creating or Opening a Project* on page 2-4)
- add files to a project (see *2.4.1 Adding files* on page 2-5)
- edit source files within a project (see *2.4.3 Editing files within a project* on page 2-6)
- build your source code into an executable image or a library (see *2.6 Building a Project* on page 2-14)
- display a summary of a project (see *2.4.4 Displaying a project summary* on page 2-6)
- scan or show the file dependencies within the project (see *2.7 Dependencies within a Project* on page 2-15)
- set global and local build options (see *2.5 Setting Options* on page 2-7)
- enter the Debugger and execute the built image (see *2.8 Entering the Debugger from the Project Manager* on page 2-15)
- change the project options (see *2.5.1 Setting project options* on page 2-7)
- review the code size of the project (see *2.6 Building a Project* on page 2-14)

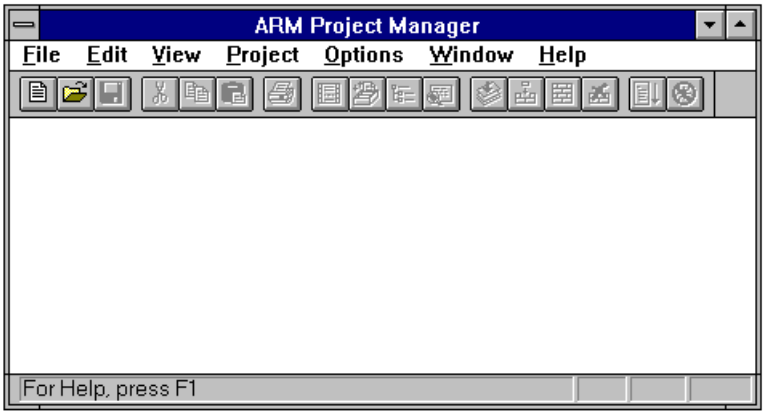
2.1.3 Shortcuts

Throughout this chapter, the selection of various options is described by making a selection from the Pulldown menus or Context menu. Note that many of these options are also available on the Toolbar and the function keys. These shortcuts for selecting options are given beside the options and their description in *Appendix A, Project Manager Options*.

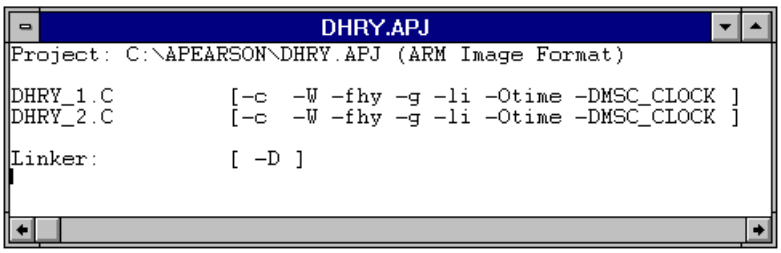
2.2 Entering the Project Manager

You can access the Project Manager by double-clicking on the Project Manager icon shown on the left. This is usually located in the ARM Toolkit window.

When you first enter the Project Manager, the following window is displayed.



If you do not have a project open, the window will be empty. If an existing project is open, a short project summary is displayed. A sample project summary is shown below.



The ARM Project Manager invokes:

- an ARM compiler (armcc, armcpp or tcc)
- an ARM assembler (armasm or tasm)
- the ARM linker (armlink) or the ARM library manager (armlib)

The choice of compiler and assembler and the choice of using the linker or the library manager depends on the project options you specify. For more information, refer to *2.5.1 Setting project options* on page 2-7.

2.3 Creating or Opening a Project

To create a new project

1. Choose **New** from the Project Manager.

   The New Project dialog box appears.

2. Enter a filename for your new project and click on **OK**. The filename extension `.apj` is added automatically, overriding any extension you specify.

A project with your specified filename is created and the Edit Project dialog box appears. Initially your new project does not contain any files. You must add your required files to the project using **Add**. This is described in *2.4.1 Adding files*.

To open an existing project

1. Choose **Open** from the Project menu.

   The Open Project dialog box appears.

2. Highlight the project you wish to open and click on **OK**.

The specified project is opened and a project summary is displayed.

2.4 Managing Project Files

You may have already created your source files with a different editor. However, you can also create or edit source files using the Project Manager's document editing options. These options can be found on the File and Edit Pulldown menus and are very similar to most other Windows Applications. See *Appendix A, Project Manager Options*. The Project Manager allows you to specify a default editor. You can specify:

- the Toolkit's own integrated editor with basic editing facilities
- the commercial editor CodeWright with more sophisticated code editing facilities

When you have written your source files, you can create a new project and add your files (see *2.3 Creating or Opening a Project*) or you can add your files to an existing project (see *2.4.1 Adding files*). You can also edit your files once you have added them to your project (see *2.4.3 Editing files within a project* on page 2-6).

2.4.1 Adding files

To add a file to your project:

1. Choose **Edit** from the Project menu.
   The Edit Project dialog box appears.
2. Click on the **Add...** button.
   The Add Project File dialog box appears.
3. Highlight the file(s) you wish to add and click on **OK**.

The specified files are added to your project and the project summary is updated to reflect the change.

Note: *For a list of types of files you can add to your project, see 2.1.1 Project file types* on page 2-2.

2.4.2 Removing files from a project

To remove a file from your project:

1. Choose **Edit** from the Project menu.
   The Edit Project dialog box appears.
2. Highlight the file(s) you wish to remove and click on **Remove**.
   The selected file is removed from your project and the project summary is updated to reflect the change.

2.4.3 Editing files within a project

To edit a file that is already in a project:

1. Choose **Edit** from the Project menu.

   The Edit Project dialog box appears.

2. Highlight the file you wish to edit and click on **Edit Selected**.

3. Edit the specified files using the default editor. If you wish to change your default editor, refer to *2.5.6 Setting editor options* on page 2-12. When you have completed your edits, choose **Close** from either the File or Project menus.

   You are prompted whether you wish to save the changes.

4. Save the changes by clicking on **Yes**.

Alternatively, you can display the project summary and double-click on the file you wish to edit.

Note: *If you choose* **Close** *from the Project menu when no file is open, your project will be closed. Before you can perform any other operations within your project, you must re-open it. To open your project, see 2.3 Creating or Opening a Project* on page 2-4*.*

2.4.4 Displaying a project summary

You can display a project summary at any time by choosing **Show** from the Project menu.

2.5 Setting Options

Before building a project, you should set your global options:

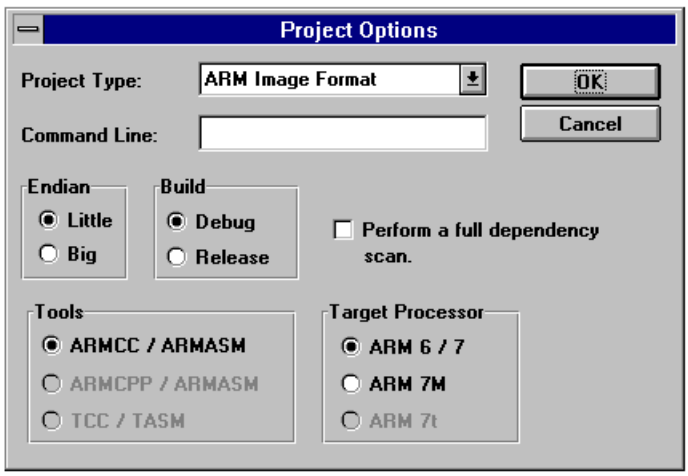
- project (see ⧉*2.5.1 Setting project options*)
- compiler (see ⧉*2.5.2 Setting compiler options* on page 2-9)
- assembler (see ⧉*2.5.3 Setting assembler options* on page 2-10)
- linker (see ⧉*2.5.4 Setting linker options* on page 2-11)
- decoder/disassembler (see ⧉*2.5.5 Setting decoder/disassembler options* on page 2-11)
- editor (see ⧉*2.5.6 Setting editor options* on page 2-12)
- directories (see ⧉*2.5.7 Setting options on directories* on page 2-12)

You can set global file options (compiler, assembler, linker and decoder/disassembler) which will apply to every file in the project. You should set these to be the most common options. You can add to them on a per file basis if you wish by editing the individual file parameters. See ⧉*2.5.8 Editing file parameters* on page 2-13, for a full description of how to do this.

2.5.1 Setting project options

You can set up your project options as follows:

1 Choose **Project...** from the Options menu.
The following dialog box appears:



Note: *The tools ARMCPP/ARMASM and TCC/TASM may be disabled if you have not licensed ARM C++ or Thumb tools.*

2 Enter the project type. This determines whether the project creates a library or an image (AOF).

3 In the **Command Line** field, specify any arguments required by the image, when it is run in the Debugger. For more information, see the ARM Software Development Toolkit Reference Manual (ARM DUI 0020).

4 Select whether to use little-endian or big-endian mode.

5 Select your build version:

Debug	This option uses the `-g` command-line option for the compiler/assembler.
Release	This option generates fully optimised code with no debug information. The linker will still generate debug information unless you use the `-nodebug` option.

6 Select the appropriate tools and the target processor.

`.c` or `.cpp` files are always compiled using armcc, armcpp or tcc, depending upon the setting of the **Tools** option.

`.s` files are always compiled using armasm or tasm.

If the tools TCC/TASM are selected, the target processor must be ARM7t.

7 If you wish to scan the file dependency hierarchy before building the project, select the **Perform a Full Dependency Scan** check box.

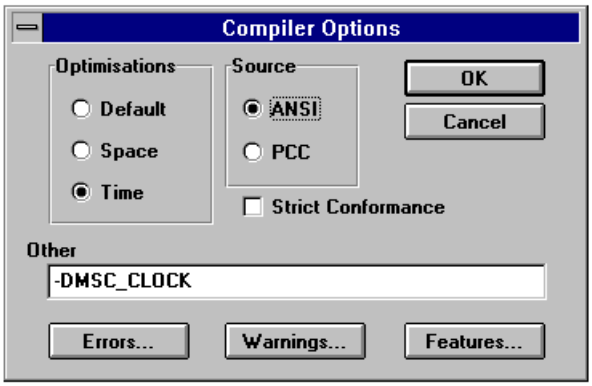
A full dependency scan opens each source file to determine included headers. Without a full dependency scan, each object is only rebuilt if the respective source file has been modified (there is no check to see if included headers have changed). A full dependency scan will be needed in most circumstances.

2.5.2 Setting compiler options

Set up your global compiler options as follows:

1 Choose **Compiler...** from the Options menu.

The following dialog box appears.



2 Select the Optimisation option that you require:

Default Balances the following two options.

Space Performs optimisations to reduce image size at the expense of increased execution time.

Time Performs optimisations to reduce execution time at the expense of a larger image.

3 Select the type of C in which your source files are written:

ANSI Source is ANSI C standard.

PCC Source is K&R old-style (PCC) C.

If you wish your code to be *strictly* conformant to ANSI or PCC, select the **Strict Conformance** check box directly below the Source options.

4 You can suppress errors during the build process. To do this for the most common errors, click on **Errors...** and mark the types of error you wish to suppress. The types are described in ➲*Appendix A, Project Manager Options*.

When you have marked the types of error you wish to suppress, click on **OK**.

5 To suppress the most common warnings during the build process click on **Warnings...** and mark the types of warning that you wish to suppress. The types are described in ➲*Appendix A, Project Manager Options*.

When you have marked the types of warning you wish to suppress, click on **OK**.

6 You can set up other global compiler features by clicking on **Features...**. These options are described in ➲*Appendix A, Project Manager Options*.

When you have marked the types of feature you wish to include, click on **OK**.

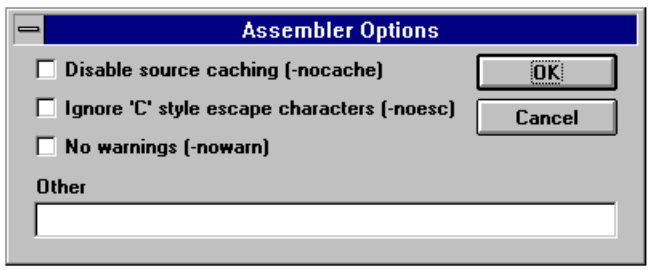
7 The most common errors, warnings and features are listed in the above dialog boxes. You can add further options using the **Other** option on the dialog box in the same way as you would when typing parameters at the command-line. The options available are described in the Software Development Toolkit Reference Manual (ARM DUI 0020).

2.5.3 Setting assembler options

To set up your global assembler options:

1 Choose **Assembler...** from the Options menu.

The following dialog box appears.



2 Select the options as required:

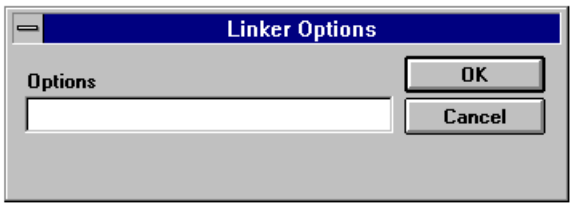
Disable source caching	Turns off source caching, (default is on). This is equivalent to the `-NOCache` command-line option.
Ignore 'C' style escape characters	Ignores C-style special characters ('\n', '\t' etc.). This is equivalent to the `-NOEsc` command-line option.
No warnings	Turns off warning messages. This is equivalent to the `-NOWarn` command-line option.

3 You can add further options in the **Other** field in the same way as you would on the command-line. The options available are described in the Software Development Toolkit Reference Manual (ARM DUI 0020).

2.5.4 Setting linker options

You can set up your global linker options by choosing **Linker...** from the Options menu.

The following dialog box appears.



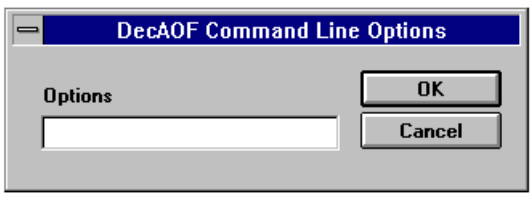
Note: *Ensure that you set up your linker options* before *building your project; if your project builds without any errors, the linker is automatically invoked.*

You can add your options in the same way as you would on the command-line. The available options are described in the Software Development Toolkit Reference Manual (ARM DUI 0020).

2.5.5 Setting decoder/disassembler options

DecAOF (Decoder for ARM Object Format) is the ARM object file disassembler. It can display information stored in the object file.

You can set the DecAOF options by choosing **DecAOF...** from the Options menu. The following dialog box appears.



You can add options in the same way as you would on the command-line. The available options are described in the Software Development Toolkit Reference Manual (ARM DUI 0020).

2.5.6 Setting editor options

To set the editor options, choose **Editor...** from the Options menu.

The following dialog box appears.

Tab stops	Specifies the tab settings.
Font...	Allows you to specify the text font, style and size.
Use CodeWright	Specifies that you wish to use CodeWright rather than the Toolkit's integrated editor as your default editor. CodeWright must have DDE enabled. See the Release Notes for information on enabling DDE.
Location	Specifies the location of the CodeWright executable.
Browse	Allows you to select your location by viewing the directory hierarchy.

2.5.7 Setting options on directories

You can set the following options by choosing **Directories...** from the Options menu. These options are initially set by the installer, but you may need to alter them if you change the location of the Project Manager, Libraries or Tools.

Project Manager	Specifies the location of the Project Manager.
Libraries	Specifies the location of the libraries.
Tools	Specifies the location of the selected tools.

Note: *If you are using Windows 3.1, the tools directory should be the location of the DOS-extended tools (the DOS directory).*
If you are using Windows95 or WindowsNT, the tools directory should be the location of the Win32 console tools (the BIN directory).

2.5.8 Editing file parameters

You can set global options for all of the files or you can set specific options for individual files. To set global options, refer to *2.5 Setting Options* on page 2-7.

To add further options for a specific file:

1. Choose **Edit** from the Project menu.
2. Highlight the file you wish to edit in the Edit Project dialog box and click on the **Edit Params...** button.

   The Edit Params dialog box appears. You can now enter parameters for the file in the same way as you would on the command-line.

2.6 Building a Project

Once you have added the files you require and set up the global options and parameters for specified files, you are ready to build your project.

To do this choose **Build** or **Rebuild All** from the Project menu.

Build Checks for any source files which have been edited or added since your last build. These files are then built into your project.

Rebuild All Rebuilds your entire project.

The Project Manager scans the file dependencies and then starts building. As it does so, it will display any warnings or errors that you have not set to be suppressed.

If errors are encountered during the build process, the errors continue to be displayed when building is complete and a message is displayed to indicate that the build was unsuccessful. A summary of the number of warnings, errors and serious errors is also given.

You can now move to the location of each error in turn to make corrections to your files. Before you do this, check the setting of all your options to ensure they are correct as this may be a cause of the problems.

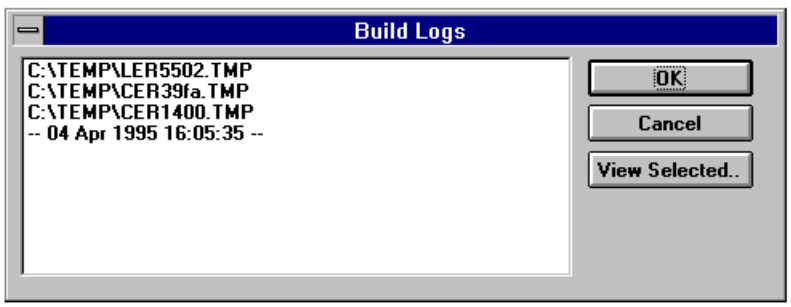
To move to the location of an error, double-click on the error message. You are taken into your default editor at the location of the error.

Note: *Depending on the type of error, you may find the error on the line in the file which is highlighted or possibly the line above.*

Move through the errors, editing your files to correct them and use **Build** or **Rebuild All** on the Project menu to rebuild your project.

2.6.1 Viewing the build logs

You can view the build logs by choosing **Build Logs** from the View menu. A window similar to the one shown below appears:



2.7 Dependencies within a Project

You can review the file dependencies within your project at any time using the following options on the Project menu:

Scan Scans the file dependencies and holds them internally. This is done automatically when you build your project but you may wish to select this option if you are unsure about the dependency status.

Show Displays the file dependency hierarchy. This is a useful way of displaying an overview of your whole project.

2.8 Entering the Debugger from the Project Manager

When you have successfully built and linked your project, there are two ways to enter the Debugger via the Project Manager, using commands on the Project menu:

Debug Loads the image and invokes the Debugger but then waits for you to execute the image.

Execute Invokes the Debugger and runs your image.

Notes: *You can only run your ARM image using the Debugger on an ARM emulator, ARM6 PIE card or EmbeddedICE.*
You can only have one instance of the ARM Debugger running at any time. If you try to start a second, a message appears informing you that you cannot perform the operation.

Project Manager

3 Debugger

This chapter describes how to use the Debugger.

3.1	Overview	3-2
3.2	Accessing the Debugger	3-3
3.3	Loading and Displaying the Image	3-3
3.4	Executing the Image	3-4
3.5	Breakpoints and Watchpoints	3-6
3.6	Viewing Variables	3-9
3.7	Setting Expressions	3-10
3.8	Viewing Registers	3-10
3.9	Viewing a Backtrace	3-11

Debugger

3.1 Overview

The Windows Debugger can be used to debug programs built using the ARM Project Manager (as a debug version) or any programs built with the command-line tools. You can also use the Debugger to benchmark code or debug code on real hardware, such as an ARM6 PIE card or EmbeddedICE, using remote debug. For more information about remote debugging, see *Chapter 4, Remote Debugging*.

Throughout this chapter, the choice of various options is described in terms of Pulldown menus or Context menu commands. Note that many of these commands are also available from the Toolbar and the Function keys. These shortcuts are shown beside the options in *Appendix B, Debugger Options*.

3.1.1 Operations

You can perform a variety of operations in the Debugger. You can:

- load and display an image (see *3.3 Loading and Displaying the Image* on page 3-3)
- execute the image (see *3.4 Executing the Image* on page 3-4)
- step through your program line by line (see *3.4.1 Stepping through the program* on page 3-5)
- set or view breakpoints and watchpoints (see *3.5 Breakpoints and Watchpoints* on page 3-6)
- continue execution of your program to the next breakpoint or watchpoint or to program termination by choosing **Go** from the Execute menu
- reload your image and re-execute the program by choosing **Reload Image** from the File menu
- view the values of the local and global variables and specified expressions (see *3.6 Viewing Variables* on page 3-9 and *3.7 Setting Expressions* on page 3-10)
- view the registers (see *3.8 Viewing Registers* on page 3-10)
- view a backtrace (see *3.9 Viewing a Backtrace* on page 3-11)

The most common debugger operations are described in this chapter. You can perform many other operations. See *Appendix B, Debugger Options* for a full description.

3.2 Accessing the Debugger

You can access the Debugger:

- by clicking on the ARM Debugger for Windows icon in the ARM Toolkit window from the Program Manager window
- via the Project Manager by choosing either **Debug** or **Execute** from the Project menu (see *2.8 Entering the Debugger from the Project Manager* on page 2-15)

Entry to the Debugger

When you first enter the Debugger, the following windows are displayed:

ARM Debugger	The parent window for all other debugger windows.
Execution Window	This displays the currently executing program. The current portion of the program is displayed as: • machine instructions (disassembly) • source code • interleaved source code and disassembly Several machine level instructions are displayed for each source statement.
Console Window	This allows interaction between yourself and the executing program. Anything printed by the program is displayed in this window and any input required by the program must be entered here. Initially, the console window shows the startup messages of your target processor, eg. the ARMulator, PIE card or EmbeddedICE.
RDI LOG	This displays the low-level communication messages between the Debugger and the target processor.

3.3 Loading and Displaying the Image

To load your image into the Debugger:

1. Choose **Load Image...** from the File menu.
   The Open File dialog box appears.
2. Specify the filename.
   You can use the **Browse** option on the Open File dialog box to select your file. The file does not have a file extension, so you can list all of the files with extension '* **.** '.
3. Enter any command-line arguments expected by your program in the **Arguments** field and click on **OK**.

Alternatively, if you have recently loaded your required image, your file will appear as a recently used file on the File menu. You can select the listed file to load your image. If you load your image in this way, the Debugger automatically loads the image using the command-line arguments that you used previously.

When the image is loaded, the current execution marker is located at the entry point of the program.

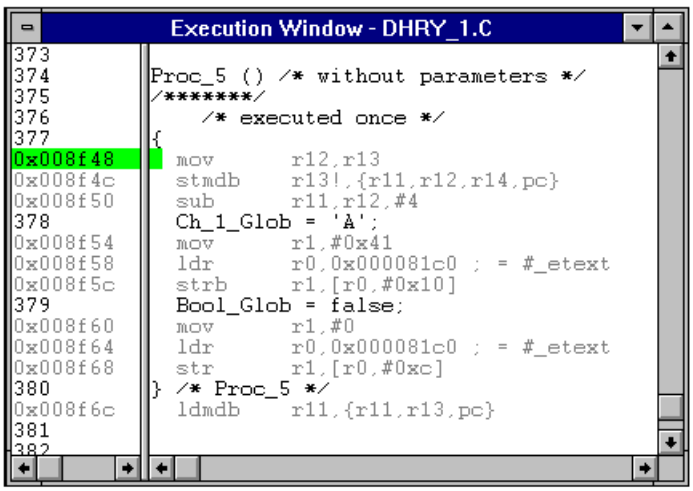
Initially the program is displayed as disassembly only, and a breakpoint is automatically set at the entry point of the image. This is usually at the `main()` function. After selecting **Go**, the program halts at this breakpoint and the program is displayed as source only.

3.4 Executing the Image

To execute your image, choose **Go** from the Execute menu or the Context menu. Your program starts execution and halts when it reaches the first breakpoint.

When the program starts executing, the Console window changes to an Active Console Window and the Status bar indicates that the program is executing. The Active Console window displays any messages printed by the program and prompts for any input required by the program. You must enter any required input into this window.

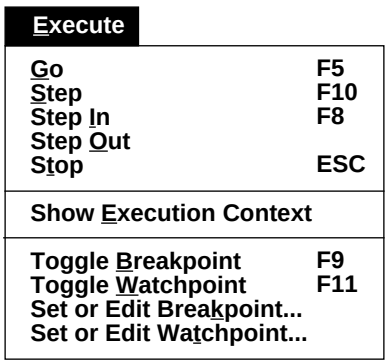
Initially, only your source code is displayed. To display the source code interleaved with the disassembly, choose **Toggle Interleaving** on the Options menu or the Context menu. This command toggles between displaying source only and displaying source interleaved with disassembly. When your source is interleaved with disassembly, the machine instructions appear in a lighter grey. An example is shown below.



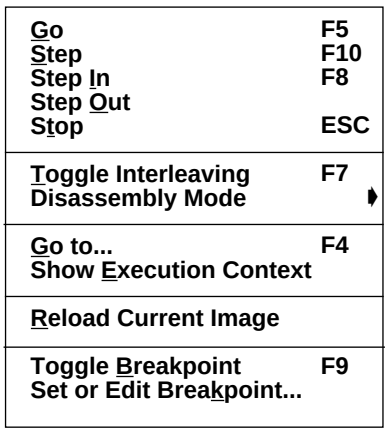
3.4.1 Stepping through the program

You can step through your program using the following commands on the Execute menu. All of the stepping commands are also available on the Context menu.

Execute Menu



Context Menu



Go Starts or continues execution of the program. The program halts at the next breakpoint or watchpoint.

Step Moves to the next line of the program. If only the source is displayed, **Step** moves to the next line of source. If disassembly is interleaved with source, **Step** moves to the next machine instruction in the disassembly.

Step In Steps through a program following all of the function calls.

Step Out Steps from the current function to where it was called from, immediately after the function call.

Debugger

3.5 Breakpoints and Watchpoints

3.5.1 Setting breakpoints

A breakpoint halts the program at a specified location. To set a breakpoint:

1. Move to the location in the program where you wish to set the breakpoint and click at that position.
2. Choose **Toggle Breakpoint** from the Execute menu or the Context menu.

If you wish to set a breakpoint at a particular function:

1. Choose **Function Names** from the View menu.

   A list of all the functions used within the program is displayed.
2. Double-click on the function where you wish to set a breakpoint. Alternatively, you can choose **Source Disassembly** from the Context menu.

   A new source window is displayed containing the function source.
3. Click at the position where you wish to set the breakpoint and choose **Toggle Breakpoint** from the Execute menu or the Context menu.

When you have created a breakpoint, it appears as a red marker on the left pane of the Execute window. On the right pane, a red marker appears somewhere on the line. If the line of code is multi-statement, the position of the marker will be determined by the statement at which the breakpoint is set. If you have set the breakpoint on the wrong statement simply choose **Toggle Breakpoint** again to remove it and recreate it at the correct position.

The program will halt execution when it reaches the breakpoint. If you continue execution and the program fails to stop at the breakpoint, this indicates that the program never reached that part of the code. You may need to set a breakpoint in a slightly different place.

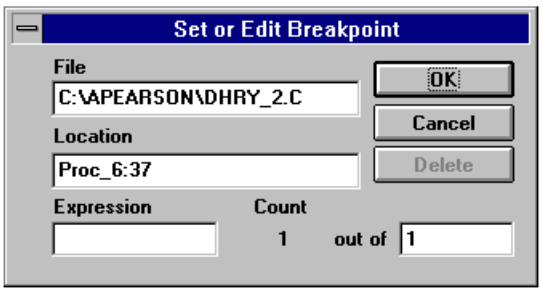
Note: *If you reload the same image, the breakpoints are retained. If you load a different image and then load your original image again, the breakpoints are cancelled and you will have to recreate them.*

Complex breakpoints

You can also set a breakpoint which will come into force after the program has passed the specified point a set number of times. These are called complex breakpoints. To set a complex breakpoint:

1. Move to the point in the program where you wish to set the breakpoint. You can do this by listing the functions and moving to the appropriate function as described for ordinary breakpoints.
2. Click at the position where you wish to set the complex breakpoint and choose **Set or Edit Breakpoint** from the Execute menu or the Context menu.

   The Set or Edit Breakpoint dialog box appears.



3 The File and Location fields are already completed. Set the count to your required value. For example if you wish to halt the program the second time it reaches this point, set the count to 2.

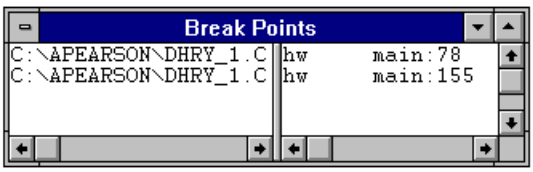
4 You can also set an expression. The program only halts when this expression is true. For example, if you specify an expression and set the count to 2, the program will halt the second time the program reaches the specified location and the expression is true.

3.5.2 Viewing breakpoints

You can view all of the breakpoints as follows:

1 Choose **Breakpoints** from the View menu.

A list of breakpoints is displayed showing the filename and the location of the breakpoints within that file. An example is shown below.



You can also edit a breakpoint by double-clicking on the breakpoint location, for example `hw main:155`. The Set or Edit Breakpoint dialog box appears (as shown above) and you can edit the breakpoint as described above.

3.5.3 Setting watchpoints

A watchpoint halts the program when a specified register or variable changes. To set a watchpoint when, for example, a specified local variable changes:

1 Choose **Variables ▸ Local** from the View menu.

A list of local variables is displayed showing the variable names and their current values.

2 Highlight the variable value on which you wish to set a watchpoint and choose **Toggle Watchpoint** from the Execute menu or the Context menu.

During execution, the program will now halt when that variables changes.

To delete a watchpoint, select the watchpoint and choose **Toggle Watchpoint** from the Execute menu or the Context menu.

Note: *If you set a watchpoint on a local variable, the watchpoints are lost as soon as you leave the function which uses the local variable.*

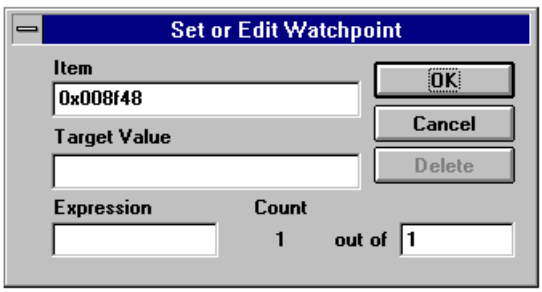
Complex watchpoints

You can also set a watchpoint which will act when a specified variable or expression reaches a given value. This is called a complex watchpoint. To set a complex watchpoint:

1 View the variable or expression on which you wish to set the watchpoint. You can do this by choosing **Variables ▸ Local** or **Variables ▸ Global** from the View menu.

2 Highlight the variable or expression on which you wish to set the complex watchpoint and choose **Set or Edit Watchpoint** from the Execute menu or the Context menu.

The Set or Edit Watchpoint dialog box appears.



3 The **Item** field is already completed. If you specify:

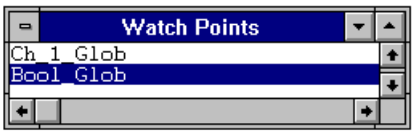
a target value	The program halts when the specified variable reaches the target value.
an expression	The program halts when the specified variable reaches the target value and the expression is true.
a count	The program halts when the variable changes (to the target value if you specify one) for the *n*th time.

3.5.4 Viewing watchpoints

You can view all of the watchpoints as follows:

1 Choose **Watchpoints** from the View menu.

A list of watchpoints is displayed.



Double-clicking on an entry in the list allows you to modify the watchpoint.

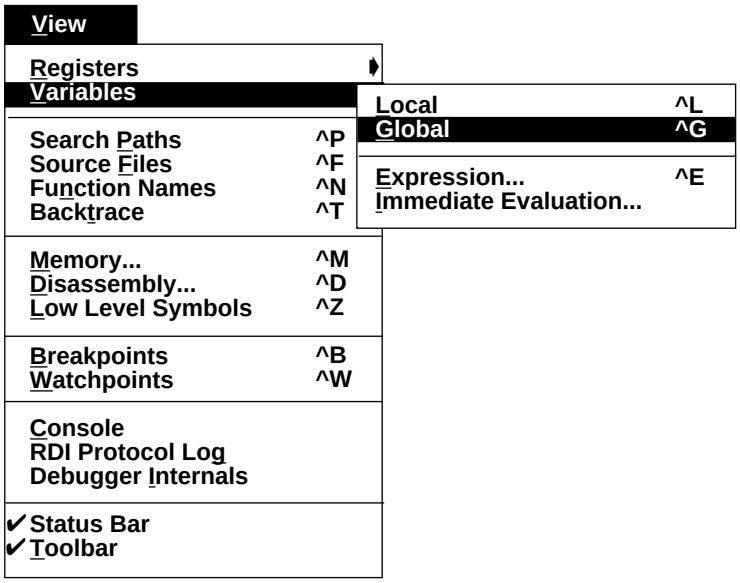
3.6 Viewing Variables

You can view local and global variables in two ways:

- display a complete list of local or global variables
- view a specific variable and its contents

Displaying a list of variables

To display a complete list of local or global variables, choose **Variables ▸ Local** or **Variables ▸ Global** from the View menu.



A list of local or global variable names and their contents is displayed. The local variables list shows the local variables used at the current point in your program.

You can double-click on the value of a variable to modify its contents. Double-clicking on a variable name displays its type.

Viewing a specific variable

To view the contents of a specific variable, highlight the variable in the source code and choose **Variables ⏵ Immediate Evaluation** from the View Menu. The specified variable and its contents at the current point in the program are displayed.

You can view a variable in this way if it is a global variable or a local variable in the current context. For other local variables, you need to open a backtrace window and highlight the line you wish to view. For more information on backtraces, see *3.9 Viewing a Backtrace* on page 3-11.

3.7 Setting Expressions

You can set expressions in two ways:

- Highlight an expression in the code and then choose **Variables ⏵ Expression** from the View menu. The selected expression appears in the Expression dialog box. Click on **OK** and the expression and its evaluation appear in the Expression window.
- To create a new expression, choose **Expression** from the View menu. The Expression dialog box appears. Enter your required expression and click on **OK**. The expression and its evaluation appear in the Expression window.

Note: *Ensure that any variable you use in the expression are in the current context, ie. the a global variables or local variables in the current local variables set.*

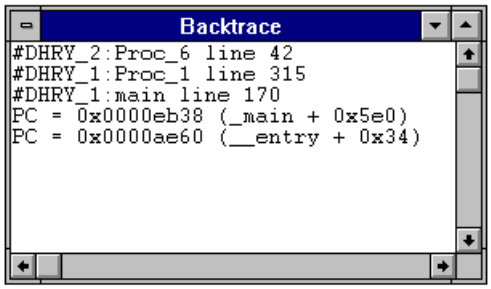
3.8 Viewing Registers

You can view a breakdown of registers in any mode by choosing **Registers** from the View menu. This allows you to examine the contents of each register at the current location in your program.

In User mode, all of the registers are displayed. In any other mode, only the banked registers are displayed.

3.9 Viewing a Backtrace

When the program has halted at a breakpoint or watchpoint, choosing **Backtrace** from the View menu will show you information about all of the currently active procedures, starting with the most recent. An example of backtrace information is shown below:



The first line of the display indicates the function you are currently in. The following lines indicate the line where each function was called. You can highlight a location pointed to in the backtrace and set breakpoints or show variables or source/disassembly using the Context menu.

Double-clicking on a line in the Backtrace window displays the disassembly or source at a given location.

Debugger

4 Remote Debugging

This chapter describes how to perform remote debugging.

4.1	Overview	4-2
4.2	ARM6 PIE Card	4-3
4.3	ARM7 PIE Card	4-6
4.4	EmbeddedICE	4-8

Remote Debugging

4.1 Overview

The ARM Software Development Toolkit supports execution of ARM and Thumb instructions on the cross-development host machine using the software emulation of an ARM processor called ARMulator or THUMBulator. These can also be reconfigured to simulate different hardware environments. The use of ARMulator allows software development and benchmarking without the need for real hardware to be available.

Advanced RISC Machines Ltd. has produced a number of Platform Independent Evaluation (PIE) cards. These cards contain real ARM hardware and connect to a host via a serial or JTAG port.

The ARM Software Development Toolkit can download code to the cards and execute the code using real hardware instead of the ARMulator. The main advantages are:

- execution speed is much faster
- real-time benchmarking can be performed

The PIE boards have a EPROM on board which contains debug monitor software called Demon. Demon communicates to the host via the serial port.

The ARM Software Development Toolkit can also communicate to any ARM-based target system, as long as either Demon is running on the target board or the target board has an ARM processor with debug support. This allows you to use real hardware to debug application software under the control of the ARM Software Development Toolkit.

The following sections describe three cases:

- ARM6 PIE card
- ARM7 PIE card
- EmbeddedICE card

4.2 ARM6 PIE Card

The ARM6 PIE card is a Platform Independent Evaluation (PIE) card based around the ARM60 processor. The ARM6 PIE card includes the following:

- ARM60 processor
- 512 Kbytes of SRAM, for data and program storage (option of 2Mb RAM)
- 128 Kbyte EPROM, containing Demon
- 20 MHz clock supply (scalable for reduced power)
- serial RS232 host interface
- interface for logic analyser / system expansion
- JTAG boundary scan test port
- user configurable big-endian operation

4.2.1 Equipment required for ARM6 PIE card remote debug

The following equipment is required for remote debugging with the ARM6 PIE card:

- host PC running ARM Windows Toolkit
- RS232 serial cable
- +5 volts DC ± 10%, 250 mA or greater PSU

You can also plug the card into a PC ISA slot (8-bit or 16-bit) to receive power. If you do this you will still need to connect the serial cable.

4.2.2 Connecting up

Figure 4-1: Connecting up the ARM6 PIE card shows how to connect up the ARM6 PIE card.

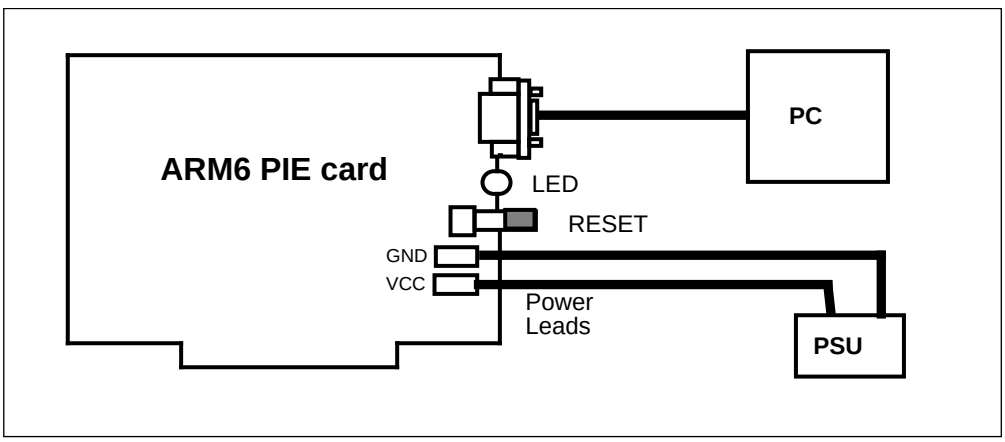


Figure 4-1: Connecting up the ARM6 PIE card

To connect the ARM6 PIE card:

1. Power the card using the two spade connectors adjacent to the red reset button.

   The spade connectors are marked with their polarity. The connector directly below the reset button is ground, with +5 volts as the connector at the bottom of the card.

2. Once the card is powered up, activate the self-test by pressing the reset button.

   The red LED should light up for approximately one second, go out for one second and then relight and stay on. If the LED fails to light or fails to stay on, the board is either faulty or incorrectly powered.

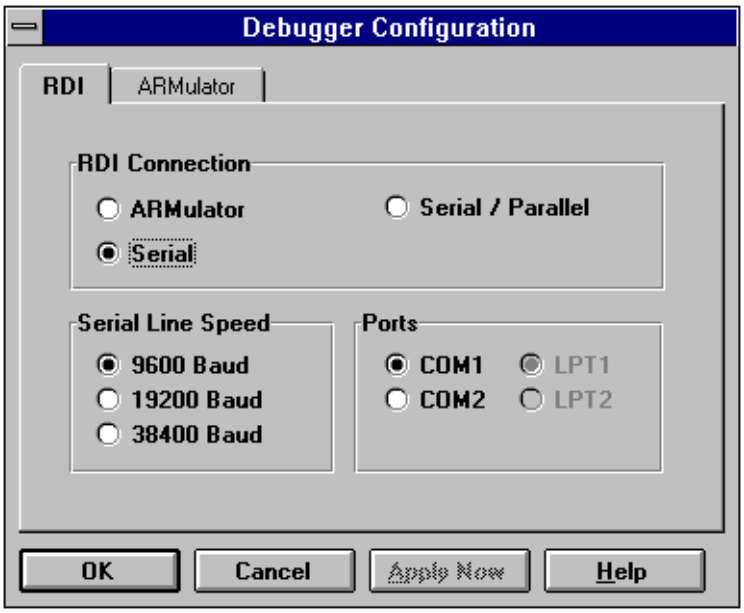
3. Connect the serial cable to the 9-pin connector on the ARM6 PIE card and the other end to the host PC.

Note: *The self-test may not function if the serial cable is connected to the ARM6 PIE card.*

4.2.3 Configuring the Debugger

To configure the Debugger:

1 Choose **Configure Debugger...** from the Options menu. The Configure Debugger property sheet appears.



2 Click on the **RDI** button to move to the Remote Debug Interface (RDI) property page.

3 Select the RDI Connection to use.

For a ARM6 PIE card or any card using Demon this will be serial.

4 Select the serial line speed and communications port.

5 Click on **OK**.

The Debugger is restarted and data is written to the ARM6 PIE card. The version number of the Demon monitor will be displayed in the Console window.

From now on you can download your program to the ARM6 PIE card by choosing **Reload Current Image** from the File menu. When the code is reloaded, a byte count is displayed as data is written to the ARM6 PIE card.

The Debugger now performs identically to the ARMulator, but programs are run on the ARM6 PIE card. This allows full source code debugging, with the use of single stepping and breakpoints etc.

4.3 ARM7 PIE Card

The ARM7 PIE card is a Platform Independent Evaluation (PIE) card based around an ARM7D family processor.

The ARM7 PIE card includes the following:

- ARM70 processor
- 512 Kbytes of SRAM, for data and program storage
- 128 Kbyte EPROM, containing Demon
- 40 MHz clock supply (scalable for reduced power)
- serial RS232 and parallel printer port host interface
- JTAG boundary scan and/or EmbeddedICE debug port
- interface for logic analyser/system expansion

You can connect an ARM7 PIE card in two different configurations:

- using a serial connection like the ARM6 PIE card
- using an EmbeddedICE board via the JTAG connector on the ARM7 PIE card

Note: *The ARM7 PIE card has a parallel connector but currently the Demon software does not support connection via this port. Do not connect this port to the host PC.*

4.3.1 Equipment required for ARM7 PIE card remote debug

The following equipment is required for remote debugging with the ARM7 PIE card:

- host PC running ARM Windows Toolkit
- RS232 Serial cable (Null Modem)
- +7 to +9 DC (unregulated), 500mA or greater PSU

4.3.2 Connecting up for serial debug

Figure 4-2: Connecting up the ARM7 PIE card shows how to connect up the ARM7 PIE card.

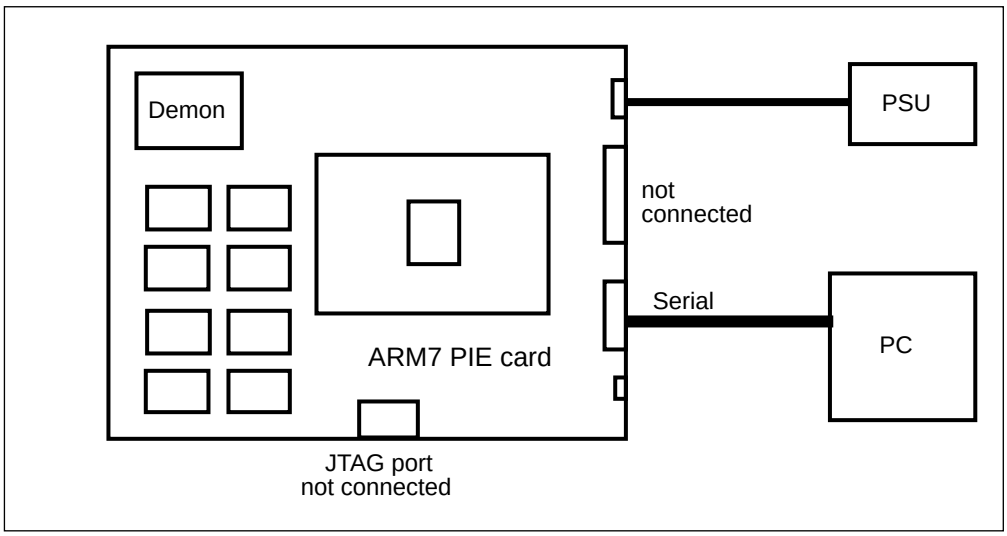


Figure 4-2: Connecting up the ARM7 PIE card

To connect the ARM7 PIE card for serial debugging:

1 Connect a PSU unit to the 2.1 mm power connector.
The positive +7 to +9V is connected to the centre connector.

2 Activate the self-test by pushing the red reset button.
The red LED should light for one to four seconds (this varies with the amount of SRAM on the card), go out for one second and then relight and stay on.
The self-test may not function if the serial cable is connected to the ARM7 PIE card.

3 Connect a serial cable between the 9-pin RS232 connector on the ARM7 PIE card and the host PC.

4.3.3 Configuring the Debugger for serial debugging

To confiugure the Debugger for serial debug, follow the procedure for the ARM6 PIE card given in *4.2.3 Configuring the Debugger* on page 4-5.

4.4 EmbeddedICE

The EmbeddedICE unit is used to convert RS232 serial data into JTAG data that can be sent to an ARM core with a debug support. The board can use a parallel connection in conjunction with the serial connection for faster downloading. In particular, there is no need to port the Demon code to the target system, as no support software is required.

The use of EmbeddedICE allows any ARM core with debug support to be debugged using the ARM Software Development Toolkit. No additional resources are required on the target board. Once EmbeddedICE is configured, all the functionality of ARMulator is now available on real hardware using the TAP controller.

4.4.1 Equipment required for EmbeddedICE remote debug

The following equipment is required for remote debugging with EmbeddedICE:

- EmbeddedICE v1.01 or later board
- +7 to +9 DC (unregulated), 500mA or greater PSU for EmbeddedICE and ARM7 PIE card
- parallel port cable (optional)
- JTAG ribbon cable
- ARM7 PIE or any target ARM processor with debug support
- RS232 serial cable (Null Modem)

4.4.2 Connecting up EmbeddedICE

Figure 4-3: Connecting up EmbeddedICE shows how to connect up EmbeddedICE.

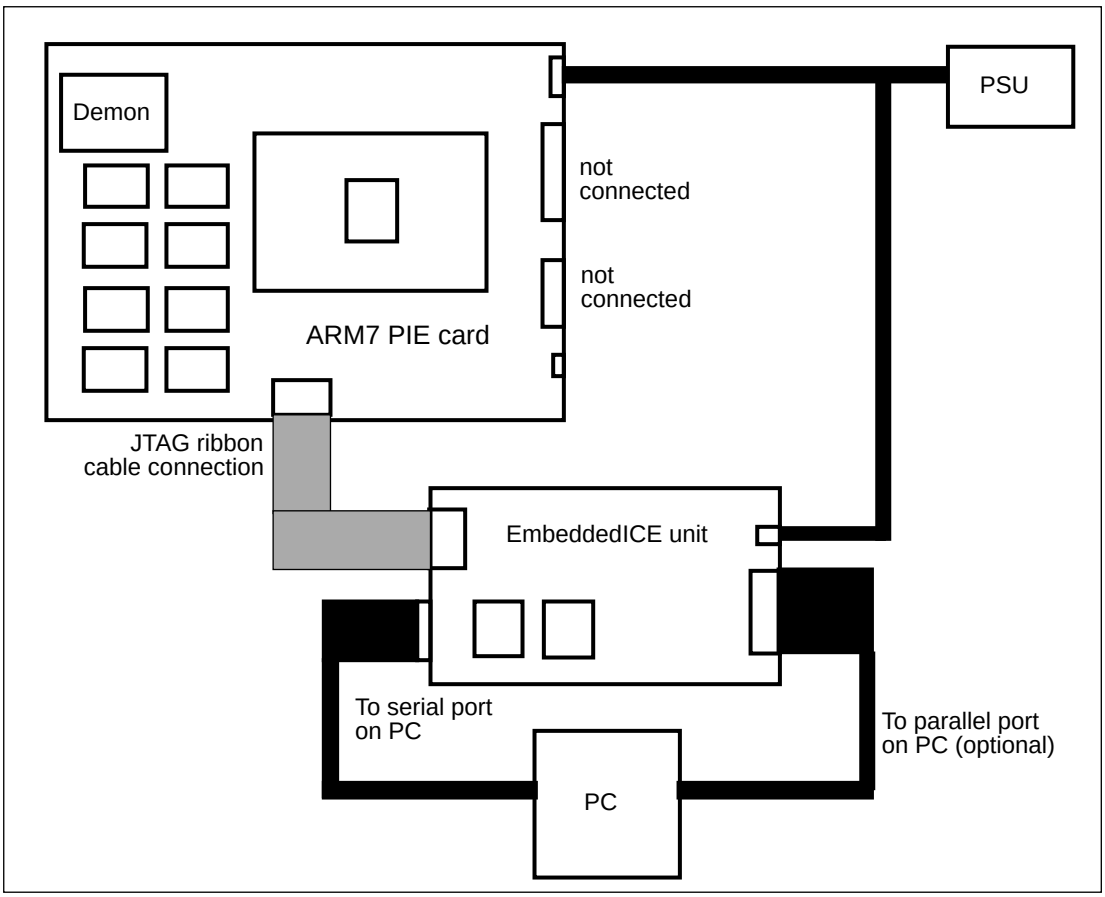


Figure 4-3: Connecting up EmbeddedICE

1 Remove the Demon EPROM from the ARM7 PIE board.

This is located in the top-left corner, marked U16 on the PCB.

Demon is only used for serial debug and can interfere with EmbeddedICE operation (interrupts are generated). The self-test will no longer function.

2 Connect a cable between the 9-pin serial port on EmbeddedICE and the comms port on the host PC.

A parallel port cable can optionally be connected between the parallel port connector on EmbeddedICE and the printer port of the host PC.

The serial port is used for bi-directional transfers but the parallel port is only used to download code to the ARM7 PIE card via EmbeddedICE.

The parallel port is significantly quicker than the serial port.

3 Power both the EmbeddedICE and ARM7 PIE cards via the 2.1mm power connector.

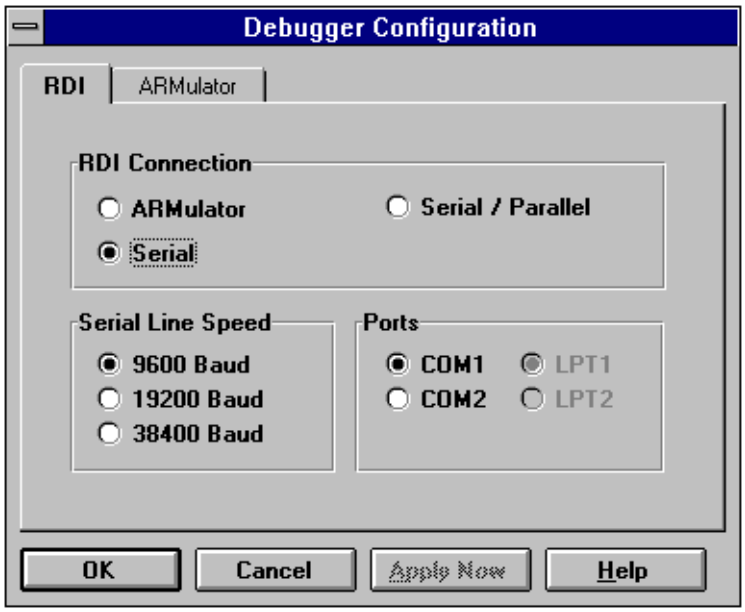
4 Attach the JTAG ribbon cable between the EmbeddedICE unit and ARM7 PIE card.

4.4.3 Configuring the Debugger for EmbeddedICE

To configure the Debugger for EmbeddedICE:

1 Choose **Configure Debugger...** from the Options menu.

The Configure Debugger property sheet appears.



2 Select the Remote Debug Interface (RDI) connection property page.

3 Select the RDI Connection to use.

For a ARM7 PIE card using JTAG or any other card using JTAG this will be serial, or serial and parallel.

EmbeddedICE can be connected to the host via just a serial connection or both serial and parallel for faster downloads. The serial line speed and comms port settings allow you to specify which ports the EmbeddedICE card is connected to and what serial baud rate to use.

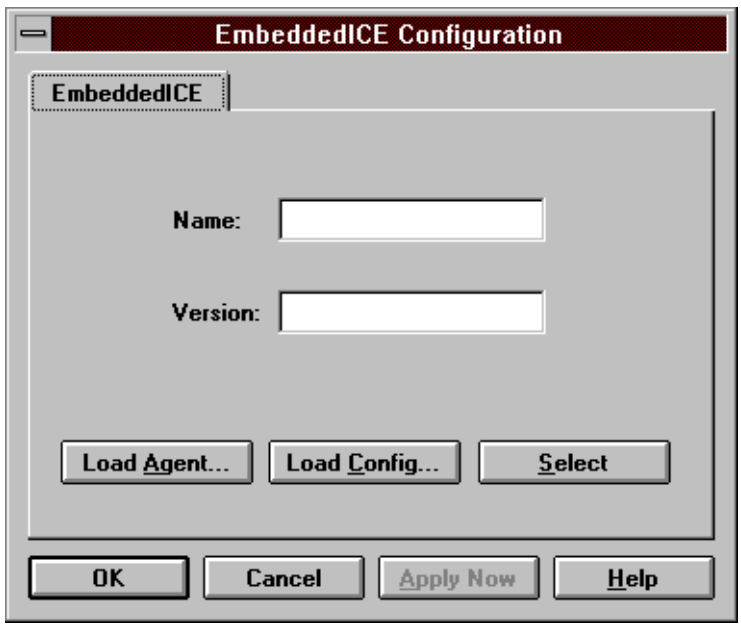
4 Click on **OK**.

The Debugger is restarted and data is written to the EmbeddedICE card. The version number of the EmbeddedICE monitor will be displayed in the Console window.

From now on you can download your program to the ARM7 PIE card by choosing **Reload Current Image** from the File menu. When the code is reloaded a byte count is displayed as data is written to the ARM7 PIE card. The Debugger now functions identically to the ARMulator, but programs are run on the ARM7 PIE card.

4.4.4 Advanced EmbeddedICE configuration

This configuration is not generally required. However, to perform advanced debugging, choose **Configure EmbeddedICE** from the Options menu.



Notes: It is possible to download an agent to EmbeddedICE which is a replacement for the EmbeddedICE ROM. The agent is then started in RAM. Click on the **Load Agent** button to select an agent to download.

It is also possible to download a different configuration file. A different configuration file is required for each ARM processor. The EmbeddedICE v1.02 ROM has configurations for the ARM7DI, ARM70DI, ARM7DMI and ARM7TDMI so new configuration files are not required for these parts. New configuration files will be required for other parts. The Name and Version fields are used with the **Select** button to select a different configuration after it has been downloaded to the EmbeddedICE card.

Remote Debugging

By default EmbeddedICE is set-up with

```
semihosting_enabled=1
```

and

```
semihosting_vector=8
```

To change this, choose **Debugger Internals** from the View menu and edit the variables.

For more details on semihosting under EmbeddedICE, see the ARM Software Development Toolkit Reference Manual (ARM DUI0020).

5 Worked Example

This chapter guides you through a worked example to create, build and benchmark a project using the Project Manager and Debugger.

5.1 Example 1: Dhrystone 2.1 Benchmark 5-2

5.2 Example 2: Software Development Example 5-10

Worked Example

5.1 Example 1: Dhrystone 2.1 Benchmark

5.1.1 Overview

This example works through benchmarking the Dhrystone 2.1 source using the Windows Toolkit.

The example is divided into the following stages:

- creating a project
- building the project
- changing the project options
- rebuilding the project with the new options
- obtaining the code size
- debugging the project using the ARM Debugger and ARMulator
- using clock to benchmark one iteration of Dhrystone
- debugging the project using the ARM Debugger and ARM6 PIE card
- conclusions

Note: *This example is concerned only with execution speed and takes no account of code size.*

Equipment required

The following equipment is required to work through this example:

- IBM PC running **Windows 3.1x**, **WindowsNT 3.5 or later** or **Windows95**
- installed Windows Toolkit
- installed Dhrystone 2.1 C source code (included with the ARM Toolkit)
- ARM6 PIE card (optional for debugging using the ARM6 PIE card)
- serial cable (optional for debugging using the ARM6 PIE card)
- PSU for ARM6 PIE card (optional for debugging using the ARM6 PIE card)

The Windows Toolkit should already be installed on your machine. For more information on installation, see the Release Notes provided with the ARM Software Development Toolkit.

Note: *For* **Windows 3.1x***, the Microsoft 32bit extension* **Win32s v1.20 or later** *must already be installed.*

5.1.2 Creating a project

The first stage of this worked example is to create your project. To do this:

1 Move to the directory containing your Dhrystone source files. If you have used the default locations during installation, this will be `c:\arm200\examples\dhry`.

2 Start the ARM Project Manager by double-clicking on the ARM Project Manager icon.

3 Create a Dhrystone memory map file called `armsd.map`. To do this:

a) Choose **New** from the File menu.

An empty text window called Untitled is opened.

b) Type the following in the window:

```
00000000 80000000 RAM 4 rw 135/85 135/85
```

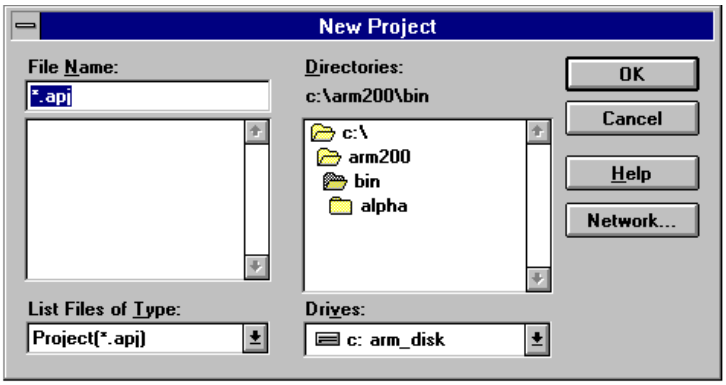
c) Choose **Save As...** from the File menu and enter the filename `armsd.map` in the dialog box that appears. You should save the file into the same directory as the source files `dhry_1.c` and `dhry_2.c`.

The format of the memory map is documented in the ARM Software Development Toolkit Reference Manual (ARM DUI 0020). The map file describes a:

- single contiguous section of RAM from 0 to 7FFFFFFF
- 32-bit databus
- read/write access
- non sequential and sequential access times of 135 and 85 nanoseconds respectively

This is typical of a 20 MHz ARM PIE card. You can create different areas of memory and use the linker to place the code and data at the correct locations.

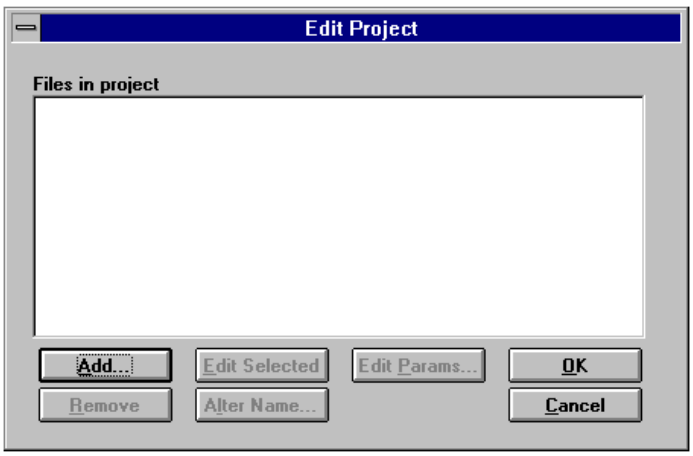
4 Choose **New** from the Project menu. The New Project dialog box appears as shown below.



Note: *This dialog box will look slightly different if you are using Windows95.*

5 Move to the `dhry` directory and select it.

6 Enter the name of your new project, `dhry1.apj` in the **Filename** field and click on **OK**. The Edit project dialog box appears as shown below.



7 Click on **Add...** and highlight the Dhrystone source files `dhry_1.c` and `dhry_2.c` and the map file `armsd.map`.

8 Click on the Project dialog box **OK** button and then click on the Edit Project dialog box **OK** button.

You have created a new project called `dhry1.apj` and added the C source files `dhry_1.c` and `dhry_2.c` and the map file `armsd.map` to the project.

If you did not type the extension `.apj`, this is added automatically.

9 You can see which files make up the `dhry1` project by choosing **Show Dependencies** from the Project menu.

5.1.3 Building the project

The next stage of this example involves building your project. To do this:

1 Choose **Rebuild All** from the project menu. This compiles and links the files `dhry_1.c` and `dhry_2.c`.

Any errors and warnings are output to the `dhry1` window. Many warnings are likely to be displayed, since Dhrystone does not declare its functions.

2 If any errors are displayed, double-click on the error message to open the source file. You are taken to the line number with the error.

3 Edit your source files to correct the errors and choose **Save** from the File menu.

4 Choose **Rebuild All** to rebuild your project.

5 Continue until no errors are encountered.

5.1.4 Changing the project options

You can obtain better performance from your program by changing some of the build options. To change some of your options and therefore improve the performance:

1 Set the option to compile without any debug information. To do this:

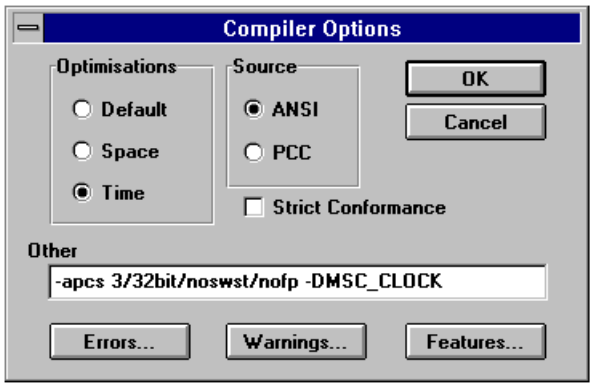
a) Choose **Project** from the Options menu.

b) Select a **Release** build rather than a **Debug** build.

c) Click on **OK**.

Having removed high-level debug information from the project, the linker automatically adds some low-level debug information. This will allow you to set breakpoints on, function names for example.

2 To get the best performance, you need to set your optimisation to time rather than code size. To do this, choose **Compile** from the Options menu and select the **Time** optimisation check box on the Compiler dialog box.



3 Click on **Warnings...** in the Compiler dialog box. The Warnings dialog box appears. Disable all the warnings and click on **OK**. This means that no warning messages will be displayed during the build process.

Leave the Compile dialog box open for the next step.

4 Change armcc so that a register is not permanently allocated as the frame pointer and disable software stack checking.

To to this, enter the following into the **Other** field in the Compile dialog box:

```
-apcs 3/32bit/noswst/nofp -DMSC_CLOCK
```

The `-DMSC_CLOCK` defines `MSC_CLOCK` as a pre-processor macro, as if by a line `#define MSC_CLOCK` in the source.

For more information on compiler options see the ARM Software Development Toolkit Reference Manual (ARM DUI 0020).

5 Click on **OK** to close the dialog box.

Generating a binary output file

To generate a binary output file and to specify separate code and data regions:

1 Choose **Linker...** from the Options menu.

2 Type the following into the Linker dialog box and then click on **OK**.

```
-Debug -AIF -BIN -Base 0x8000 -DATA 0x100000
```

This ensures that the ARM Symbolic Debugger tables are included in the output image. The `-AIF -BIN` tells the linker to generate an AIF binary file with code at address 0x8000 and data at address 0x100000.

Obtaining the code size of project modules

You may also wish to obtain the code size of the project modules. You can set this up using the ARM Object Format Decoder, DecAOF.

To change the DecAOF options:

1 Choose **DecAOF...** from the Options menu.

The DecAOF dialog box appears.

2 Enter `-q` in the Options field.

For more information about DecAOF, see the Software Development Toolkit Reference Manual (ARM DUI 0020).

5.1.5 Rebuilding the project with the new options

You now need to rebuild your project to take into account the new project options.

To rebuild the project, choose **Rebuild All** from the Project menu. You should not get any compiler warnings but the linker will display some warnings because of the command-line options used to build the project.

5.1.6 Obtaining the code size

To check the code size after building the project:

1 Choose **Show Dependencies** from the Project menu.

The Project Dependencies window appears.

2 Highlight the first object file, `dhry_1.o` and click on the **DecAOF** button.

The following information is displayed:

```
c$$code3400
c$$data 48
c$$zidata 10200
```

This shows the code size in bytes for the object file `dhry_1.o`. The `zidata` is the zero initialised data.

3 Repeat this for the other object file `dhry_2.o`.

5.1.7 Debugging the project using the ARM Debugger and ARMulator

To debug your project using the Debugger and ARMulator, choose **Debug** from the Project menu. The ARM Debugger starts and loads the `dhry1` image. By default the ARM Debugger uses the ARMulator.

Checking the clock speed

Before executing the program, ensure the clock speed is set to 20MHz. To do this:

1. Choose **Configure Debugger...** from the Options menu.
   The Configure Debugger property sheet appears.
2. Click on **ARMulator** and ensure the clock speed is set to 20MHz.
3. Click on **OK** to close the dialog box.

Executing the program

Choose **Go** from the Execute menu to run Dhrystone.

Since a map file is included in the project, the simulation will run at real speed, ie. the timer function in C correctly reports the amount of time expired. This is the same as reported on real hardware. The only disadvantage is the real time to execute is about 10 times longer. If you enter 30,000 as the number of runs through the benchmark, the benchmark will run for a few minutes and then report the results. The following results have been obtained using the ARMulator:

Microseconds for one run through the benchmark:	70.7
Dhrystone per Second:	14150.9

5.1.8 Using clock to benchmark one iteration of Dhrystone

You may need to find out the amount of time for a single iteration of an algorithm. To do this you should set a breakpoint which will be reached once per iteration. To set the breakpoint:

1. Choose **Reload Current Image** from the File menu.
2. Choose **Low Level Symbols** from the View menu. The Low Level Symbols window opens.
3. Highlight Proc5 and choose **Toggle Breakpoint** from the Execute menu. A breakpoint is set on the entry to Proc5. In the Dhrystone source, Proc5 is called at the beginning of the main loop.
4. Choose **Go** from the Execute menu. This runs the code until the breakpoint is reached. You will need to enter the number of runs when prompted.
5. When the program halts at the first breakpoint, select **Debugger Internals** from the View menu. The Debugger Internals dialog box is displayed.
6. Record the time given in the clock variable. This is the execution time in Microseconds.

7 If you wish, you can display the memory statistics. This indicates where the external memory accesses have been occurring, and how long they took. To display the memory statistics, select **statistics_inc** by double-clicking on the three full stops.

The statistics_inc dialog box is displayed.

S cycles Sequential, ie. the CPU request transfer to or from the same, one word or half word after the preceding address.

N cycles Non sequential, ie. the CPU request transfer to or from an address which is unrelated to the address used in the preceding cycle.

I cycles Internal cycles. ie. the CPU does not require a transfer, as it is performing an internal function.

C cycles Coprocessor cycles.

F cycles Fast clock cycles for cached processors (FCLK).

8 Select **Go** from the Execute menu. The program continues to run until the breakpoint at Proc5 is reached for a second time, ie. one iteration of Dhrystone.

9 Record the time in the clock variable again.

The difference between the two values gives you the number of Microseconds per iteration.

The following values have been obtained for `clock`:

clock at first hit of breakpoint on Proc_5:8207454 µs

clock at second hit of breakpoint on Proc_5: 8207524 µs

8207524 - 8207454 = µs for one iteration of Dhrystone

= 70 µs for one iteration of Dhrystone

Dhrystones per second = 1/time for one Dhrystone iteration

= 1/70 µs

= 14,286 Dhrystones per second

5.1.9 Debugging the project using the ARM Debugger and ARM6 PIE card.

To debug the project using the Debugger and ARM6 PIE card:

1 Connect the ARM6 PIE card to the PC using a serial cable and power the card.

2 Run the card self-test program by pressing the reset button. The red LED lights for approximately one second, goes off for one second and then relights and stays on. For more information on connecting the ARM6 PIE card refer to the PIE User Guide.

3 In the Debugger choose **Configure Debugger...** from the Options menu.

The Configure Debugger dialog box appears. The RDI (Remote Debug Interface) connection will be set to ARMulator. Choose **Serial** and specify the COM port and serial line speed. Initially, set the serial line speed to 9600 baud. You can increase this later if your PC supports higher data rates.

4 Click on **OK**.

The Debugger will now communicate with the ARM6 PIE card. In a few seconds the ARM60 Demon string will appear in the console window. If this does not happen, check your serial cables.

You are now ready to download the Dhrystone program to the ARM6 PIE card.

To do this:

1 Choose **Reload Current Image** from the File menu. Reloading the image takes about 30 seconds with a 9600 baud rate. The download will be significantly quicker with a faster serial line speed. (Six seconds with 38400 baud rate)

2 Choose **Go** from the Execute menu. This runs Dhrystone on the ARM6 PIE card. The results are displayed in the Console Window. The Dhrystone was configured to run with 30,000 runs. For the 20 MHz ARM6 PIE card, the following results have been obtained:

Microseconds for one run through Dhrystone	70.7
Dhrystones per second:	14150.9

5.1.10 Conclusions

The same results are obtained from the three methods used to calculate the time to execute one iteration of Dhrystone. This proves the simulations are correct.

The ARM6 PIE card is not the best vehicle on which to benchmark code since its performance is poor because the interrupt routines run out of 8-bit ROM, and the SRAM has overheads of between one and two wait states. Without the wait states you would see the following performance on a 20MHz ARM60:

Microseconds for on run through Dhrystone:	32.1
Dhrystone per second:	31128.4

Worked Example

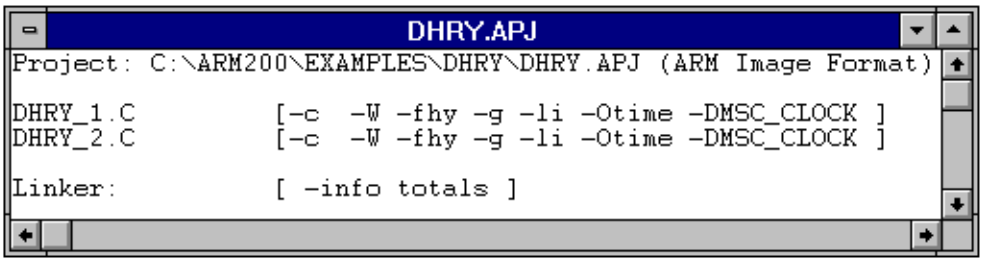
5.2 Example 2: Software Development Example

This example describes how to debug an application using the Windows Toolkit. It assumes that you have a basic understanding of the Windows Toolkit and that you have followed the previous example in *5.1 Example 1: Dhrystone 2.1 Benchmark* on page 5-2.

5.2.1 Preparing the project

To work through this example, you need to introduce some errors into the `dhry` project:

1. Open the `dhry.apj` project by choosing **Open** from the Project menu.
   The Open dialog box appears.
2. Select the `dhry` project you created in example one and click on **OK**.
   The `dhry.apj` project summary is displayed.



3. Edit the `dhry_1.c` source file as follows:
   a) Double-click on the `DHRY_1.C` filename in the project summary.
   The file `DHRY_1.C` is loaded into a text editor ready for editing.
   b) Go to line 107 by choosing **Go To...** from the Edit menu.
   c) Enter 107 in the GoTo dialog box and click on **OK**.
   Line 107 should be a simple print call that starts a new line.
   d) Edit the line to remove the close bracket. Move the next line and remove the semicolon (;) at the end of the line.
   e) Choose **Go To...** from the Edit menu again, enter 145 and click on **OK**.
   Change the line by changing <= to <:

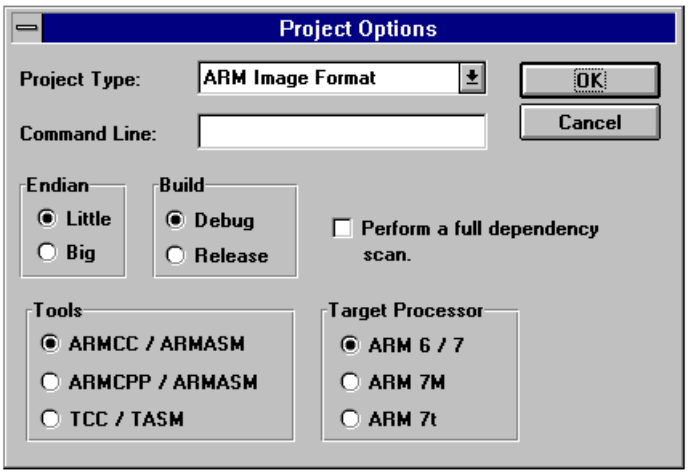
   ```
   for (Run_Index = 1; Run_Index <= Number_Of_Runs; ++Run_Index)
   ```
   becomes:
   ```
   for (Run_Index = 1; Run_Index < Number_Of_Runs; ++Run_Index)
   ```
4. Save and Close the File by choosing **Save** and then **Close** from the File menu.

5.2.2 Rebuilding the project

The next step is to change the project options so that a debug build is performed and to rebuild the project.

1 Choose **Project...** from the Options menu.

The Project Options dialog box appears.



2 Select **Debug Build** and click on **OK**.

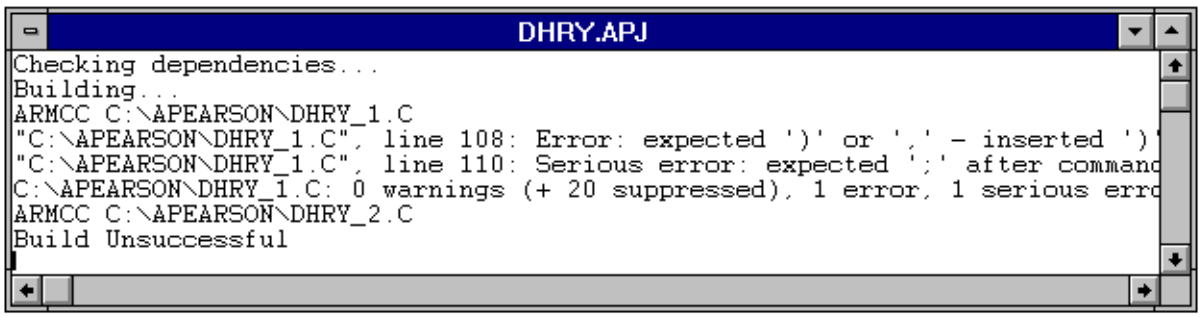
A message will be displayed asking you if you wish to clear the output window.

3 Click on **Yes**.

The project summary is redisplayed.

4 Choose **Rebuild All DHRY.APJ** from the Project menu.

The Project Manager will then compile `DHRY_1.C` and `DHRY_2.C`. The build should be unsuccessful.



5.2.3 Correcting the errors

When you have built your code, two error messages will be displayed in the project output window. The first error message in the project output window indicates a missing bracket:

```
line 108: Error: expected ')' or ',', - inserted ‘)’ before ‘;’
```

To remove the error:

1. Double-click on the message to highlight the error line in the editor.
2. Add the close bracket) by editing the line.

The second error in the project output window states:

```
line 110:Serious Error:expected ‘;’ after command-inserted before ‘printf’
```

To remove the error:

1. Choose **Next Error** from the View menu.

   The line above is not terminated with a semicolon (;) and so the compiler treats both lines as one.
2. Edit the line to replace the semicolon (;) at the end of the line.

5.2.4 Rebuilding the project

You have now corrected the two compiler errors so you can rebuild the project as follows:

1. Choose **Build DHRY.APJ** from the Project menu.

   You will be asked to save changes to `DHRY_1.C`.
2. Click on **Yes**.

   `DHRY_1.C` will then be recompiled and linked with `DHRY_2.C` and the library. The build should be successful, with no errors.

5.2.5 Debugging the project

The next step is to debug the project using the ARM Debugger. To do this:

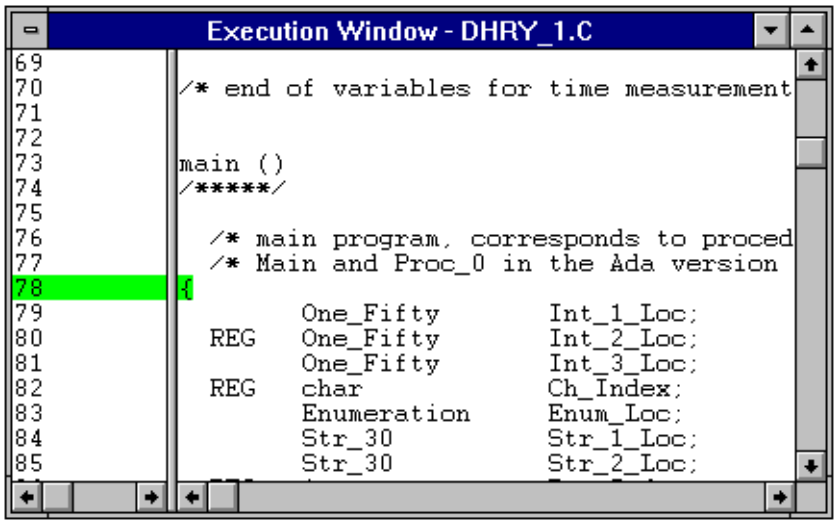
1. Choose **Debug DHRY.APJ** from the Project menu in the Project Manager.

   The ARM Debugger starts with the `dhry` project.

2. Choose **Go** from the Execute menu in the Debugger.

   This will execute the program until a breakpoint is reached.

   When you choose **Debug** from the Project Manager, a breakpoint is automatically set on the first entry after `main`. The code before `main` is created by the C compiler to initialise the C environment. In the Execution window you should see the source code to `DHRY_1.C`. A green cursor shows the current program counter position.



3. Go to line 110, by choosing **Go To...** from the Search menu.

   The Go To dialog box appears.

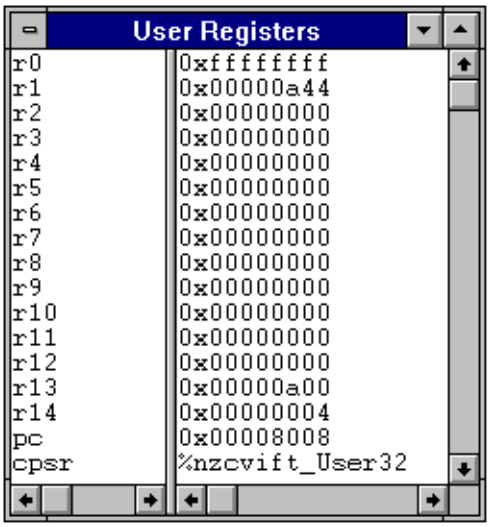
4. Enter 110 and click on **OK**. You are taken to line 110.

5. Click at the beginning of line 110 and choose **Toggle Breakpoint** from the Execute menu.

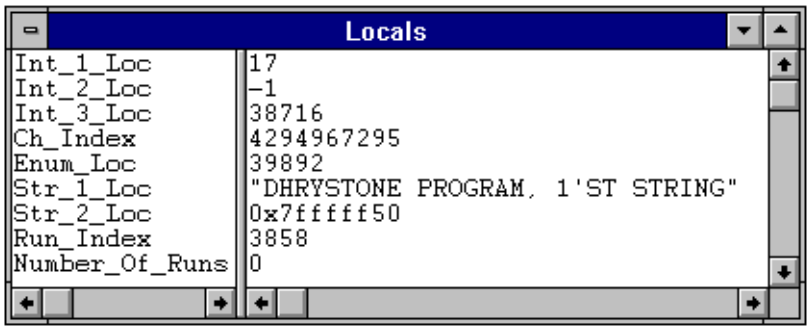
   A red cursor should appear in the line number. You have just set a breakpoint on this instruction.

6. Choose **Go** from the Execute menu to execute the code until the breakpoint is reached.

7 Choose **Register ⏵ User** from the View menu to display the ARM register.
The User Register window appears.



8 Choose **Variables ⏵ Local** from the View menu to view the local variables.



The output of the Dhrystone program appears in the Console window.

Checking execution of Dhrystone

To check that Dhrystone is executed the right number of times:

1 Set a breakpoint on line 129 by moving to the Execution window, choosing **Go To...** from the Search menu, entering 129 in the GoTo dialog box and choosing **Toggle Breakpoint** from the Execute menu.

2 Choose **Go** from the Execute menu. The Console window will prompt you for the number of runs through Dhrystone.

3 Enter 2. The code will now stop at the breakpoint at line 129. The Local Variables window will show that `Number_Of_Runs = 2`.

4 Single step the code by choosing **Step** from the Execute menu. Keep selecting **Step** until the code reaches the middle of line 146. Note that in the Local Variable window `Run_Index` now equals 1.

5 Choose **Go** from the Execute menu. The code runs for one loop of Dhrystone and then ends. This is clearly an error as we entered two as the number of runs.

Detecting the error

The next step is to detect the error:

1 Reload the project by choosing **Reload Current Image** from the File menu.

Current breakpoints will be remembered.

2 Choose **Go** from the Execute menu.

3 View all of the currrent breakpoints by choosing **Breakpoints** from the View menu. The current breakpoints will be displayed.

4 Clear the breakpoints at line 78, line 100 and line 129 by selecting each breakpoint in turn in the view and choosing **Toggle Breakpoint** from the Execute menu.

5 Select **Go** from the Execute menu.

6 Set a breakpoint on line 149.

7 Enter 3 as the number of runs in the Console window. The program will stop on the breakpoint at line 149.

8 Display the local variables by choosing **Variables ▸ Local** from the View menu and note the values of `Run_Index` and `Number_Of_Runs`.

9 Select **Go** from the Execute menu and note the value of `Run_Index` in the Local Variable window. It is updated before loop is executed.

10 Select **Go** and the code ends.

The loop at line 146 is checking for `Run_Index` to be less than `Number_Of_Runs` but `Run_Index` is updated at the start of the loop and not the end. You therefore need to change the loop so that the check is made for `Run_Index` to be less than or equal to `Number_Of_Runs`.

1 Exit the Debugger by choosing **Exit** from the File menu.

2 Switch to the Project Manager and go to line 145 in `dhry_1.c`.

3 Edit the line to add the ‘less than or equal’ check and then rebuild the project.

4 Repeat the above to verify the code does run the correct number of times.

Worked Example

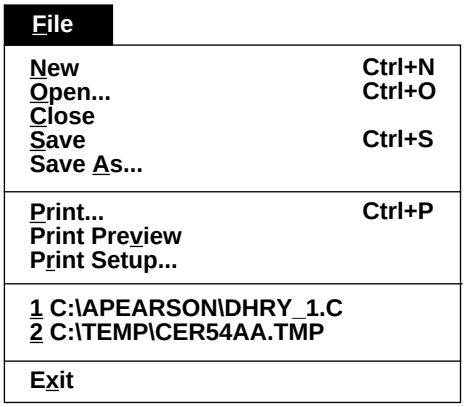
A Project Manager Options

This appendix lists and describes the options available in the Project Manager.

Many of the options listed in this appendix can also be found on the Toolbar. The relevant icons are shown to the left side of the menu name. Where function key shortcuts are also available, these are shown to the right of the menu name.

A.1	File Menu	A-2
A.2	Edit Menu	A-3
A.3	View Menu	A-4
A.4	Project Menu	A-5
A.5	Options Menu	A-7
A.6	Windows Menu	A-14
A.7	Help Menu	A-15

A.1 File Menu



New Ctrl+N Creates a new document.

Open... Ctrl+O Opens an existing document.

Close Closes an open document.

Save Ctrl+S Saves an open document using the same filename.

Save As... Displays the Save As... dialog box. Specify your filename and click on **OK**.

Print... Ctrl+P Displays the Print dialog box. Specify the document you wish to print and click on **OK**.

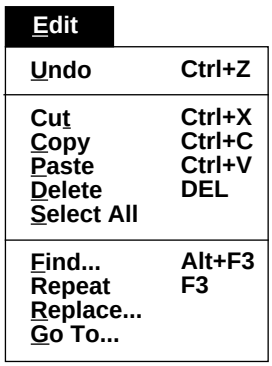
Print Preview Displays the current document on the screen as it would appear when printed.

Print Setup... Allows you to select a printer and printer connection.

Exit Exits the ARM Project Manager.

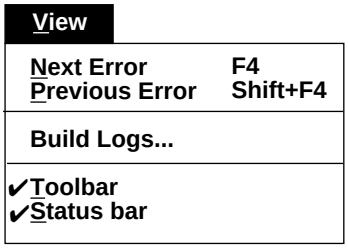
You can use the numbers and filenames near the bottom of the File menu to open the most recently used documents. Choose the number the corresponds with the document you wish to open.

A.2 Edit Menu



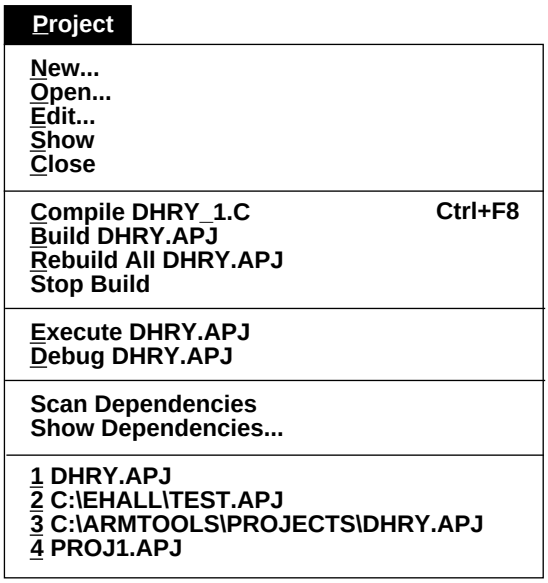
	Undo	Ctrl+Z	Undoes the previous editing command.
	**Cut**	Ctrl+X	Cuts the selected data from the document and moves it to the clipboard.
	**Copy**	Ctrl+C	Copies the selected data from the document to the clipboard.
	**Paste**	Ctrl+V	Pastes data from the clipboard into the document at the current cursor position.
	Delete	DEL	Deletes data from the document.
	Select All		Selects all the data in the document.
	Find...	Alt+F3	Displays the Find dialog box. Specify the text you wish to search for and click on **OK**.
	Repeat	F3	Repeats the previous Find operation.
	Replace...		Displays the Replace dialog box. Specify the text you wish to replace and the text you wish to use as a replacement.
	Go To....		Displays the Go To diaog box. Specify your required line in the document and click on **OK**.

A.3 View Menu



	Next Error	F4	Moves to the next error in the source file.
	Previous Error	Shift+F4	Moves to the previous error in the source file.
	**Build Logs...**		Displays the Build Logs dialog box and allows you to view the build log files.
	Toolbar		Displays or hides the toolbar.
	Status bar		Displays or hides the status bar.

A.4 Project Menu



New... Displays the New Project dialog box and allows you to create a new project.

Open... Displays the Open Project dialog box and allows you to open an existing project.

Edit.. Displays the Edit Project dialog box and allows you to edit a project as follows:

Add... Displays the Add Project dialog box and allows you to add a file to your project.

Remove Removes a selected file from your project.

Edit Selected Opens a selected file using your default editor.

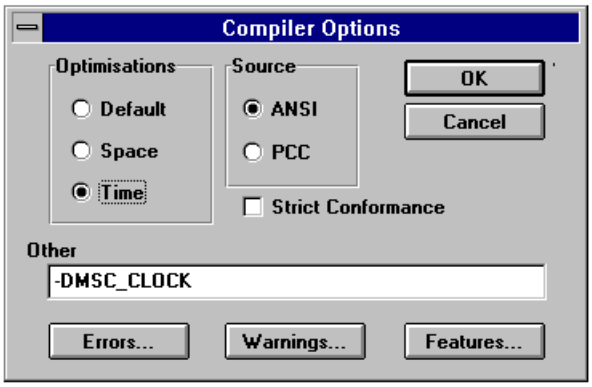
Alter Name... Allows you to change the name of the selected file.

Edit Prams... Displays the Tools Parameters dialog boxand allows you to enter parameters for the tools.

	**Show**		Displays a project summary.
	Close		Closes a project.
	**Compile**	Ctrl+F8	Compiles the file in the currently selected editor window.
	**Build**	Shift+F8	Checks for any source files which have been edited since the project was last built and then builds these files into the target application.
	**Rebuild All**	Alt+F8	Rebuilds the target application from scratch.
	**Stop Build**		Abandons the current build.
	**Execute**		Invokes the Debugger and runs the image.
	**Debug**	Ctrl+F5	Loads the image and invokes the Debugger. A breakpoint is set on the first instruction in `main`.
	Scan Dependencies		Scans the file dependencies and holds them internally.
	**Show Dependencies...**		Displays the file dependency hierarchy.

A.5 Options Menu

A.5.1 Compiler Options



For more information on these options and the options you can enter in the **Other** field, refer to the ARM Software Development Toolkit Reference Manual (ARM DUI0020).

Note: *The compiler options* `-c`, `-S` *and* `-M` *cannot be used in the Project Manager.*

Optimisation options

Default Balances the following two optimisations.

Space Performs optimisations to reduce image size at the expense of increased execution time.

Time Performs optimisations to reduce execution time at the expense of a larger image.

Source options

ANSI Indicates the source is ANSI C standard.

PCC Indicates the source is K&R old-style (PCC) C standard.

Error-suppressing options

Suppress all implicit cast errors
Suppresses all the implicit cast errors, eg. `implicit cast of non-0 int to pointer`. Equivalent to `-ec` command-line option.

Suppress error if 0 length
Suppresses the error if a zero-length array is used.
Equivalent to the `-ez` command-line option.

Suppress syntax checking for #if

Suppresses syntax checking for skipped `#if` statements.
Equivalent to the `-ei` command-line option.

Suppress error casts such as short -> pointer

Suppresses errors for unclean casts such as short to pointer.
Equivalent to the `-ef` command-line option.

Suppress errors if extra chars on preprocessor line

Suppresses the error which occurs if there are extraneous characters at the end of the preprocessor line. Equivalent to the `-ep` command-line option.

Warning-suppressing options

Disable all warnings

Suppresses all warning messages.
Equivalent to the `-W` command-line option.

Use of = in condition context

Suppresses `Use of = in a condition context` warnings. This warning is given when the compiler encounters a statement such as:

```
if (a = b) {...
```

The warning is already suppressed in `-pcc` mode.
This option is equivalent to the `-Wa` command-line option.

Deprecated declaration

Suppresses messages given when a declaration without argument types is encountered in ANSI mode (the warning is suppressed in `-pcc` mode).

This option is equivalent to the `-Wd` command-line option.

Inventing extern int <function>

Suppresses `Inventing extern int` message, which may be useful when compiling old-style C in ANSI mode. Warning is suppressed in `-pcc` mode.

This option is equivalent to the `-Wf` command-line option.

Implicit narrowing cast

Suppresses `Implicit narrowing cast` warning. This warning is issued when the compiler detects the implicit narrowing of a long expression in an int or char context, or the implicit narrowing of a floating-point expression in an integer or narrower floating-point context.

This option is equivalent to the `-Wh` command-line option.

Implicit returning non void context

Suppresses `Implicit return in non-void context` warning. This is most often caused by a return from a function which was assumed to return int (because no other type was specified) but is being used as a void function.

This option is equivalent to the `-Wv` command-line option.

Non ANSI #include <...>

Suppresses `non-ANSI #include <...>` warning. ANSI requires that you only use `#include <...>` for ANSI headers, but it is useful to disable this warning when compiling code not conforming to this aspect of the standard.

This option is equivalent to the `-Wp` command-line option.

Feature options

Embedded function names in code area

Embeds function names in the code area. This improves the readability of the output produced by the stack backtrace run-time support function and the `_mapstore()` function. However, it does increase the size of the code area by around 5%.

This option is equivalent to the `-fn` command-line option.

String literals writable

Allow string literals to be writeable by allocating them in the program's data area rather than the notionally read-only code area. Note that this also stops the compiler re-using a multiply occurring string literal.

This option is equivalent to the `-fw` command-line option.

Always use signed int as type of enum

This option is equivalent to the `-fy` command-line option.

External objects not declared before use

This options is equivalent to the `-fh` command-line option.

Data Flow Anomalies

Checks for certain types of data flow anomalies. The compiler performs data flow analysis as part of code generation. The checks enabled by this option indicate when an automatic variable might have been used before it has been assigned a value.

This option is equivalent to the `-fa` command-line option.

Unused declarations

Reports on all unused declarations, including those from standard headers.

This option is equivalent to the `-fv` command-line option.

Unused preprocessor symbols

Reports on preprocessor symbols defined but not used during compilation.

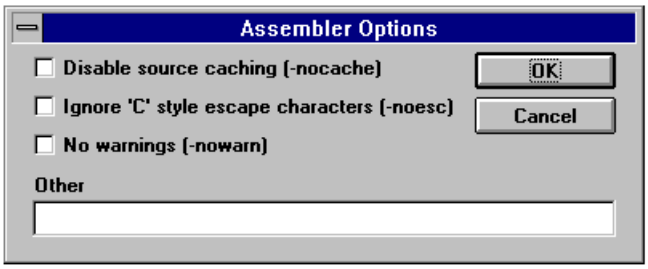
This option is equivalent to the `-fm` command-line option.

Explicit casts of integer to pointer

Reports on explicit casts of integers into pointers,
eg. `char *cp = (char *) anInteger;`
This warning indicates potential portability problems in future.

This option is equivalent to the `-fp` command-line option.

A.5.2 Assembler Options



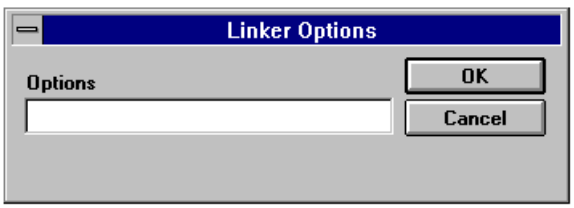
Disable source caching
Turns off source caching (the default is on). Source caching is performed when reading source files on the first pass, so that they can be read from memory during the second pass. This option is equivalent to the `-NOCache` command-line option.

Ignore 'C' style escape characters
Ignores C-style special characters (`\n`, `\t` etc.). This is equivalent to the `-NOEsc` command-line option.

No warning Turns off warning messages. This is equivalent to the `-NOWarn` command-line option.

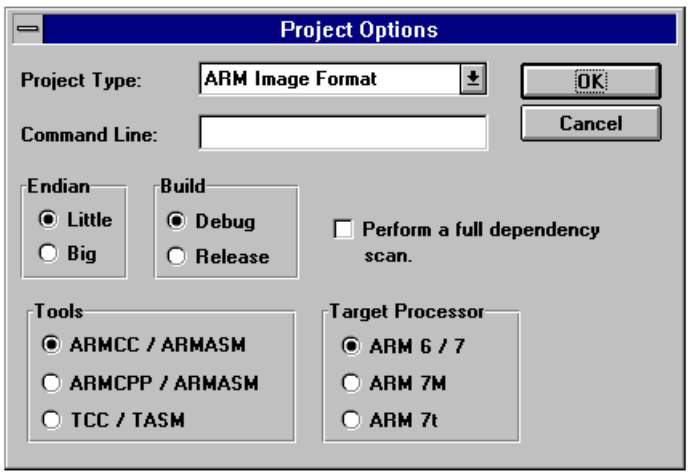
For more information on the options you can enter in the **Other** field, refer to the ARM Software Development Toolkit Reference Manual (ARM DUI0020).

A.5.3 Linker Options



For information on the linker options, refer to the ARM Software Development Toolkit Reference Manual (ARM DUI0020).

A.5.4 Project Options



Project Type Specifies whether the project is built into an image or a library.

Command Line Specifies the arguments for the image when it is run in the Debugger.

Endian Specifies the endianness.

Build Builds your project as a Debug or Release version:

- **Debug** Uses the `-g` command-line option for the compiler or assembler.
- **Release** Generates fully optimised code. The linker will still generate debug information unless you use the `-nodebug` option.

Tools Specifies the tools to be used.

`.c` or `.cpp` files are always compiled using armcc, armcpp or tcc depending on the setting of this option.
`.s` files are always compiled using armasm or tasm.

Target Processor Specifies the target processors.

Note: *The tools ARMCPP/ARMASM and TCC/TASM may be disabled if you have not licensed ARM C++ or Thumb tools.*

A.5.5 Editor Options

Tab Stops Specifies the tab settings.

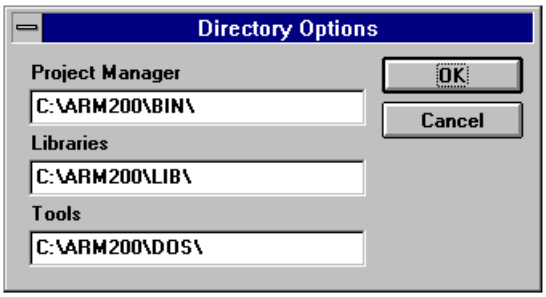
Font... Displays the Font dialog box and allows you to specify the font, style and size of the text.

Use CodeWright Specifies that you wish to use CodeWright as the default editor. CodeWright must have DDE enabled. Refer to the Release Notes for more information on enabling DDE.

Location Specifies the location of the CodeWright executable.

Browse... Allows you to select your location by viewing the directory hierarchy.

A.5.6 Directories Options

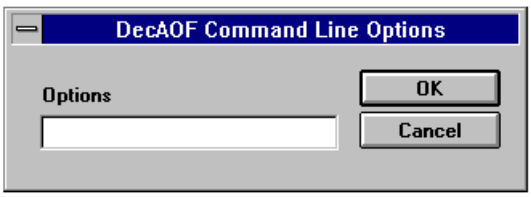


Project Manager Specifies the location of the Project Manager.

Libraries Specifies the location of the libraries.

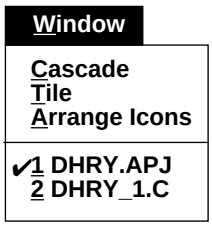
Tools Specifies the location of the tools.

5.2.6 DecAOF Options



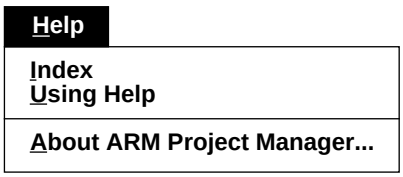
For information on the DecAOF options, refer to the Software Development Toolkit Reference Manual (ARM DUI 0020).

A.6 Windows Menu



Cascade	Arranges windows in an overlapped fashion.
Tile	Arranges windows in non-overlapped tiles.
Arrange Icons	Arranges icons of closed windows.

A.7 Help Menu



Index	Offers you an index of topics on which you can get help.
Using Help	Provides general instructions on using help.
About ARM Project Manager...	Displays the version number of this application.

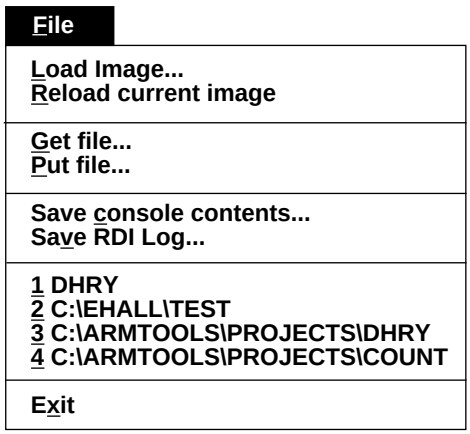
B Debugger Options

This appendix lists and describes the options available in the Debugger.

Many of the options listed in this appendix can also be found on the Toolbar. The relevant icons are shown to the left side of the menu name. Where function key shortcuts are available, these are shown to the right of the menu name.

B.1	File Menu	B-2
B.2	Edit Menu	B-3
B.3	Search Menu	B-4
B.4	View Menu	B-5
B.5	Execute Menu	B-8
B.6	Options Menu	B-9
B.7	Item Menu	B-11
B.8	Window Menu	B-12
B.9	Help Menu	B-13

B.1 File Menu



Load Image... Displays the Open File dialog box. Specify the filename of the image to load and any command-line arguments expected by the program.

Reload current image Reloads the current image. You must choose this command before you can re-execute a program.

Get file... Displays the Open dialog box and allows you to download a specified file directly into memory.

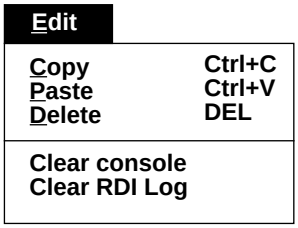
Put file... Copies a specified file onto the disk from memory.

Save console contents... Displays the Save As dialog box and allows you to save the contents of the Console window to a specified file.

Save RDI log... Saves the contents of the RDI Log to a specified file.

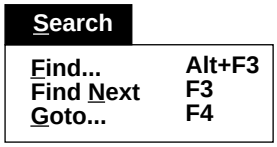
Exit Exits the Debugger.

B.2 Edit Menu



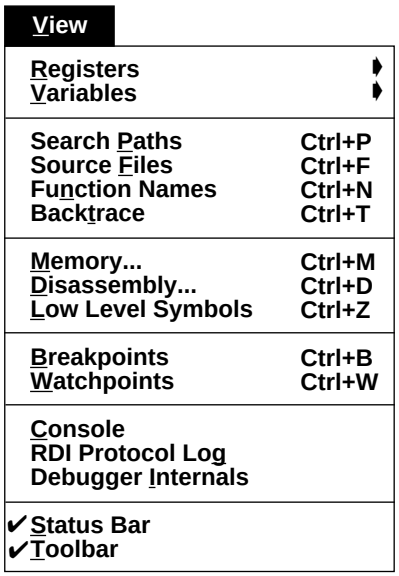
Copy	Ctrl+C	Copies selected data from a source file to the clipboard. If nothing is selected, the entire content of the window is copied to the clipboard.
Paste	Ctrl+V	Pastes data from the clipboard into the console window and an input box at the location of the cursor.
Delete	DEL	Deletes selected text, eg. expressions, breakpoints, watchpoints, search paths.
Clear console		Empties the console window.
Clear RDI log		Empties the RDI log.

B.3 Search Menu



Find...	Alt+F3	Displays the Regular Expression Search dialog box, allows you to search through memory, disassembly or source for specified text and moves to the line (for source) and address (for memory and disassembly) of that specified text.
Find Next	F3	Repeats a Find operation.
Goto...	F4	Moves to a specified line of source.

B.4 View Menu



Registers

Displays a breakdown of all the registers in any mode, allowing you to examine the contents of each register at the current location in your program.
The toolbar icon displays the registers in User mode.

Variables

Allows you to view the values of local and global variables and your own specified expressions.

Local

Displays the values of all the local variables as you step through the program.

Global

Displays the values of all the global variables as you step through the program.

Expression

Allows you to specify an expression and evaluates this expression as you step through the program.

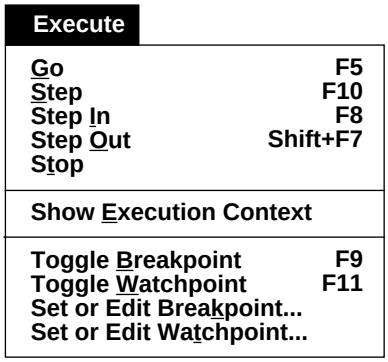
Immediate Evaluation

Displays the value of a selected variable or expression.

	Search Paths	Ctrl+P	Displays search paths for the source files. If you have built your program using source file tools, you may need to add search paths. If you have used the Project Manager this is done automatically for you.
	Source Files	Ctrl+F	Displays source files which are part of the program.
	Function Names	Ctrl+N	Displays a list of functions in the program.
	**Backtrace**	Ctrl+T	Displays information about all currently active procedures, starting with the most recent.
	**Memory...**	Ctrl+M	Displays the Memory Address dialog box and allows you to view the memory at a specified address.
	**Disassembly...**	Ctrl+D	Displays the Disassembly Address dialog box and allows you to view a specified address. The memory is interpreted as machine instructions. You can change the format of the display to force ARM only, Thumb only or let the debugger decide which.
	Low Level Symbols	Ctrl+Z	Displays all the symbols in the image including the ones in the C library.
	Breakpoints	Ctrl+B	Displays a list of breakpoints.
	Watchpoints	Ctrl+W	Displays a list of watchpoints
	Console		Brings the Console window to the front.
	RDI Protocol Log		Brings the RDI Log window to the front.
	Debugger Internals...		Displays internals of the Debugger.
	`statistics`		Displays any statistics which the target processor has been recording.

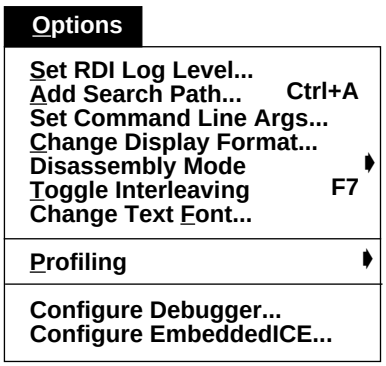
`vector_catch`	Indicates whether or not execution should be caught when various conditions arise. The default value is `%RUsPDAifE`. Capital letters indicate that the condition is to be intercepted. `R` reset `U` undefined instruction `s` SWI `P` prefetch abort `D` data abort `A` address exception `i` IRQ `f` FIQ `E` error.
`cmdline`	Displays the argument string for the debugee.
`rdi_log`	Denotes that RDI logging is enabled if this is non-zero, and serial line logging is enabled if bit 1 is set (initially set to zero).
`clock`	Denotes the number of microseconds since simulation began.
`memstats`	Indicates how many reads or writes have happened to the memory.
`statistics_inc`	Is similar to statistics but displays the difference between the current statistics and those when the statistics variable was last read.
Status Bar	Displays or hides the Status bar.
Toolbar	Displays or hides the Toolbar.

B.5 Execute Menu



	**Go**	F5	Starts execution of the program.
	**Step**	F10	Steps through the program line by line if just the source of the program is displayed, or machine instruction by machine instruction if the source is interleaved with disassembly.
	**Step In**	F8	Steps through a program following all the function calls.
	**Step Out**		Returns from the current location in a called function, to its originating code, immediately after the function call.
	**Stop**	ESC	Stops executing the program.
	Show Execution Context		Centres the code in the Execution window on the current execution marker.
	**Toggle Breakpoint**	F9	Sets or removes a breakpoint at the current execution marker.
	**Toggle Watchpoint**	F11	Sets or removes a watchpoint.
	Set or Edit Breakpoint		Allows you to create complex breakpoints to halt the program when execution reaches that point for the *n*th time.
	Set or Edit Watchpoint		Allows you to create a complex watchpoint to halt the program when a variable or expression reaches a specified value.

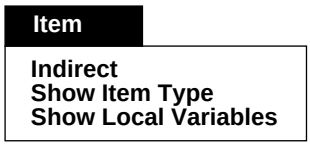
B.6 Options Menu



Set RDI Log Level...		Specifies the amount of information displayed in the RDI Log window.
Add Search Path...	Ctrl+A	Allows you to add a specified file to the search path.
Set Command Line Args...		Allows you to change the command-line arguments of the program when entering the Debugger.
Change Display Format...		Allows you to change the format of the current window. eg. specifying %#x changes the format to hex, specifying nothing reverts to the default format.
Disassembly Mode		Allows you to select a mode of ARM/Thumb, ARM only or Thumb only.
Toggle Interleaving	F7	Toggles between interleaving the source with machine instructions and just displaying the source.
Change Text Font...		Changes the text font in all of the windows.
Profiling		Displays profiling information. Profiling has the following options:
 **Toggle Profiling**		Activates or deactivates the ARM Profiler. When profiling is On, the Profiler displays a flat profile giving the percentage time spent in each function excluding the time spent in any of its children.

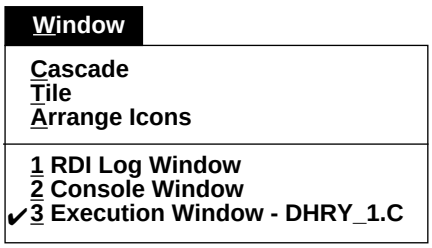
Call graph profiling	Displays additional profiling information; the percentage time accounted for by calls to all children of each function and the percentage time allocated to calls from different parents.
Clear collected	Clears the profiling information displayed.
Write to file	Allows you to write the profiling information to a file. For more information about the profiler, see the ARM Software Development Toolkit Reference Manual (ARM DUI 0020).
Configure Debugger...	Displays the Debugger Configuration property sheet. The RDI property page allows you to select the target, eg. ARMulator or Remote Debugger. The ARMulator property page allows you to select the processor you wish the ARMulator to emulate.
Configure EmbeddedICE...	Displays the EmbeddedICE Configuration dialog box.

B.7 Item Menu



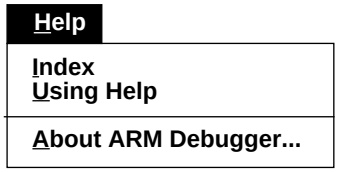
Indirect	This is context sensitive and provides additional information about a current item.
Show Item Type	Displays the type of a selected item.
Show Local Variables	Displays the local variables for a particular context from a backtrace window.

B.8 Window Menu



Cascade	Arranges windows in an overlapped fashion.
Tile	Arranges windows in non-overlapped tiles.
Arrange Icons	Arranges icons of closed windows.

B.9 Help Menu



Index	Offers you an index of topics on which you can get help.
Using Help	Provides general instructions on using help.
About ARM Debugger...	Displays the version number of this application.

Index

A

About APM option A-15, B-13
Accessing
    Debugger 2-15, 3-3
    Project Manager 2-3
Add search paths option B-9
Adding
    project files 2-5
    search paths B-9
amasm tool 1-3
Applications
    within the Windows Toolkit 1-6
ARM Debugger window 3-3
ARM6 PIE card
    definition 1-3
    remote debugging 4-3
ARM7 PIE card
    definition 1-3
    remote debugging 4-6
armcc
    floating point A-8
    integers A-8, A-9
    pointers A-9
    standard headers A-9
armcc tool 1-3
armlink tool 1-3
armsd tool 1-3
Arrange icons option A-14, B-12
Assembler options 2-10, A-10

B

Backtrace option B-6
Backtraces
    displaying 3-11
    viewing 3-11
Benchmarking example 5-2

Index

Breakpoints
  complex 3-6
  setting 3-6
  viewing 3-7
Build logs
  viewing 2-14
Build project option A-6
Building
  projects 2-14

C

Cascade option A-14, B-12
Change display format option B-9
Change text font option B-9
Clear console option B-3
Clear RDI log option B-3
Close file option A-2
Close project option A-6
Comments
  on the book 1-5
  on the software 1-5
Compile file option A-6
Compiler options 2-9, A-7
Complex breakpoints 3-6
Complex watchpoints 3-8
Configuration option B-10
Configuring
  advanced 4-11
  Debugger 4-5
    for EmbeddedICE 4-10
    for serial debugging 4-7
  EmbeddedICE 4-11
Console window 3-3
Conventions
  filenaming 1-4
  in the book 1-2
Copy option A-3, B-3
Creating
  projects 2-4
Cut option A-3

D

Debug option A-6
Debugger
  accessing 2-15, 3-3
  advanced configuring 4-11
  configuring 4-5, 4-11
    for EmbeddedICE 4-10
    for serial debugging 4-7
  entering 2-15, 3-3
  overview 3-2
Debugger icon 3-3
Debugger internals option B-6
Debugger operations 3-2
DecAOF options 2-11
decaof tool 1-3
Decoder/disassembler options 2-11
Definitions
  Context menu 1-7
  Function keys 1-7
  hardware 1-3
  Pulldown menus 1-7
  Toolbar 1-7
  tools 1-3
Delete option A-3, B-3
Deleting
  project files 2-5
Dependencies
  scanning 2-15
  within projects 2-15
Directories options 2-12, A-12
Disassembly mode option B-9
Disassembly option B-6
Displaying
  backtraces 3-11
  images 3-3
  registers 3-10
  variables 3-9

E

Edit menu A-3, B-3
Edit params option 2-13
Edit project option A-5
Editing
  file parameters 2-13
  project files 2-6
Editor options 2-12, A-12
EmbeddedICE
  configuring 4-11
  definition 1-3
  remote debugging 4-8
Entering
  Debugger 2-15, 3-3
  Project Manager 2-3
Error suppressing options 2-9, A-7
Examples
  benchmarking 5-2
  software development 5-10
  using the Debugger 5-1
  using the Project Manager 5-1
Execute menu B-8
Execute option A-6
Executing
  images 3-4
Execution window 3-3
Exit file option A-2, B-2
Expressions
  setting 3-10

F

Feature options 2-9, A-9
File menu A-2, B-2
Files
  adding to a project 2-5
  editing 2-6
  editing parameters 2-13
  naming conventions 1-4
  removing from a project 2-5
  types in a project 2-2
  used in the Windows Toolkit 1-4
  within a project 2-5
Find next option B-4
Find option A-3, B-4
Function names option B-6

G

Get file option B-2
Go option B-8
Goto option B-4

H

Hardware
  ARM6 PIE card 1-3
  ARM7 PIE card 1-3
  EmbeddedICE 1-3
Hardware definitions 1-3
Help
  in the Windows Toolkit 1-8
  menu A-15, B-13

I

Icons
  Debugger 3-3
  Project Manager 2-3
Images
  displaying 3-3
  executing 3-4
  loading 3-3
Immediate evaluation option B-5
Index option A-15, B-13
Indirect option B-11
Item menu B-11

Index

L

Linker options 2-11, A-10
Load image option B-2
Loading
    images 3-3
Low level symbols option B-6

M

Memory option B-6
Menus
    edit A-3, B-3
    execute B-8
    file A-2, B-2
    help A-15, B-13
    item B-11
    options A-7, B-9
    overview 1-7
    project A-5
    search B-4
    view A-4, B-5
    windows A-14, B-12

N

Navigating programs 3-5
New file option A-2
New project option A-5

O

Online help 1-8
Open file option A-2
Open project option A-5
Opening
    projects 2-4
Opening files A-2
Operations
    in the Debugger 3-2
    within the Project Manager 2-2
Optimisation options 2-9, A-7
Options
    about APM A-15, B-13
    add search paths B-9
    arrange icons A-14, B-12
    assembler 2-10, A-10
    backtrace B-6
    build project A-6
    cascade A-14, B-12
    change display format B-9
    change text font B-9
    clear console B-3
    clear RDI log B-3
    close file A-2
    close project A-6
    compile file A-6
    compiler 2-9, A-7
    configuration B-10
    copy A-3, B-3
    cut A-3
    debug A-6
    debugger internals B-6
    DecAOF 2-11
    decoder/disassembler 2-11
    delete A-3, B-3
    directories 2-12, A-12
    disassembly B-6
    disassembly mode B-9
    edit params 2-13
    edit project A-5
    editor 2-12, A-12
    error suppressing A-7
    execute A-6
    exit file A-2, B-2
    features A-9
    find A-3, B-4
    find next B-4
    function names B-6
    get file B-2
    go B-8
    goto B-4
    immediate evaluation B-5
    index A-15, B-13
    indirect B-11
    linker 2-11, A-10
    load image B-2

low level symbols B-6
memory B-6
new file A-2
new project A-5
open file A-2
open project A-5
optimisation A-7
overview of Project Manager 2-7
paste A-3, B-3
print file A-2
print preview A-2
print setup A-2
project 2-7, A-11
  build version 2-8
put file B-2
RDI protocol logs B-6
rebuild all A-6
reload current image B-2
repeat A-3
replace A-3
Save as A-2
save console contents B-2
save file A-2
save RDI log B-2
scan dependencies A-6
search paths B-6
select all A-3
set command-line args B-9
set or edit breakpoint B-8
set or edit watchpoint B-8
set RDI log level B-9
show dependencies A-6
show execution context B-8
show item type B-11
show local variables B-11
show project A-6
source A-7
source files B-6
status bar A-4, B-7
step B-8
step in B-8
step out B-8
stop B-8, B-9
stop build A-6
tile A-14, B-12
toggle breakpoint B-8
toggle interleaving B-9
toggle watchpoint B-8
toolbar A-4, B-7
undo A-3
using help A-15, B-13
view breakpoints B-6
view build logs A-4
view console B-6
view expressions B-5
view next error A-4
view previous error A-4
view registers B-5
view variables B-5
view watchpoints B-6
warning suppressing A-8
Options menu A-7, B-9
Overviews
  Debugger 3-2
  Project Manager 2-2
  remote debugging 4-2
  Windows Toolkit 1-6

P

Parameters
  editing file 2-13
Paste option A-3, B-3
Print file option A-2
Print preview option A-2
Print setup option A-2
Programming tools 1-3
Project Manager
  accessing 2-3
  entering 2-3
  operations 2-2
  options, see Options
  overview 2-2
Project menu A-5
Project options 2-7, A-11
Project summary 2-6

Index

Projects
  building 2-14
  creating 2-4
  dependencies 2-15
  displaying a summary 2-6
  file types 2-2
  managing files 2-5
  opening 2-4
Pulldown 1-7
Put file option B-2

R

RDI Log window 3-3
RDI protocol logs option B-6
Rebuild All option A-6
Registers
  displaying 3-10
  viewing 3-10
Reload current image option B-2
Remote debugging
  overview 4-2
  using ARM6 PIE card 4-3
  using ARM7 PIE card 4-6
  using EmbeddedICE 4-8
Removing
  project files 2-5
Repeat option A-3
Replace option A-3

S

Save As option A-2
Save console contents option B-2
Save file option A-2
Save RDI log option B-2
Scan Dependencies option A-6
Scanning
  project dependencies 2-15
Search menu B-4
Search paths option B-6
Select All option A-3
Set command-line args option B-9
Set or edit breakpoint option B-8
Set or edit watchpoint option B-8
Set RDI log level option B-9
Setting
  breakpoints 3-6
  expressions 3-10
  watchpoints 3-8
Shortcuts 2-2
Show dependencies option A-6
Show execution context option B-8
Show item type option B-11
Show local variables option B-11
Show project option A-6
Showing
  project dependencies 2-15
Software development example 5-10
Source files option B-6
Source option 2-9
Source options A-7
Status bar option A-4, B-7
Step in option B-8
Step option B-8
Step out option B-8
Stepping through programs 3-5
Stop Build option A-6
Stop option B-8, B-9
Summary
  of a project 2-6

T

tasm tool 1-3
tcc tool 1-3
Tile option A-14, B-12
Toggle breakpoint option B-8
Toggle interleaving option B-9
Toggle watchpoint option B-8
Toolbar option A-4, B-7
Tools
  armasm 1-3
  armcc 1-3
  armlink 1-3
  armsd 1-3
  decaof 1-3

tasm 1-3
tcc 1-3
used by the Toolkit 1-3
Types
of files in a project 2-2

U

Undo option A-3
Using help option A-15, B-13

V

Variables
displaying 3-9
viewing 3-9
View build logs option A-4
View expressions option B-5
View menu A-4, B-5
View next error option A-4
View previous error option A-4
Viewing
backtraces 3-11
breakpoints 3-7
breakpoints option B-6
build logs 2-14
console option B-6
register option B-5
registers 3-10
variables 3-9
variables option B-5
watchpoints 3-9
watchpoints option B-6

W

Warning options 2-9
Warning suppressing options A-8
Warnings
suppressing 2-9
Watchpoints
complex 3-8
setting 3-8
viewing 3-9
Windows
ARM Debugger 3-3
console 3-3
execution 3-3
RDI Log 3-3
when entering Debugger 3-3
Windows menu A-14, B-12
Windows Toolkit
help 1-8
menus 1-7
overview 1-6
Worked examples 5-1

Index