Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. CodeWarrior is a trademark or registered trademark of Freescale Semiconductor, Inc. in the United States and/or other countries. All other product or service names are the property of their respective owners.

Copyright © 2006 by Freescale Semiconductor, Inc. All rights reserved.

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

How to Contact Us

<table>
<thead>
<tr>
<th>Corporate Headquarters</th>
<th>Freescale Semiconductor, Inc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7700 West Parmer Lane</td>
</tr>
<tr>
<td></td>
<td>Austin, TX 78729</td>
</tr>
<tr>
<td></td>
<td>U.S.A.</td>
</tr>
<tr>
<td>World Wide Web</td>
<td><a href="http://www.freescale.com/codewarrior">http://www.freescale.com/codewarrior</a></td>
</tr>
<tr>
<td>Technical Support</td>
<td><a href="http://www.freescale.com/support">http://www.freescale.com/support</a></td>
</tr>
</tbody>
</table>
Table of Contents

1 Introduction 7
   Read the Developer Notes ........................................... 7
   Features .............................................................. 7
   CodeWarrior Editions ................................................. 8
   About this Manual .................................................... 9
   Documentation Overview ............................................ 10
   Additional Information Resources ................................ 10

2 Getting Started 13
   System Requirements .................................................. 13
   Host Requirements .................................................... 13
   Target Board Requirements ......................................... 14
   CodeWarrior IDE ...................................................... 14
   CodeWarrior Development Process ................................ 15
      Project Files ....................................................... 17
      Editing Code ....................................................... 18
      Building: Compiling and Linking ................................ 18
      Debugging ......................................................... 19
      Disassembling .................................................... 19

3 Application Tutorial 21
   Create a Project ...................................................... 21
   Build the Project .................................................... 25
   Debug the Application .............................................. 27

4 Target Settings 33
   Target Settings Overview .......................................... 33
   ColdFire Settings Panels .......................................... 34
      Target Settings ................................................... 35
      BatchRunner PreLinker ........................................... 37
      BatchRunner PostLinker .......................................... 37
      ColdFire Target .................................................. 38
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColdFire Assembler</td>
<td>38</td>
</tr>
<tr>
<td>ELF Disassembler</td>
<td>41</td>
</tr>
<tr>
<td>ColdFire Processor</td>
<td>45</td>
</tr>
<tr>
<td>ColdFire Linker</td>
<td>48</td>
</tr>
<tr>
<td>Debugger PIC Settings</td>
<td>53</td>
</tr>
<tr>
<td><strong>5 Debugging</strong></td>
<td>55</td>
</tr>
<tr>
<td>Target Settings for Debugging</td>
<td>55</td>
</tr>
<tr>
<td>CF Debugger Settings Panel</td>
<td>56</td>
</tr>
<tr>
<td>Remote Debugging Panel</td>
<td>59</td>
</tr>
<tr>
<td>CF Exceptions Panel</td>
<td>63</td>
</tr>
<tr>
<td>Debugger Settings Panel</td>
<td>66</td>
</tr>
<tr>
<td>CF Interrupt Panel</td>
<td>68</td>
</tr>
<tr>
<td>Remote Connections for Debugging</td>
<td>69</td>
</tr>
<tr>
<td>Abatron Remote Connections</td>
<td>71</td>
</tr>
<tr>
<td>Freescale Remote Connections</td>
<td>73</td>
</tr>
<tr>
<td>P&amp;E Microsystems Remote Connections</td>
<td>76</td>
</tr>
<tr>
<td>ISS Remote Connection</td>
<td>80</td>
</tr>
<tr>
<td>BDM Debugging</td>
<td>83</td>
</tr>
<tr>
<td>Connecting a P&amp;E Parallel Connector</td>
<td>83</td>
</tr>
<tr>
<td>Connecting an Abatron BDI Device</td>
<td>83</td>
</tr>
<tr>
<td>Debugging ELF Files without Projects</td>
<td>85</td>
</tr>
<tr>
<td>Updating IDE Preferences</td>
<td>85</td>
</tr>
<tr>
<td>Customizing the Default XML Project File</td>
<td>86</td>
</tr>
<tr>
<td>Debugging an ELF File</td>
<td>87</td>
</tr>
<tr>
<td>Additional ELF-Debugging Considerations</td>
<td>87</td>
</tr>
<tr>
<td>Special Debugger Features</td>
<td>88</td>
</tr>
<tr>
<td>ColdFire Menu</td>
<td>88</td>
</tr>
<tr>
<td>Working with Target Hardware</td>
<td>89</td>
</tr>
<tr>
<td>Using the Simple Profiler</td>
<td>90</td>
</tr>
<tr>
<td><strong>6 Instruction Set Simulator</strong></td>
<td>91</td>
</tr>
<tr>
<td>Features</td>
<td>91</td>
</tr>
<tr>
<td>ColdFire V2</td>
<td>91</td>
</tr>
<tr>
<td>ColdFire V4e</td>
<td>92</td>
</tr>
</tbody>
</table>
Table of Contents

9 Memory Configuration Files 121
   Command Syntax ........................................ 121
   Command Explanations ................................ 122
   range ....................................................... 122
   reserved .................................................. 123
   reservedchar .............................................. 123

Index 125
Introduction

This manual explains how to use CodeWarrior™ development tools to develop applications for the Freescale™ ColdFire® family of integrated microprocessors.

This chapter consists of these sections:

- Read the Developer Notes
- Features
- CodeWarrior Editions
- About this Manual
- Documentation Overview
- Additional Information Resources

Read the Developer Notes

Before using the CodeWarrior IDE, read the developer notes. These notes contain important information about last-minute changes, bug fixes, incompatible elements, or other topics that may not be included in this manual.

NOTE The release notes for specific components of the CodeWarrior IDE are located at location: (CodeWarrior_Dir)\Release_Notes, where (CodeWarrior_Dir) is the CodeWarrior installation directory.

If you are new to the CodeWarrior IDE, read this chapter and the Getting Started chapter. This chapter provides references to resources of interest to new users; the Getting Started chapter helps you become familiar with the software features.

Features

The CodeWarrior Development Studio for ColdFire Architectures includes these features:

- Support for the latest ColdFire processors: MCF5222x and MCF5223x.
- Support for previous processors of the ColdFire family, such as MCF547x/548x, MCF5206, MCF5208, MCF5213 (and its variants MCF5211 and MCF5212),...
Introduction

CodeWarrior Editions

MCF523x, MCF5282, MCF5271, MCF5272, MCF5275, MCF5282, MCF5307, MCF5329, and MCF5249. For more information, see ColdFire Processor:

- Flash-programmer and hardware-diagnostics support. For more information, see Using Hardware Tools.
- Remote connection debugging support for a range of protocols:
  - Abitron protocols, see Abatron Remote Connections.
  - Freescale’s USB TAP, see Freescale Remote Connections.
  - P&E Microsystems protocols, see P&E Microsystems Remote Connections.
- Instruction Set Simulator (ISS) for V2 and V4e processor cores. For more information, see Remote Connections for Debugging and Instruction Set Simulator.
- Simple profiler support. For more information, see Using the Simple Profiler. (This profiler support is not available for MCF521x, MCF5222x, and MCF5223x processors.)

CodeWarrior Editions

There are three editions of CodeWarrior™ Development Studio for ColdFire® Architectures, version 6.3. Table 1.1 shows their feature differences.

Table 1.1 CodeWarrior ColdFire 6.3 Edition Features

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IDE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Compiles source code</td>
<td>ASM and C</td>
<td>ASM and C</td>
<td>ASM, C, and C++</td>
</tr>
<tr>
<td>Code size restrictions</td>
<td>128KB</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Compiler optimization levels</td>
<td>Unlimited</td>
<td>Unlimited</td>
<td>Unlimited</td>
</tr>
<tr>
<td>3rd-party plug-ins</td>
<td>No RTOS</td>
<td>No RTOS</td>
<td>Unlimited RTOS plug-ins</td>
</tr>
<tr>
<td>CodeWarrior Debugger</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table 1.1  CodeWarrior ColdFire 6.3 Edition Features (continued)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugger hardware connections</td>
<td>P&amp;E Parallel and USB, Cyclone Max, and USB TAP</td>
<td>P&amp;E Parallel and USB, Cyclone Max, and USB TAP</td>
<td>P&amp;E Parallel, USB, and Lightning; Abatron serial and TCP/IP, Cyclone Max, and USB TAP</td>
</tr>
<tr>
<td>V2, V4e simulator</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Flash programmers</td>
<td>CodeWarrior Flash Programmer (129 megabytes) and ColdFire Flasher standalone plug-in</td>
<td>CodeWarrior Flash Programmer and ColdFire Flasher standalone plug-in</td>
<td>CodeWarrior Flash Programmer and ColdFire Flasher standalone plug-in</td>
</tr>
<tr>
<td>Real time operating system (RTOS)</td>
<td>Not available</td>
<td>Not available</td>
<td>Plug-ins available</td>
</tr>
<tr>
<td>Availability</td>
<td>Free with evaluation board</td>
<td>Available through all channels</td>
<td>Available through all channels. 30-day evaluation copy also available</td>
</tr>
</tbody>
</table>

About this Manual

Table 1.2 lists the contents of this manual.

Table 1.2  Chapter, Appendix Contents

<table>
<thead>
<tr>
<th>Chapter/Appendix</th>
<th>Explains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>New features; contents of this manual; technical support; further documentation</td>
</tr>
<tr>
<td>Getting Started</td>
<td>System requirements; overview of CodeWarrior development tools</td>
</tr>
<tr>
<td>Application Tutorial</td>
<td>Tutorial for writing and debugging programs</td>
</tr>
<tr>
<td>Target Settings</td>
<td>Controlling the compiler and linker</td>
</tr>
</tbody>
</table>
Introduction

Table 1.2 Chapter, Appendix Contents (continued)

<table>
<thead>
<tr>
<th>Chapter/Appendix</th>
<th>Explains</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debugging</td>
<td>Debugger settings panels; remote debugging connections</td>
</tr>
<tr>
<td>Instruction Set Simulator</td>
<td>Instruction Set Simulator, including configuration for your requirements.</td>
</tr>
<tr>
<td>Using Hardware Tools</td>
<td>Flash programmer and hardware diagnostics tools</td>
</tr>
<tr>
<td>Using Debug Initialization Files</td>
<td>Debug initialization files</td>
</tr>
<tr>
<td>Memory Configuration Files</td>
<td>Defining access for areas of memory</td>
</tr>
</tbody>
</table>

Documentation Overview

Documentation for your CodeWarrior tools comes in three formats:

- **PDF manuals** — in subdirectory \Help\PDF of your installation directory.

  **NOTE** For complete information about a particular topic, you may need to look in this Targeting manual and in the corresponding generic CodeWarrior manual.

  To view any PDF document, you need Adobe® Acrobat® Reader software, which you can download from: [http://www.adobe.com/acrobat](http://www.adobe.com/acrobat)

- **CHM help files** — information in Microsoft® HTML Help CHM format, in folder \Help of the CodeWarrior installation directory. To view this information, start the CodeWarrior IDE, then select Help > Online Manuals from the main menu bar.

- **CodeWarrior online help** — information about using the IDE and understanding error messages. To access this information, start the CodeWarrior IDE, then select Help > CodeWarrior Help from the main menu bar.

Additional Information Resources

- CodeWarrior IDE and related documentation can be found in the \HelpPDF subdirectory of your CodeWarrior installation directory:

- For general information about the CodeWarrior IDE and debugger, see the *IDE 5.7 User's Guide*.

- For information specific to building (compiling and linking), see the *ColdFire Build Tools Reference*. 
Introduction

Additional Information Resources

- For information about the Main Standard Libraries for C/C++, see the *MSL C Reference* and the *MSL C++ Reference*.

- For PDF-format documentation about Freescale processors and cores, go to the \Freescale_Documentation subdirectory of your CodeWarrior installation directory.

- For Freescale documentation and resources, visit the Freescale, Inc. web site: http://www.freescale.com

- For additional electronic-design and embedded-system resources, visit the EG3 Communications, Inc. web site: http://www.eg3.com

- For monthly and weekly forum information about programming embedded systems (including source-code examples), visit the *Embedded Systems Programming* magazine web site: http://www.embedded.com
Getting Started

This chapter helps you install the CodeWarrior™ Development Studio for ColdFire Architectures. It also gives an overview of the CodeWarrior environment and tools.

This chapter consists of these sections:

- System Requirements
- CodeWarrior IDE
- CodeWarrior Development Process

System Requirements

Your host computer system and your target board must meet minimum requirements.

Host Requirements

Your computer (PC) needs:

- 800 MHz Pentium®-compatible microprocessor
- Windows® 2000 or XP operating system
- 512 megabytes of RAM
- CD-ROM drive
- 350 megabytes free memory space, plus space for projects and source code
- Serial port (or Ethernet connector), to connect your PC to the embedded target — for debugging with an Abatron BDI device or P&E Cyclone Max through Serial
- Parallel port (or P&E Lightning board) — to use a P&E parallel cable to connect to BDM/JTAG targets
- USB port — To use a P&E USB BDM debug cable (Cyclone Max or P&E USB Multilink) or Freescale USB TAP
- Ethernet connector -- to connect your PC to the embedded target for debugging with an Abatron BDI device or P&E Cyclone Max through TCP/IP
Target Board Requirements

Your functional embedded system needs:

- ColdFire evaluation board, with a supported processor.
- Serial or null-modem cables to connect the host computer and target board, in case you would like to send printf output to a terminal; your target board determines the specific cables you need.
- BDM connector to be able to connect a supported BDM cable.
- Appropriate power supply for the target board.

CodeWarrior IDE

The CodeWarrior IDE consists of a project manager, a graphical user interface, compilers, linkers, a debugger, a source-code browser, and editing tools. You can edit, navigate, examine, compile, link, and debug code, within the one CodeWarrior environment. The CodeWarrior IDE lets you configure options for code generation, debugging, and navigation of your project.

Unlike command-line development tools, the CodeWarrior IDE organizes all files related to your project. You can see your project at a glance, so organization of your source code files is easy. Navigation among those files is easy, too.

When you use the CodeWarrior IDE, there is no need for complicated build scripts or makefiles. To add or delete source code files from a project, you use your mouse and keyboard, instead of tediously editing a build script.

For any project, you can create and manage several configurations for use on different computer platforms. The platform on which you run the CodeWarrior IDE is called the host. From the host, you can use the CodeWarrior IDE to develop code to target various platforms.

Note the two meanings of the term target:

- **Platform Target** — The operating system, processor, or microcontroller in which/on which your code will execute.
- **Build Target** — The group of settings and files that determine what your code is, as well as controlling the process of compiling and linking.

The CodeWarrior IDE lets you specify multiple build targets. For example, a project can contain one build target for debugging and another build target optimized for a particular operating system (platform target). These build targets can share project files, even though each build target uses its own settings. After you debug the program, the only actions necessary to generate a final version are selecting the project’s optimized build target and using a single `make` command.
The CodeWarrior IDE’s extensible architecture uses plug-in compilers and linkers to target various operating systems and microprocessors. For example, the IDE internally calls a C translator, compiler, and linker.

Most features of the CodeWarrior IDE apply to several hosts, languages, and build targets. However, each build target has its own unique features. This manual explains the features unique to the CodeWarrior IDE for Freescale ColdFire processors.

For comprehensive information about the CodeWarrior IDE, see the Code Warrior IDE User’s Guide.

**CodeWarrior Development Process**

The CodeWarrior IDE helps you manage your development work more effectively than you can with a traditional command-line environment. Figure 2.1 depicts application development using the IDE.
Getting Started
CodeWarrior Development Process

Figure 2.1 CodeWarrior IDE Application Development

Notes:
(1) Use any combination: stationery (template) files, library files, or your own source files.
(2) Compiler, linker, debugger settings; target specification; optimizations.
(3) Edit source and resource files.
(4) Possible corrections: adding a file, changing settings, or editing a file.
Project Files

A CodeWarrior project consists of source-code, library, and other files. The project window (Figure 2.2) lists all files of a project, letting you:

- Add files
- Remove files
- Specify the link order
- Assign files to build targets
- Have the IDE generate debug information for files

Figure 2.2  Project Window

NOTE  Figure 2.2 shows a floating project window. Alternatively, you can dock the project window in the IDE main window or make the project window a child of the main window. You can have multiple project windows open at the same time; if the windows are docked, their tabs let you control which one is at the front of the main window.

The CodeWarrior IDE automatically handles dependencies among project files, storing compiler and linker settings for each build target. The IDE tracks which files have changed since your last build, recompiling only those files during your next project build.

A CodeWarrior project is analogous to a collection of makefiles, as the same project can contain multiple builds. Examples are a debug version and release version of code, both
Getting Started
CodeWarrior Development Process

part of the same project. As earlier text explained, build targets are such different builds within a single project.

Editing Code

The CodeWarrior text editor handles text files in MS-DOS, UNIX, and MacOS formats. To edit a source code file (or any other editable project file), double-click its filename in the project window. The IDE opens the file in the editor window (Figure 2.3). This window lets you switch between related files, locate particular functions, mark locations within a file, or go to a specific line of code.

Figure 2.3 Editor Window

NOTE Figure 2.3 shows a floating editor window. Alternatively, you can dock the project window in the IDE main window or make the project window a child of the main window.

Building: Compiling and Linking

For the CodeWarrior IDE, building includes both compiling and linking. To start building, you select Project > Make, from the IDE main menu bar. The IDE compiler:

- Generates an object-code file from each source-code file of the build target, incorporating appropriate optimizations.
Getting Started

CodeWarrior Development Process

- Updates other files of the build target, as appropriate.
- In case of errors, issues appropriate messages and halts.

When compilation is done, building moves on to linking. The IDE linker:
- Links the object files into one executable file, in the link order you specify.
- In case of errors, issues appropriate error messages and halts.

When linking is done, you are ready to test and debug your application.

NOTE
It is possible to compile a single source file. To do so, select the filename in the project window, then select Project > Compile from the main menu bar. Another useful option is compiling only the modified files of the build target: select Project > Bring Up To Date from the main menu bar.

Debugging

To debug your application, select Project > Debug from the main menu bar. The debugger window opens, displaying your program code.

Run the application from within the debugger to observe results. The debugger lets you set breakpoints, to check register, parameter, and other values at specific points of code execution.

NOTE
To debug code stored in Flash memory, you first must program the Flash. (The Flash Programmer subsection explains how to program a flash device.)

When your code executes correctly, you are ready to add features, to release the application to testers, or to release the application to customers.

NOTE
Another debugging feature of the CodeWarrior IDE is viewing preprocessor output. This helps you track down bugs caused by macro expansions or another subtlety of the preprocessor. To use this feature, specify the output filename in the project window, then select Project > Preprocess from the main menu bar. A new window opens to show the preprocessed file.

Disassembling

To disassemble a compiled or ELF file of your project, select the file’s name in the project window, then select Project > Disassemble. In order to add an ELF file to the project view for disassembly, in Windows Explorer, point to the ELF file and drag it to the project in the CodeWarrior IDE. Right click on the file name and select disassemble.
After disassembling the file, the CodeWarrior IDE creates a .dump file that contains the disassembled file’s object code in assembly format, and debugging information in Debugging With Attribute Record Format (DWARF). The .dump file’s contents appear in a new window.
Application Tutorial

This chapter takes you through the CodeWarrior™ IDE programming environment. This tutorial does not teach you programming. It instead teaches you how to use the CodeWarrior IDE to write and debug applications for a target platform.

The examples in this chapter have been chosen for MCF5208-based code development. If you are using a different ColdFire processor, you will need to substitute the appropriate device name in the examples. Also, this tutorial assumes that debugging will be done using CCS-SIM instead of real hardware. See Remote Debugging Panel for details on connecting to a physical target evaluation board (EVB).

If you are using an EVB, then before you start this tutorial, you should configure the hardware. Typically, this entails:

- Verifying all jumper-header and switch settings,
- Connecting the EVB to your computer, and
- Connecting EVB power.

NOTE For complete setup instructions, see the EVB’s own documentation.

This chapter consists of these sections:

- Create a Project
- Build the Project
- Debug the Application

Create a Project

This section shows how to use stationery to create a new project for a ColdFire EVB, and how to set up the project to make a standalone application. Follow these steps:

1. Select Programs > Freescale CodeWarrior > CodeWarrior for ColdFire V6.3 > CodeWarrior IDE. The CodeWarrior IDE starts and the main window (Figure 3.1) appears.
2. From the main menu bar, select File > New. A New dialog box (Figure 3.2) appears.
   a. Select ColdFire Stationery.
   b. In the Project name text box, type MyProj.
NOTE  The default project location is the CodeWarrior installation directory. For example, if the project name is `abc` and the installation directory is `CodeWarrior_Dir`, the default location is `CodeWarrior_Dir\abc`. For a different location, click the `Set` button, then use the subsequent dialog box to specify the location. Clicking `OK` returns you to the `New` dialog box.

c. Click `OK`. The `New Project` dialog box (Figure 3.3) appears.

Figure 3.1  CodeWarrior IDE Main Window

Figure 3.2  New Dialog Box
   a. Click the CF_M5208EVB expand control — the tree structure displays the subordinate option C.
   b. Select C, as Figure 3.4 shows.

NOTE Many possible ColdFire target processors have an external bus, so can use large external RAM devices for debugging applications during development. But M5211, M5212, and M5213 processors do not have an external bus, so must accommodate applications in on-chip memory. Although this on-chip RAM accommodates CodeWarrior stationery, it probably is too small for full development of your application. Accordingly, for these processors, you should locate your applications in flash memory. (The Flash Programmer subsection explains how to program a flash device.)
c. Click **OK**. The CodeWarrior IDE creates a new project consisting of the folders and files (header, initialization, common, and so forth) that the M5208 C stationery specifies. The project window (Figure 3.5) appears.

**Figure 3.5 Project Window**

![Project Window](image)

4. Make sure that the target field (immediately under the project-window tab) specifies M5208EVB Console Debug.

**NOTE** Files in the *project data folder* include information about the project file, various target settings, and object code. Do not change the contents of this folder, or the CodeWarrior IDE could lose project settings.

5. This completes project creation. You are ready to build the project, per the procedure of the next section.

**NOTE** While your source file (`main.c`) is open in the editor window, you can use all editor features to work with your code. If you wish, you can use a third-party editor to create and edit your code, provided that this editor saves the file as plain text. For information about the editor window, touching files, and file synchronization, and removing/adding text files, see *IDE User’s Guide*. 

---

| 24 | ColdFire Architectures v6.3 - Targeting Manual |
Build the Project

This section shows how to select the linker, set up remote debugging, and build (compile and link) your project.

**NOTE** The stationery for this project includes a default setup for the linker specific to the application’s target platform.

Follow these steps:

1. Select the appropriate linker.
   a. Select *Edit > Target Settings* (where *Target* is the name of the current build target). The *Target Settings* window (Figure 3.6) appears.

   **NOTE** In this tutorial, the name of the build target is *M5208EVB Console Debug*. So the Target Settings window title is *M5208EVB Console Debug Settings*.

   **Figure 3.6** Target Settings Window: Target Settings Panel

   ![Target Settings Window: Target Settings Panel](image)

   b. From the *Target Settings Panels* list, select *Target Settings*.
      The *Target Settings* panel moves to the front of the window.
   c. Use the *Linker* list box to specify the Embedded ColdFire Linker.
   d. Click *Apply*. The IDE saves the new linker setting for the build target.
NOTE This linker change applies only to the current build target. To use a different build target, you must specify its appropriate linker.

For an actual target board, instead of the simulator, you would need to make board connections by this point.

2. Set Up Remote Debugging.
   a. From the Target Settings Panels list, select Remote Debugging.

      The Remote Debugging settings panel moves to the front of the Target Settings window, as Figure 3.7 shows.

      Figure 3.7 Target Settings Window: Remote Debugging Panel

      ![Remote Debugging Panel](image)

      b. Use the Connection list box to specify CCS-SIM.
      c. Click OK. The IDE completes the remote debugging setup, and the Target Settings window closes.

3. From the main menu bar, select Project > Make. The IDE updates all files, links code into the finished application, and displays any error messages or warnings in the Errors & Warnings window.

NOTE The Make command applies to all source files: the IDE opens them all, the compiler generates object code, then the linker creates an executable file.
(The Compile command applies only to selected files. The Bring Up To Date
command compiles all changed files, without linking.)
The Project window lets you view compiler progress, or stop the build.

4. This completes building your project. You are ready for the debugging procedure of the next section.

**Debug the Application**

This section explains you how to test whether your application runs as you expect. Topics include starting the debugger, setting a breakpoint, and viewing registers. Follow these steps:

1. Set debugger preferences.
   a. Select *Edit > Target Settings*, (where *Target* is the name of the current build target). The *Target Settings* window appears.
   b. From the *Target Settings Panels* list, select *CF Debugger Settings*. The *CF Debugger Settings* panel moves to the front of the window, as Figure 3.8 shows.

**Figure 3.8 The CF Debugger Settings Panel**

![CF Debugger Settings Panel](image)

   c. Make sure that the *Target Processor* list box specifies 5208 (or the platform target you have specified).
   d. Make sure that the *Target OS* list box specifies BareBoard.
e. Click **OK**. The IDE saves the debugger settings, and the **Target Settings** window closes.

**NOTE**  The default target initialization and memory configuration files are in subdirectory `\E68K_Support\Initialization_Files`, of the CodeWarrior installation directory.

2. From the IDE main menu, select **Project** > **Debug**. A progress bar appears as the system downloads the output file to the target. The debugger starts; the **Debugger** window (**Figure 3.9**) appears.

**NOTE**  For a ROM build target, you must load the application to Flash memory before you can perform Step 2.

**Figure 3.9  Debugger Window**

![Debugger Window](image)

a. Note the toolbar at the top of the window; it includes command buttons **Run**, **Stop**, **Kill**, **Step Over**, **Step Into**, and **Step Out**.

b. Note the **Stack** pane, at the upper left. This pane shows the function calling stack.

c. Note the **Variables** pane, at the upper right. This pane lists the names and values of any local variables.
d. Note the Source pane, the largest pane of the window. This pane displays C/C++ or assembly source code.

3. Set a breakpoint.
   a. In the Source pane, find the line containing the open brace ( { ) character.
   b. In the far left-hand column of this line, click the grey dash. A red circle replaces the dash, indicating that the debugger set a breakpoint at the location. Figure 3.10 shows the red-circle indicator.

Figure 3.10 Setting a breakpoint

4. View registers.
   a. From the main menu bar, select View > Registers. The Registers window (Figure 3.11) appears.
   b. Use the expand controls to drill down through register categories to individual registers — when you reach individual registers, their values appear at the right side of the window.
   c. You may edit register values directly in the Registers window.
   d. Close the Registers window.
5. View memory.
   
a. In the **Source** pane of the **Debugger** window, right-click on **main**. The view-memory context menu (**Figure 3.12**) appears.

**Figure 3.12 View Memory Context Menu**

- Find and Open File...
- Go to function definition of main()
- Preprocess
- Set Software Breakpoint
- Set Thread Specific Breakpoint
- View As...
  - View Data
  - View Memory
  - View Memory As...
b. From this context menu, select View Memory. The View Memory window Figure 3.13 appears.

Figure 3.13 View Memory Window

![View Memory Window]

- Note that the View Memory window displays hexadecimal and ascii values for several addresses, starting at the address of main.
- In the Display text box, type a valid address in RAM or ROM.
- Press the Enter key. Window contents change, to display memory values starting at the address you entered.

**NOTE** You can edit the contents of the View Memory window. This window also lets you disassemble a random part of memory.

f. Close the View Memory window.

6. Run the application.
Application Tutorial
Debug the Application

a. From the main menu bar, select Project > Run, or click the Run button of the Debugger window. A console window (Figure 3.14) appears, displaying the Hello-World-message result of the application.

Figure 3.14 Console Window

b. Click the Kill button of the Debugger window. The debugger stops the application, the IDE stops the debugger, and the Debugger window closes.

c. This completes the procedure — you have created and debugged a simple application. You may close any open windows.
Target Settings

This chapter explains the settings panels specific to ColdFire software development. Use the elements of these panels to control assembling, compiling, linking, and other aspects of code generation.

NOTE For documentation of the target settings panels included in all CodeWarrior products, see the IDE User Guide.

This chapter consists of these sections:
- Target Settings Overview
- ColdFire Settings Panels

Target Settings Overview

A CodeWarrior project contains one or more build targets. A build target is a named collection of files and settings that the CodeWarrior IDE uses to generate an output file. A build target contains all build-specific target settings. Target settings define:
- The files that belong to a build target.
- The behavior of the compiler, assembler, linker, and other build tools.

The build target feature lets you create different versions of your program for different purposes. For example, you might have a debug build target. This build target would include no optimizations so it is easy to debug. You might also have a release build target. This build target would be heavily optimized so it uses less memory or runs faster.

You control target settings through target settings panels that you access through the Target Settings window.

To open this window, select Edit > Target Settings, from the main-window menu bar. (Target is a target name, such as CF_Simulator, within your CodeWarrior project.) An alternate way to bring up the Target Settings window is to bring the Targets page to the front of the project window, then double-click the project name.

Figure 4.1 shows this Target Settings window. (The CodeWarrior IDE User’s Guide explains all elements of this window.)
Target Settings
ColdFire Settings Panels

Use the tree listing of panels, in the Target Settings Panels pane, to display any settings panel. If necessary, click the expand control to see a category’s list of panels. Clicking a panel name immediately puts that panel in the Target Settings pane.

Figure 4.1 Target Settings Window

Note these buttons, at the bottom of the window:

- **Apply** — Implements your changes, leaving the **Target Settings** window open. This lets you bring up a different settings panel.
- **OK** — Implements your changes, closing the **Target Settings** window. Use this button when you make the last of your settings changes.
- **Revert** — Changes panel settings back to their most recently saved values. (Modifying any panel settings activates this button.)
- **Factory Settings** — Restores the original default values for the panel.
- **Import Panel** — Copies panel settings previously saved as an XML file.
- **Export Panel** — Saves settings of the current panel to an XML file.

ColdFire Settings Panels

*Table 4.1* lists the target settings panels specific to developing applications for the ColdFire target. The following section describes these panels in detail.
Target Settings

ColdFire Settings Panels

Table 4.1 ColdFire Target Settings Panels

<table>
<thead>
<tr>
<th>Panel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Settings</td>
<td>ColdFire Processor</td>
</tr>
<tr>
<td>BatchRunner PreLinker</td>
<td>ELF Disassembler</td>
</tr>
<tr>
<td>BatchRunner PostLinker</td>
<td>ColdFire Linker</td>
</tr>
<tr>
<td>ColdFire Target</td>
<td>Debugger PIC Settings</td>
</tr>
<tr>
<td>ColdFire Assembler</td>
<td></td>
</tr>
</tbody>
</table>

NOTE For debugger-specific panels CF Debugger Setting, CF Exceptions, Debugger Settings, and Remote Debugging, see the Debugging chapter. For information about the C/C++ Language and C/C++ Warnings panels, see the C Compilers Reference manual. For details on all other panels, see the IDE User’s Guide.

Target Settings

Use the Target Settings panel (Figure 4.2) to define the build target and select the appropriate linker. Table 4.2 explains the elements of this panel.

NOTE You must use this settings panel to select a linker before you can specify the compiler, linker settings, or any other project details.

Figure 4.2 Target Settings Panel

![Target Settings Panel](image)
# Target Settings

**ColdFire Settings Panels**

## Table 4.2 Target Settings Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Name text box</td>
<td>Specifies the name of the build target; this name appears subsequently on the Targets page of the project window.</td>
<td>Default: None. This build-target name is <em>not</em> the name of your final output file.</td>
</tr>
<tr>
<td>Linker list box</td>
<td>Specifies the linker: Select ColdFire.</td>
<td>Default: ColdFire. Controls visibility of other relevant panels.</td>
</tr>
<tr>
<td>Pre-linker list box</td>
<td>Specifies the pre-linker that performs work on object code before linking.</td>
<td>Default: None. If your project includes Flash programming, select BatchRunner PreLinker. For more information, see BatchRunner PreLinker.</td>
</tr>
<tr>
<td>Post-linker list box</td>
<td>Specifies the post-linker that performs additional work on the final executable.</td>
<td>Default: None. Post-linking often includes object code format conversion. If your project includes Flash programming, select BatchRunner PostLinker. For more information, see BatchRunner PostLinker.</td>
</tr>
<tr>
<td>Output Directory text box</td>
<td>Specifies the directory for the final linked output file. To specify a non-default directory, click the Choose button. To clear this text box, click the Clear button.</td>
<td>Default: Directory that contains the project file.</td>
</tr>
<tr>
<td>Save project entries using relative paths checkbox</td>
<td>Clear — Specifies minimal file searching; each project file must have a unique name. Checked — Specifies relative file searching; project may include two or more files that have the same name.</td>
<td>Default: Clear.</td>
</tr>
</tbody>
</table>
BatchRunner PreLinker

The BatchRunner PreLinker settings panel (Figure 4.3) lets you run a batch file before the IDE begins linking your project. To specify such a batch file, click the Choose button, then use the subsequent dialog box to navigate to and select the file. Clicking the OK button of the dialog box returns you to this panel, filling in the name of the batch file.

Figure 4.3 BatchRunner PreLinker Panel

BatchRunner PostLinker

The BatchRunner PostLinker settings panel (Figure 4.4) lets you run a batch file after the IDE builds your project. To specify such a batch file, click the Choose button, then use the subsequent dialog box to navigate to and select the file. Clicking the OK button of the dialog box returns you to this panel, filling in the name of the batch file.

Figure 4.4 BatchRunner PostLinker Panel

To pass the name of the output file as a parameter to the batch file, check the Pass linker output file as %1 parameter to batch file checkbox.
Target Settings
ColdFire Settings Panels

ColdFire Target

Use the ColdFire Target panel (Figure 4.5) to specify the type of project file and to name your final output file. Table 4.3 explains the elements of this panel. (To create alternative builds, compiling for different targets, use the __option() pre-processor function with conditional compilation.)

Figure 4.5 ColdFire Target Panel

Table 4.3 ColdFire Target Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Type list box</td>
<td>Specifies the kind of project: Application — executable project</td>
<td>Default: Application.</td>
</tr>
<tr>
<td></td>
<td>Library — static library</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shared Library — shared library</td>
<td></td>
</tr>
<tr>
<td>File Name text box</td>
<td>Specifies the name of your final linked output file.</td>
<td>Default: None. Convention: use extension.elf for an application, .lib or .a for a library.</td>
</tr>
</tbody>
</table>

ColdFire Assembler

Use the ColdFire Assembler panel (Figure 4.6) to control the source format or syntax for the CodeWarrior assembler, and to specify the target processor, for which you are generating code. Table 4.4 explains the elements of this panel.
**Figure 4.6 ColdFire Assembler Panel**

![ColdFire Assembler Panel](image)

**Table 4.4 ColdFire Assembler Panel Elements**

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor list box</td>
<td>Specifies the target processor.</td>
<td>Default: MCF52xx.</td>
</tr>
<tr>
<td>Processor has MAC</td>
<td>Clear — Tells assembler that the target processor does not have a</td>
<td>Default: Clear. You can check both the MAC and EMAC checkboxes.</td>
</tr>
<tr>
<td>Processor has EMAC</td>
<td>Clear — Tells assembler that the target processor does not have an</td>
<td>Default: Clear. You can check both the MAC and EMAC checkboxes.</td>
</tr>
<tr>
<td></td>
<td>enhanced multiply accumulator (EMAC) unit.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Checked — Tells assembler that the target processor does have a MAC.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table 4.4 ColdFire Assembler Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
</table>
| Processor has FPU checkbox | Clear — Tells assembler that the target processor does *not* have a floating-point unit (FPU).  
Checked — Tells assembler that the target processor *does* have an FPU. | Default: Clear |
| Labels Must End With `:` checkbox | Clear — System does *not* require labels to end with colons.  
Checked — System *does* require labels to end with colons. | Default: Checked. |
| Directives Begin With `:` checkbox | Clear — System does *not* require directives to start with periods.  
Checked — System *does* require directives to start with periods. | Default: Checked. |
| Case Sensitive Identifiers checkbox | Clear — Tells assembler to ignore case in identifiers.  
Checked — Tells assembler to consider case in identifiers. | Default: Checked. |
| Allow Space In Operand Field checkbox | Clear — Tells assembler to *not* allow spaces in operand fields.  
Checked — Tells assembler to *allow* spaces in operand fields. | Default: Checked. |
Table 4.4 ColdFire Assembler Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Listing File checkbox</td>
<td>Clear — Tells assembler to not generate a listing file.</td>
<td>Default: Clear. A listing file contains the file source, along with line numbers, relocation information, and macro expansions.</td>
</tr>
<tr>
<td></td>
<td>Checked — Tells assembler to generate a listing file.</td>
<td></td>
</tr>
<tr>
<td>Prefix File text box</td>
<td>Specifies the name of the assembly prefix file.</td>
<td>Default: None. Useful for include files that define common constants, global declarations, and function names. Otherwise, the assembler’s default prefix file suffices.</td>
</tr>
</tbody>
</table>

**ELF Disassembler**

Use the **ELF Disassembler** panel (Figure 4.7) to control settings for the disassembly view; you see this view when you disassemble object files. **Table 4.5** explains the elements of this panel.
Target Settings

ColdFire Settings Panels

Figure 4.7 ELF Disassembler Panel

Table 4.5 ELF Disassembler Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Headers checkbox</td>
<td>Clear — Keeps ELF header information out of the disassembled output.</td>
<td>Default: Checked.</td>
</tr>
<tr>
<td></td>
<td>Checked — Puts ELF header information into the disassembled output.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Checked — Puts additional information into the disassembled output.</td>
<td>For the .symtab section, additional information includes numeric equivalents for descriptive constants. For the .line, .debug, .extab, and .extabindex sections, additional information includes an unstructured hex dump.</td>
</tr>
<tr>
<td>Show Symbol and String Tables checkbox</td>
<td>Clear — Keeps symbol table out of the disassembled module.</td>
<td>Default: Checked.</td>
</tr>
</tbody>
</table>
### Table 4.5 ELF Disassembler Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Relocations checkbox</td>
<td>Clear — Keeps relocation information out of the disassembled module.</td>
<td>Default: Checked. Relocation information pertains to the .real.text and .reala.data sections.</td>
</tr>
<tr>
<td></td>
<td>Checked — Puts relocation information into the disassembled module.</td>
<td></td>
</tr>
<tr>
<td>Show Code Modules checkbox</td>
<td>Clear — Keeps any of the four types of ELF code sections out of the disassembled module; disables the four subordinate checkboxes. Checked — Activates the four subordinate checkboxes. For each checked subordinate checkbox, puts ELF code section into the disassembled module.</td>
<td>Default: Checked.</td>
</tr>
<tr>
<td>Use Extended Mnemonics checkbox</td>
<td>Clear — Keeps extended mnemonics out of the disassembled module.</td>
<td>Default: Checked. This checkbox is active only if the Show Code Modules checkbox is checked.</td>
</tr>
<tr>
<td></td>
<td>Checked — Puts instruction extended mnemonics into the disassembled module.</td>
<td></td>
</tr>
<tr>
<td>Show Source Code checkbox</td>
<td>Clear — Keeps source code out of the disassembled module.</td>
<td>Default: Checked. This checkbox is active only if the Show Code Modules checkbox is checked.</td>
</tr>
<tr>
<td></td>
<td>Checked — Lists source code in the disassembled module. Display is mixed mode, with line-number information from original C source code.</td>
<td></td>
</tr>
</tbody>
</table>
### Table 4.5 ELF Disassembler Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Addresses and Object Code checkbox</td>
<td>Clear — Keeps addresses and object code out of the disassembled module. Checked — Lists addresses and object code in the disassembled module.</td>
<td>Default: Checked. This checkbox is active only if the Show Code Modules checkbox is checked.</td>
</tr>
<tr>
<td>Show Comments checkbox</td>
<td>Clear — Keeps disassembler comments out of the disassembled module. Checked — Shows disassembler comments in sections that have comment columns.</td>
<td>Default: Checked. This checkbox is active only if the Show Code Modules checkbox is checked.</td>
</tr>
<tr>
<td>Show Data Modules checkbox</td>
<td>Clear — Blocks output of ELF data sections for the disassembled module; disables the Disassemble Exception Tables checkbox. Checked — Outputs <code>.rodata</code>, <code>.bss</code>, or other such ELF data sections in the disassembled module. Activates the Disassemble Exception Tables checkbox.</td>
<td>Default: Checked.</td>
</tr>
<tr>
<td>Disassemble Exception Tables checkbox</td>
<td>Clear — Keeps C++ exception tables out of the disassembled module. Checked — Includes C++ exception tables in the disassembled module.</td>
<td>Default: Clear. This checkbox is active only if the Show Data Modules checkbox is checked.</td>
</tr>
</tbody>
</table>
ColdFire Processor

Use the ColdFire Processor panel (Figure 4.8) to control code-generation settings. Table 4.6 explains the elements of this panel.

Figure 4.8 ColdFire Processor Panel

Table 4.6 ColdFire Processor Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target CPU list box</td>
<td>Specifies the target ColdFire processor.</td>
<td>Default: MCF5282.</td>
</tr>
</tbody>
</table>
| Code Model list box | Specifies access addressing for data and instructions in the object code:  
                        Smart — Relative (16-bit) for function calls in the same segment; otherwise absolute (32-bit). 
                        Near (16 bit) — Relative for all function calls. 
                        Far (32 bit) — Absolute for all function calls. | Default: Far (32 bit). Far is useful if your source file generates more than 32K of code, or if there is an out-of-range link error message. Near requires adjusting the .lcf. For .lcf information, see the ColdFire Build Tools Reference manual. |
### Table 4.6 ColdFire Processor Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Struct Alignment list box</td>
<td>Specifies record and structure alignment in memory:</td>
<td>Default: 68k 4-byte.</td>
</tr>
<tr>
<td></td>
<td>68K 2-byte — Aligns all fields on 2-byte boundaries, except for fields of only 1 byte.</td>
<td>This panel element corresponds to the <code>options align</code> pragma.</td>
</tr>
<tr>
<td></td>
<td>68K 4-byte — Aligns all fields on 4-byte boundaries.</td>
<td>Natural-boundary alignment means 1-byte for a 1-byte character, 2-bytes for a 16-bit integer, and so on.</td>
</tr>
<tr>
<td></td>
<td>PowerPC 1-byte — Aligns each field on its natural boundary.</td>
<td>NOTE: When you compile and link, alignment should be the same for all files and libraries.</td>
</tr>
<tr>
<td></td>
<td>Far (32 bit) — Storage in far data space; available memory is the only size limit.</td>
<td>This panel element corresponds the <code>far_data</code> pragma.</td>
</tr>
<tr>
<td></td>
<td>Near (16 bit) — Storage in near data space; size limit is 64K.</td>
<td></td>
</tr>
<tr>
<td>Parameter Passing list box</td>
<td>Specifies parameter-passing level:</td>
<td>Default: Compact.</td>
</tr>
<tr>
<td></td>
<td>Compact — Passes on even-sized boundary for parameters smaller than int (2 for short and char).</td>
<td>These levels correspond to the <code>compact_abi</code>, <code>standard_abi</code>, and <code>register_abi</code> pragmas.</td>
</tr>
<tr>
<td></td>
<td>Standard — Like compact, but always padded to 4 bytes.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Register — Passes in scratch registers D0 — D2 for integers, A0 — A1 for pointers and fp0 — fp1</td>
<td>NOTE: Be sure that all called functions have prototypes.</td>
</tr>
<tr>
<td></td>
<td>when FPU codegen is selected; this can speed up programs that have many small functions.</td>
<td>When you compile and link, parameter passing should be the same for all files and libraries.</td>
</tr>
</tbody>
</table>
### Table 4.6 ColdFire Processor Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Floating Point list box</td>
<td>Specifies handling method for floating-point operations:</td>
<td>Default: Software. For software selection, your project must include the appropriate FP_ColdFire C runtime library file. Greyed out if your target processor lacks an FPU.</td>
</tr>
<tr>
<td></td>
<td>Software — C runtime library code emulates floating-point operations.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hardware — Processor hardware performs floating point operations; only appropriate for processors that have floating-point units.</td>
<td></td>
</tr>
<tr>
<td>4-Byte Integers checkbox</td>
<td>Clear — Specifies 2-byte integers.-checked — Specifies 4-byte integers.</td>
<td>Default: Checked.</td>
</tr>
<tr>
<td>A6 Stack Frames checkbox</td>
<td>Clear — Disables call-stack tracing; generates faster and smaller code.</td>
<td>Default: Checked. Checking this checkbox corresponds to using the a6frames pragma.</td>
</tr>
<tr>
<td></td>
<td>Checked — Enables call-stack tracing; each stack frame sets up and restores register A6.</td>
<td></td>
</tr>
<tr>
<td>Emit Macsbug Symbols checkbox</td>
<td>Clear — Does not generate Macsbug symbols. Checked — Generates Macsbug symbols inside code after RTS statements.</td>
<td>Default: Clear. A Macsbug symbol is the routine name, appended after the routine, in Pascal format. These symbols are appropriate only for older debuggers.</td>
</tr>
</tbody>
</table>
ColdFire Linker

Use the ColdFire Linker panel (Figure 4.9) to control the final form of your object code. Table 4.7 explains the elements of this panel.
Figure 4.9  ColdFire Linker Panel

Table 4.7  ColdFire Linker Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store Full Path Names checkbox</td>
<td>Clear — In debugging information in the linked ELF file, uses only names of source files. Checked — Includes source-file paths in the debugging information in the linked ELF file. Default: Checked. Clearing this checkbox saves target memory, but increases the time the debugger needs to find the source files.</td>
<td></td>
</tr>
</tbody>
</table>
## Target Settings

*ColdFire Settings Panels*

Table 4.7 ColdFire Linker Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Link Map checkbox</td>
<td>Clear — Does <em>not</em> generate a link map.</td>
<td>Default: Checked. A link map includes addresses of all objects and functions, a memory map of sections, and values of symbols the linker generates. A link map has the same filename as the output file, but with extension .xMAP.</td>
</tr>
<tr>
<td></td>
<td>Checked — Does generate a link map (a text file that identifies definition files for each object and function of your output file); activates the List Unused Objects and Show Transitive Closure checkboxes.</td>
<td></td>
</tr>
<tr>
<td>List Unused Objects checkbox</td>
<td>Clear — Does <em>not</em> include unused objects in the link map.</td>
<td>Default: Clear. This checkbox is active only if the Generate Link Map checkbox is checked. NOTE: The linker never deadstrips unused assembler relocatables or relocatables built with a non-CodeWarrior compiler. But checking this checkbox gives you a list of such unused items; you can use this list to remove the symbols.</td>
</tr>
<tr>
<td></td>
<td>Checked — Does include unused objects in the link map.</td>
<td></td>
</tr>
<tr>
<td>Show Transitive Closure checkbox</td>
<td>Clear — Does <em>not</em> include the link map objects that main() references.</td>
<td>Default: Checked. This checkbox is active only if the Generate Link Map checkbox is checked. Listings after this table show the effect of this checkbox.</td>
</tr>
<tr>
<td></td>
<td>Checked — Recursively lists in the link map all objects that main() references.</td>
<td></td>
</tr>
<tr>
<td>Disable Deadstripping checkbox</td>
<td>Clear — Lets linker remove unused code and data.</td>
<td>Default: Clear.</td>
</tr>
<tr>
<td></td>
<td>Checked — Prevents the linker from removing unused code or data.</td>
<td></td>
</tr>
<tr>
<td>Element</td>
<td>Purpose</td>
<td>Comments</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Generate ELF Symbol Table</td>
<td>Clear — Omits the ELF symbol table and relocation list from the ELF output file.</td>
<td>Default: Checked.</td>
</tr>
<tr>
<td>checkbox</td>
<td>Checked — Includes an ELF symbol table and relocation list in the ELF output file.</td>
<td></td>
</tr>
<tr>
<td>checkbox</td>
<td>Checked — Reports only fatal warning messages; does not affect display of messages from other parts of the IDE.</td>
<td></td>
</tr>
<tr>
<td>checkbox</td>
<td>Checked — Generates an S3-type S-record file, suitable for printing or transportation to another computer system. Activates the Max S-Record text box and the EOL character list box.</td>
<td>The S-record has the same filename as the executable file, but with extension .S19, .S3 records include code, data, and their 4-byte addresses.</td>
</tr>
<tr>
<td>Max S-Record Length text box</td>
<td>Specifies maximum number of bytes in S-record lines that the linker generates. The maximum value for this text box is 252.</td>
<td>Default: 80. This text box is active only if the Generate S-Record File checkbox is checked. Note: Many embedded systems limit S-record lines to 24 or 26 bytes. A value of 20 to 30 bytes lets you see the S-record on a single page.</td>
</tr>
<tr>
<td>EOL Character list box</td>
<td>Specifies the end-of-line character for the S-record file, by operating system: DOS, UNIX, or MAC.</td>
<td>Default: DOS. This text box is active only if the Generate S-Record File checkbox is checked.</td>
</tr>
</tbody>
</table>
Target Settings
ColdFire Settings Panels

Table 4.7 ColdFire Linker Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generate Binary Image checkbox</td>
<td>Clear — Does not create a binary version of the S-record file. Checked — Saves a binary version of the S-record file to the project folder; The binary file has the .b filename extension. Activates the Max Bin Record Length text box.</td>
<td>Default: Clear. Binary file format is address (4 bytes), byte count (4 bytes), and data bytes (variable length).</td>
</tr>
<tr>
<td>Max Bin Record Length text box</td>
<td>Specifies data-byte length for each binary record. The maximum value is 252.</td>
<td>Default: None. This text box is active only if the Generate Binary Image checkbox is checked.</td>
</tr>
<tr>
<td>Entry Point text box</td>
<td>Specifies the program starting point: the first function the debugger uses upon program start.</td>
<td>Default: __start. (This default function is in file ColdFire__startup.c. It sets up the ColdFire EABI environment before code execution. Its final task is calling main().)</td>
</tr>
<tr>
<td>Force Active Symbols text box</td>
<td>Specifies symbols to be included in the output file even if not referenced; makes such symbols immune from deadstripping.</td>
<td>Default: None. Use spaces to separate multiples symbols.</td>
</tr>
</tbody>
</table>

Listing 4.1 and Listing 4.2 show the effect of the Show Transitive Closure checkbox.

Listing 4.1 Sample Code for Transitive Closure

```c
void alpha1()
{
    int a = 1001;
}
void alpha()
{
    int b = 1002;
    alpha1();
}
int main(void)
{
    alpha();

```
return 1;
}

If you checked the **Show Transitive Closure** checkbox of the **ColdFire Linker** panel and compiled the source files,

- The linker would generate a link map file, and
- The link map file would include text such as that of **Listing 4.2**.

**Listing 4.2  Link Map: Effects of Show Transitive Closure**

```markdown
# Link map of __start
1] __start (func, global) found in C_4i_CF_Runtime.a E68k_startup.o
2] __main (func, global) found in main.c
3] __alpha (func, global) found in main.c
4] __alpha1 (func, global) found in main.c
```

**Debugger PIC Settings**

Use the **Debugger PIC Settings** panel (**Figure 4.10**) to specify an alternate address where you want your ELF image downloaded on the target.

**Figure 4.10  Debugger PIC Settings Panel**

![Debugger PIC Settings Panel](image)

Usually, Position Independent Code (PIC) is linked so that the entire image starts at address 0x00000000. To specify a different target address for loading the PIC module:

1. Check the **Alternate Load Address** checkbox — this activates the text box.
2. Enter the address in the text box.

At download time, the debugger downloads your ELF file to this new address of the target.

**NOTE**  The debugger does not verify that your code can execute at the new address. However, the PIC generation settings of the compiler and linker, and the startup routines of your code, correctly set any base registers and perform any appropriate relocations.
Debugging

This chapter explains aspects of debugging that are specific to the ColdFire architectures. For more general information about the CodeWarrior debugger, see the IDE 5.7 User’s Guide.

To start the CodeWarrior debugger, select Project > Debug. The debugger window appears; the debugger loads the image file that the current build target produces. You can use the debugger to control program execution, insert breakpoints, and examine memory and registers.

NOTE The automatic loading of the previous paragraph depends on the load options you specify, and on whether your application code is in ROM or Flash memory.

This chapter consists of these sections:

- Target Settings for Debugging
- Remote Connections for Debugging
- BDM Debugging
- Debugging ELF Files without Projects
- Special Debugger Features

Target Settings for Debugging

Several target settings panels control the way the debugger works:

- CF Debugger Settings Panel
- Remote Debugging Panel
- CF Exceptions Panel
- Debugger Settings Panel
- CF Interrupt Panel

To access these panels, select Edit > Target Settings, from the main menu bar. (Target is the current build target in the CodeWarrior project.) The Target Settings window (Figure 5.1) appears.
Debugging
Target Settings for Debugging

Figure 5.1 Target Settings Window

Table 5.1 lists additional panels that can affect debugging.

Table 5.1 Additional Settings Panels That May Affect Debugging

<table>
<thead>
<tr>
<th>Panel</th>
<th>Impact</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ Warnings</td>
<td>compiler warnings</td>
<td>C Compilers Reference</td>
</tr>
<tr>
<td>ColdFire Linker</td>
<td>controls symbolics, linker warnings</td>
<td>ColdFire Linker</td>
</tr>
<tr>
<td>ColdFire Processor</td>
<td>optimizations</td>
<td>ColdFire Processor</td>
</tr>
<tr>
<td>Global Optimizations</td>
<td>optimizations</td>
<td>IDE User’s Guide</td>
</tr>
</tbody>
</table>

CF Debugger Settings Panel

Use the CF Debugger Settings panel (Figure 5.2) to select debugger hardware and control interaction with the target board. Table 5.2 explains the elements of this panel.
Debugging
Target Settings for Debugging

Figure 5.2 CF Debugger Settings Panel

Table 5.2 CF Debugger Settings Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Processor list box</td>
<td>Specifies the target processor.</td>
<td>Your stationery selection automatically makes this specification.</td>
</tr>
<tr>
<td>Target OS list box</td>
<td>Specifies a real-time operating system; for bare board development, select BareBoard.</td>
<td>Default: BareBoard. If you have Professional-Edition software and install an RTOS, that RTOS becomes a selection of this list box. (Special- and Standard-Edition software, however, does not support an RTOS.)</td>
</tr>
<tr>
<td>Use Target Initialization File checkbox</td>
<td>Clear — Specifies not using a target initialization file; deactivates file subordinate text box and Browse button. Checked — Tells the debugger to use the specified target initialization file. To enter a pathname in the text box, click the Browse button, then use the file-select dialog box to specify the file.</td>
<td>Default: checked. The initialization file is in subdirectory &quot;\E68K_Support\Initialization_Files&quot;, of the CodeWarrior installation directory (or directory that contains your project). Clear this checkbox, if you are using an Abatron-based remote connection. Make sure this checkbox is checked, if you are using a P&amp;E Micro-based remote connection.</td>
</tr>
</tbody>
</table>
### Debugging

**Target Settings for Debugging**

Table 5.2  CF Debugger Settings Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use Memory Configuration File checkbox</td>
<td>Clear — Specifies <em>not</em> using a memory configuration file; deactivates file subordinate text box and Browse button. Checked — Tells the debugger to use the specified memory configuration file. To enter a pathname in the text box, click the Browse button, then use the file-select dialog box to specify the file.</td>
<td>Default: Unchecked. The memory configuration file is in subdirectory <code>\E68K_Support\Initialization_Files</code>, of the CodeWarrior installation directory (or directory that contains your project). Do not check this checkbox, if you are using a Abatron-based remote connection. Check this checkbox, if you are using a P&amp;E Micro-based remote connection.</td>
</tr>
<tr>
<td>Initial Launch: Executable checkbox</td>
<td>Clear — Does <em>not</em> download program executable code or text sections for initial launch. Checked — Downloads program executable code and text sections for initial launch.</td>
<td>Default: Checked. Initial launch is the first time you debug the project after you start the debugger from the IDE.</td>
</tr>
<tr>
<td>Initial Launch: Constant Data checkbox</td>
<td>Clear — Does <em>not</em> download program constant data sections for initial launch. Checked — Downloads program constant data sections for initial launch.</td>
<td>Default: Checked. Initial launch is the first time you debug the project after you start the debugger from the IDE.</td>
</tr>
<tr>
<td>Initial Launch: Initialized Data checkbox</td>
<td>Clear — Does <em>not</em> download program initialized data sections for initial launch. Checked — Downloads program initialized data sections for initial launch.</td>
<td>Default: Checked. Initial launch is the first time you debug the project after you start the debugger from the IDE.</td>
</tr>
<tr>
<td>Initial Launch: Uninitialized Data checkbox</td>
<td>Clear — Does <em>not</em> download program uninitialized data sections for initial launch. Checked — Downloads program uninitialized data sections for initial launch.</td>
<td>Default: Clear. Initial launch is the first time you debug the project after you start the debugger from the IDE.</td>
</tr>
</tbody>
</table>
Remote Debugging Panel

Use the Remote Debugging panel (Figure 5.3) to set up connections for remote debugging. Table 5.3 explains the elements of this panel. Text following the figure and table provides more information about adding and changing remote connections.

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successive Runs: Executable checkbox</td>
<td>Clear — Does not download program executable code or text sections for successive runs. Checked — Downloads program executable code and text sections for successive runs.</td>
<td>Default: Clear. Successive runs are debugging actions after initial launch. Note that rebuilding the project returns you to the initial-launch state.</td>
</tr>
<tr>
<td>Successive Runs: Constant Data checkbox</td>
<td>Clear — Does not download program constant data sections for successive runs. Checked — Downloads program constant data sections for successive runs.</td>
<td>Default: Clear. Successive runs are debugging actions after initial launch. Note that rebuilding the project returns you to the initial-launch state. NOTE: If you check this checkbox, avoid cycling board power. Doing so can prevent application rebuilding and code reloading, making debugging unnecessarily difficult.</td>
</tr>
<tr>
<td>Successive Runs: Initialized Data checkbox</td>
<td>Clear — Does not download program initialized data sections for successive runs. Checked — Downloads program initialized data sections for successive runs.</td>
<td>Default: Checked. Successive runs are debugging actions after initial launch. Note that rebuilding the project returns you to the initial-launch state.</td>
</tr>
<tr>
<td>Successive Runs: Uninitialized Data checkbox</td>
<td>Clear — Does not download program uninitialized data sections for successive runs. Checked — Downloads program uninitialized data sections for successive runs.</td>
<td>Default: Checked. Successive runs are debugging actions after initial launch. Note that rebuilding the project returns you to the initial-launch state.</td>
</tr>
<tr>
<td>Verify Memory Writes checkbox</td>
<td>Clear — Does not confirm that a section written to the target matches the original section. Checked — Confirms that any section written to the target matches the original section.</td>
<td>Default: Clear.</td>
</tr>
</tbody>
</table>
NOTE  Special-Edition software supports the following remote connections: P&E Microsystems USB, P&E Microsystems Cyclone Max, and Freescale USB TAP. Standard-Edition software supports only P&E Microsystems USB remote connection. For any other type of remote connection, you must have Professional-Edition software.

Figure 5.3  Remote Debugging Panel

Table 5.3  Remote Debugging Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection list box</td>
<td>Specifies the remote-connection type: the remote debugger, along with its default settings.</td>
<td>Possible remote connections include Abatron Serial or TCP/IP; CCS-SIM; and P&amp;E Microsystems Parallel, USB, Cyclone Max, and Lightning, and Freescale USB TAP. However, you must add any such additional connection before it is available in this list box.</td>
</tr>
<tr>
<td>Edit Connection button</td>
<td>Starts process of adding a remote connection, or changing settings of an existing remote connection.</td>
<td>For instructions, see text after this table.</td>
</tr>
</tbody>
</table>
Table 5.3 Remote Debugging Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote download text box</td>
<td>Specifies the absolute path to the directory in which to store downloaded files. This option does not apply to bareboard development.</td>
<td>Default: None.</td>
</tr>
<tr>
<td>Launch remote host application checkbox</td>
<td>Clear — Prevents the IDE from starting a host application on the remote computer. Checked — IDE starts a host application on the remote computer. (Also enables the corresponding text box, for the absolute path to the remote host application.) This option does not apply to bareboard development.</td>
<td>Default: Clear</td>
</tr>
<tr>
<td>Download OS checkbox</td>
<td>Clear — Prevents downloading a bootable image to the target system. Checked — Downloads the specified bootable image to the target system. (Also enables the Connection list box and OS Image Path text box.)</td>
<td>Default: Clear</td>
</tr>
<tr>
<td>Connection list box</td>
<td>Specifies the remote-connection type for downloading the bootable image to the target board.</td>
<td>Disabled if the Download OS checkbox is clear. Lists only the remote connections you add via the Remote Connections panel.</td>
</tr>
<tr>
<td>OS Image path</td>
<td>Specifies the host-side path of the bootable image to be downloaded to the target board.</td>
<td>Disabled if the Download OS checkbox is clear.</td>
</tr>
</tbody>
</table>

Adding Remote Connections

**NOTE** Special-Edition software supports the following remote connections: P&E Microsystems USB, P&E Microsystems Cyclone Max, and Freescale USB TAP. Standard-Edition software supports only P&E Microsystems USB remote connection. For any other type of remote connection, you must have Professional-Edition software.
Debugging
Target Settings for Debugging

To add a remote connection, use the Remote Connections panel:

1. Select Edit > Preferences. The IDE Preferences window appears.

2. From the IDE Preferences Panels list, select Remote Connections. The Remote Connections panel moves to the front of the IDE Preferences window. Figure 5.4 shows the IDE Preferences window at this point.

Figure 5.4 IDE Preferences Window: Remote Connections Panel

3. Click the Add button. The New Connection dialog box (Figure 5.5) appears.

Figure 5.5 New Connection Dialog Box

4. In the Name text box, enter a name for the new connection.
5. Use the Debugger list box to specify the debugger for the new remote connection: ColdFire Abatron, ColdFire P&E Micro, or ColdFire CCS (the simulator and USB TAP).

6. Check the Show in processes list checkbox to add this new connection to the official list. (To see this list of processes, select View > Systems > List.) Checking this checkbox also adds this new connection to the remote-connection list that pops up when you debug certain kinds of files.)

7. Use the Connection Type list box to specify the type — the remaining fields of the dialog box change appropriately.

8. Use the remaining fields of the New Connection dialog box to make any appropriate changes to the default values for connection type, port, rate, and so forth.

9. Click OK. The dialog box closes; the Remote Connections panel window displays the new connection.

10. This completes adding the new connection. You may close the IDE Preferences window.

**Changing Remote Connections**

To change to an already-configured remote connection, use the Remote Debugging panel (Figure 5.3):

1. Click the arrow symbol of the Connection list box. The list of connections appears.

2. Select another connection. The list collapses; the list box displays your selection.

3. Click the Edit Connections button. A dialog box appears, showing the configuration of the remote connection.

4. Use the dialog box to make any appropriate configuration changes.

5. Click OK. The dialog box closes, confirming your configuration changes.

**NOTE** Any changes you make using the Remote Debugging panel apply to all targets that use the specified connection.

**CF Exceptions Panel**

The CF Exceptions panel (Figure 5.6) is available with P&E Microsystems, simulator, and Freescale USB TAP remote connections. Use this panel to specify hardware exceptions that the debugger should catch. Table 5.4 explains the elements of this panel.

Before you load and run the program, the debugger inserts its own exception vector for each exception you check in this panel. To use your own exception vectors instead, you should clear the corresponding checkboxes.
Debugging
Target Settings for Debugging

If you check any boxes, the debugger reads the Vector_Based_Register (VBR), finds the corresponding existing exception vector, then writes a new vector at that register location. The address of this new vector is offset 0x408 from the VBR address. For example, if the VBR address is 0x0000 0000, the new vector at address 0x0000 0408 covers the checked exceptions.

The debugger writes a Halt instruction and a Return from Exception instruction at this same location.

NOTE If your exceptions are in Flash or ROM, do not check any boxes of the CF Exceptions panel. Abatron remote connections ignore this panel, using instead the exception definitions in the Abatron firmware.

Figure 5.6 CF Exceptions Panel

<table>
<thead>
<tr>
<th>CF Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exception Handling (checked exceptions will have their vectors overwritten to allow a debugger break when they occur)</td>
</tr>
<tr>
<td>2 Access Error</td>
</tr>
<tr>
<td>3 Address Error</td>
</tr>
<tr>
<td>4 Illegal Instruction</td>
</tr>
<tr>
<td>5 Divide By Zero</td>
</tr>
<tr>
<td>6 Privilege Violation</td>
</tr>
<tr>
<td>7 Trap</td>
</tr>
<tr>
<td>8 Unimplemented Instruction A Opcode</td>
</tr>
<tr>
<td>9 Unimplemented Instruction B Opcode</td>
</tr>
<tr>
<td>10 Unimplemented Instruction C Opcode</td>
</tr>
<tr>
<td>11 Unimplemented Instruction D Opcode</td>
</tr>
<tr>
<td>12 Non-FC breakpoint debug interrupt</td>
</tr>
<tr>
<td>13 PC breakpoint debug interrupt</td>
</tr>
<tr>
<td>14 Format Error</td>
</tr>
<tr>
<td>15 Uninitialized Interrupt</td>
</tr>
<tr>
<td>24 Spurious Interrupt</td>
</tr>
<tr>
<td>31 IRQ#? - break button</td>
</tr>
<tr>
<td>46 TRAP 114 for Console I/O</td>
</tr>
<tr>
<td>61 Unsupported Instruction</td>
</tr>
</tbody>
</table>

* - If unselected these may effect the debugger's ability to control the target

Table 5.4 CF Exceptions Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Access Error checkbox</td>
<td>Clear — Ignores access errors. Checked — Catches and displays access errors.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>3 Address Error checkbox</td>
<td>Clear — Ignores address errors. Checked — Catches and displays address errors.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>4 Illegal Instruction checkbox</td>
<td>Clear — Ignores invalid instructions. Checked — Catches and displays invalid instructions.</td>
<td>Default: Checked</td>
</tr>
</tbody>
</table>
### Table 5.4 CF Exceptions Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 Divide by zero checkbox</td>
<td>Clear — Ignores an attempt to divide by zero. Checked — Catches and displays any attempt to divide by zero.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>8 Privilege Violation checkbox</td>
<td>Clear — Ignores privilege violations. Checked — Catches and displays privilege violations.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>12 Non-PC breakpoint debug interrupt checkbox</td>
<td>Clear — Ignores non-PC breakpoint debug interrupts. Checked — Catches and displays non-PC breakpoint debug interrupts.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>13 PC breakpoint debug interrupt checkbox</td>
<td>Clear — Ignores PC breakpoint debug interrupts. Checked — Catches and displays PC breakpoint debug interrupts.</td>
<td>Default: Clear</td>
</tr>
<tr>
<td>14 Format Error checkbox</td>
<td>Clear — Ignores format errors. Checked — Catches and displays format errors.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>15 Uninitialized Interrupt checkbox</td>
<td>Clear — Ignores uninitialized interrupts. Checked — Catches and displays uninitialized interrupts.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>24 Spurious Interrupt checkbox</td>
<td>Clear — Ignores spurious interrupts. Checked — Catches and displays spurious interrupts.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>31 IRQ7 - break button checkbox</td>
<td>Clear — Ignores use of the IRQ7 break button. Checked — Catches and displays uses of the IRQ7 break button.</td>
<td>Default: Checked</td>
</tr>
</tbody>
</table>
Debugging
Target Settings for Debugging

Table 5.4 CF Exceptions Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>46 TRAP #14 for Console I/O checkbox</td>
<td>Clear — Ignores trap 14 for console I/O. Checked — Catches and displays uses of trap 14 for console I/O.</td>
<td>Default: Clear.</td>
</tr>
</tbody>
</table>

Debugger Settings Panel

Use the Debugger Settings panel (Figure 5.7) to select and control the debug agent. Table 5.5 explains the elements of this panel.

Figure 5.7 Debugger Settings Panel
### Table 5.5 Debugger Settings Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location of Relocated Libraries and Code Resources text box</td>
<td>Specifies the pathname of libraries or other resources related to the project. Type the pathname into this text box. Alternatively, click the Choose button, then use the subsequent dialog box to specify the pathname.</td>
<td>Default: None</td>
</tr>
<tr>
<td>Stop on application launch checkbox</td>
<td>Clear — Does not specify any debugging entry point; deactivates the subordinate options buttons and text box. Checked — Specifies the debugging entry point, via a subordinate option button: Program entry point, Default language entry point, or User specified.</td>
<td>Default: Checked, with Default language entry point option button selected. If you select the User specified option button, type the entry point in the corresponding text box.</td>
</tr>
<tr>
<td>Log System Messages checkbox</td>
<td>Clear — Does not log system messages. Checked — Logs system messages.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>Update data every checkbox</td>
<td>Clear — Does not update data; deactivates the subordinate text box. Checked — Regularly updates data; enter the number of seconds in the subordinate text box.</td>
<td>Default: Clear</td>
</tr>
<tr>
<td>Cache symbolics between runs checkbox</td>
<td>Clear — Does not store symbolic values in cache memory between runs. Checked — After each run, stores symbolic values in cache memory.</td>
<td>Default: Checked</td>
</tr>
</tbody>
</table>
Debugging
Target Settings for Debugging

Table 5.5  Debugger Settings Panel Elements (continued)

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop at Watchpoints checkbox</td>
<td>Check — Does not stop at watchpoints. Checked — Stops at watchpoints.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>Console Encoding list box</td>
<td>Specifies the type of console encoding.</td>
<td>Default: None</td>
</tr>
</tbody>
</table>

CF Interrupt Panel

Debugging an application involves single-stepping through code. But if you do not modify interrupts that are part of normal code execution, the debugger could jump to interrupt-handler code, instead of stepping to the next instruction.

So before you start debugging, you must mask some interrupt levels, according to your processor. To do so, use the CF Interrupt panel (Figure 5.8): Table 5.6 explains the elements of this panel.

Figure 5.8  CF Interrupt Panel

![CF Interrupt Panel](image)

Table 5.6  CF Interrupt Panel Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask Interrupts</td>
<td>Clear — Ignores interrupts. Checked — Masks interrupts of the specified and lower levels, but allows higher-level interrupts.</td>
<td>Default: Clear.</td>
</tr>
<tr>
<td>Interrupt Level list box</td>
<td>Specifies the interrupt level, from 0 (low) to 7 (high).</td>
<td>Default: 0.</td>
</tr>
</tbody>
</table>
NOTE  The exact definitions of interrupt levels are different for each target processor, and masking all interrupts can cause inappropriate processor behavior. This means that finding the best interrupt level to mask can involve trial and error.

Be alert for any code statements that change the interrupt mask: stepping over such a statement can modify your settings in this panel.

Remote Connections for Debugging

To debug an application on the remote target system, you must use a remote connection.

A remote connection is the physical connection from the host to the target board, together with the settings that describe how the CodeWarrior IDE should connect to and control program execution on target boards or systems. The remote connection includes the debugger protocol, connection type, and connection parameters the IDE should use when it connects to the target system. This section shows you how to access remote connections in the CodeWarrior IDE, and describes the various debugger protocols and connection types the IDE supports.

NOTE  We have included several types of remote connections in the default CodeWarrior installation. You can modify these default remote connections to suit your particular needs.

TIP  When you import a Makefile into the CodeWarrior IDE to create a CodeWarrior project, the IDE asks you to specify the type of debugger interface (remote connection) you want to use. To debug the generated CodeWarrior project, you must properly configure the remote connection you selected when you created the project.

For Special- and Standard-Edition software, the ColdFire debugger uses a plug-in architecture to support the P&E Microsystems Parallel and USB remote-connection protocols.

For Professional-Edition software, the ColdFire debugger uses a plug-in architecture to support any of these remote-connection protocols:

- Abatron Serial
- Abatron TCP-IP
- Freescale USB-TAP
- P&E Microsystems Parallel
- P&E Microsystems USB
Debugging
Remote Connections for Debugging

- P&E Microsystems Lightning
- P&E Microsystems Cyclone MAX Serial
- P&E Microsystems Cyclone MAX TCP-IP
- P&E Microsystems Cyclone MAX USB
- Simulator (CCS-SIM)

NOTE In addition to the protocols mentioned above, Code Warrior for ColdFire now also allows use of additional run control devices through the GDI protocol. For hardware interfaces which use the GDI protocol, please refer to the user’s manual of that particular interface for installation procedures. As interfaces which use the GDI protocol make use of proprietary DLL files, Freescale does not offer support for capabilities of a particular interface or guarantee functionality. Please contact your interface supplier for any support questions.

Before you debug a project, you must configure or modify the settings of your remote-connection protocol. Follow these steps:

1. From the main menu bar, select Edit > Target Settings. The Target Settings window appears.

2. Select Target Settings Panels > Debugger > Remote Debugging. The Remote Debugging panel moves to the front of the window.

3. Use the Connection list box to specify a remote connection. The supported list of remote connections is shown in Figure 5.9.
4. Click the **Edit Connection** button. A corresponding remote connection dialog box appears.

5. Use the dialog box to input communication settings, according to text below.

**Abatron Remote Connections**

Figure 5.10 shows the configuration dialog box for an Abatron *serial* remote connection. Figure 5.11 shows the configuration dialog box for an Abatron *TCP/IP* remote connection. Table 5.6 explains the elements of these dialog boxes.
Debugging
Remote Connections for Debugging

Figure 5.11  TCP/IP Abatron Remote-Connection Dialog Box

Table 5.7  Abatron Dialog-Box Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name text box</td>
<td>Identifies the remote connection. For an existing connection, already has a value.</td>
<td></td>
</tr>
<tr>
<td>Debugger list box</td>
<td>Identifies the debugger. For an existing connection, already specifies ColdFire Abatron</td>
<td></td>
</tr>
<tr>
<td>Show in processes list checkbox</td>
<td>Clear — Leaves the connection off the official list. Checked — Adds connection to the official list (select View &gt; Systems &gt; List); also adds connection to the pop-up list for debugging certain kinds of file.</td>
<td>Default: Clear</td>
</tr>
<tr>
<td>Connection Type list box</td>
<td>Specifies serial or TCP/IP. Changing this value changes the subordinate elements of the dialog box, as Figure 5.10 and Figure 5.11 show.</td>
<td></td>
</tr>
<tr>
<td>Port list box</td>
<td>Specifies the serial port: COM1, COM2, COM3, ... or COM256. Default: COM1.</td>
<td></td>
</tr>
<tr>
<td>Rate list box</td>
<td>Specifies transfer speed: 300, 1200, 2400, 9600, 9,200, 38,400, 57,600, 115,200, or 230,400 baud. Default: 38,400 baud.</td>
<td></td>
</tr>
<tr>
<td>Data Bits list box</td>
<td>Specifies number of data bits per character: 4, 5, 6, 7, or 8. Default: 8</td>
<td></td>
</tr>
</tbody>
</table>
NOTE For an Abatron remote connection, be sure to clear the checkboxes Use Target Initialization File and Use Memory Configuration File, of the CF Debugger Settings panel.

### Freescale Remote Connections

Figure 5.12 shows the dialog box that appears for Freescale’s USB TAP remote connection when the Connection Type list box is set to USB-TAP BDM. Figure 5.13 shows the dialog box that appears when the Connection Type list box is set to CCS Remote Connection. Table 5.8 explains the elements of these dialog boxes.
Debugging
Remote Connections for Debugging

Figure 5.12 USB-TAP BDM

![USB-TAP BDM Diagram]

Figure 5.13 CCS Remote Connection

![CCS Remote Connection Diagram]
### Table 5.8 USB TAP Dialog Box Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name text box</td>
<td>Identifies the remote connection.</td>
<td>For an existing connection, already has a value.</td>
</tr>
<tr>
<td>Debugger list box</td>
<td>Identifies the debugger.</td>
<td>For an existing connection, already specifies ColdFire CCS</td>
</tr>
<tr>
<td>Show in processes list checkbox</td>
<td>Clear — Leaves the connection off the official list. Checked — Adds connection to the official list (select View &gt; Systems &gt; List); also adds connection to the pop-up list for debugging certain kinds of file.</td>
<td>Default: Clear</td>
</tr>
<tr>
<td>Connection Type list box</td>
<td>Specifies USB-TAP BDM or CCS remote connection. Changing this value changes the other elements of this dialog box, as in Figure 5.12 and Figure 5.13.</td>
<td>Default: USB-TAP BDM</td>
</tr>
<tr>
<td>Use Default Serial Number (in USB-TAP BDM dialog box)</td>
<td>Each USB TAP device has a serial number (SN) burned into flash memory. The SN enables using multiple USB TAP devices during a single debugging session. Normally is checked.</td>
<td>Default: Checked.</td>
</tr>
<tr>
<td>USB TAP Serial Number (hex) (in USB-TAP BDM dialog box)</td>
<td>If the “Use Default Serial Number” checkbox is checked, enter the serial number here. Should be same SN burned into flash on the device.</td>
<td>Default: Grayed out and blank.</td>
</tr>
<tr>
<td>CCS Timeout</td>
<td>Timeout (in seconds) that CCS waits for a reply from the target before retrying.</td>
<td>Default: 10</td>
</tr>
<tr>
<td>Interface Clock Frequency (in USB-TAP BDM dialog box)</td>
<td>Clock speed of the BDM or JTAG interface.</td>
<td>Default: 5.12MHz</td>
</tr>
<tr>
<td>Use Remote CCS (in CCS Remote Connection dialog box)</td>
<td>If checked, CCS uses Server IP Address to communicate with remote USB TAP (via TCP-IP).</td>
<td>Default: Not checked</td>
</tr>
</tbody>
</table>
Table 5.8 USB TAP Dialog Box Elements (continued)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server IP Address</td>
<td>IP address of the CCS server or debug interface device, to be used if Use Remote CCS checkbox is checked.</td>
<td>Default: Grayed out, with value: 127.0.0.1</td>
</tr>
<tr>
<td>Specify CCS Executable</td>
<td>If checked, then enter the path and file name of the CCS executable you’d like to use.</td>
<td>Default: Not checked</td>
</tr>
<tr>
<td>Multi-Core Debugging</td>
<td>If checked, then enables multi-core debugging support.</td>
<td>Default: Not checked. In ColdFire debugging, not used.</td>
</tr>
<tr>
<td>JTAG Configuration File</td>
<td>If Multi-Core Debugging is checked, then this field can be used to specify a desired JTAG configuration file.</td>
<td>Default: Grayed out. In ColdFire debugging, not used.</td>
</tr>
<tr>
<td>Reset Target on Launch</td>
<td>Determines whether target board (and processor) is reset when debugging is initiated.</td>
<td>Default: Checked</td>
</tr>
<tr>
<td>Enable Logging</td>
<td>If checked, logs CCS communications to log window or file.</td>
<td>Default: Not checked</td>
</tr>
</tbody>
</table>

P&E Microsystems Remote Connections

Figure 5.14, Figure 5.15, Figure 5.16, Figure 5.17, Figure 5.18, and Figure 5.19 show the configuration dialog boxes for PE Micro remote connections. Table 5.9 explains the elements of these dialog boxes.
Figure 5.14 P&E Micro Remote Connection (Parallel)

Figure 5.15 P&E Micro Remote Connection (USB)

Figure 5.16 P&E Micro Remote Connection (Lightning)
Debugging
Remote Connections for Debugging

Figure 5.17 P&E Micro Remote Connection (Cyclone Max Serial)

Figure 5.18 P&E Micro Remote Connection (Cyclone Max USB)

Figure 5.19 P&E Micro Remote Connection (Cyclone Max TCPIP)
## Table 5.9 P&E Micro Dialog Box Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name text box</td>
<td>Identifies the remote connection.</td>
<td>For an existing connection, already has a value.</td>
</tr>
<tr>
<td>Debugger list box</td>
<td>Identifies the debugger.</td>
<td>For an existing connection, already specifies ColdFire P&amp;E Micro</td>
</tr>
<tr>
<td>Show in processes list checkbox</td>
<td>Clear — Leaves the connection off the official list. Checked — Adds connection to the official list (select View &gt; Systems &gt; List); also adds connection to the pop-up list for debugging certain kinds of file.</td>
<td>Default: Clear</td>
</tr>
<tr>
<td>Connection Type list box</td>
<td>Specifies Parallel, USB, Lightning, Cyclone Max Serial, Cyclone Max USB, or Cyclone Max TCPIP.</td>
<td>Changing this value changes the subordinate elements of the dialog box, as Figure 5.14, Figure 5.15, Figure 5.16, Figure 5.17, Figure 5.18, and Figure 5.19 show.</td>
</tr>
<tr>
<td>Parallel Port list box</td>
<td>Specifies the parallel port: LPT1, LPT2, LPT3, or LPT4.</td>
<td>Default: LPT1.</td>
</tr>
<tr>
<td>Speed text box (in parallel dialog box)</td>
<td>Integer that modifies the data stream transfer rate: 0 specifies the fastest rate. The greater the integer, the slower the rate.</td>
<td>For a parallel remote connection there is no firm mathematical relationship, so you may need to experiment to find the best transfer rate. In case of problems, try value 25.</td>
</tr>
<tr>
<td>USB Port list box (in USB dialog box and Cyclone Max USB dialog box)</td>
<td>Specifies the USB port: USB 0, USB 1, USB 2, or USB 4.</td>
<td>Default: USB 0</td>
</tr>
</tbody>
</table>
**Debugging**

**Remote Connections for Debugging**

**NOTE**
For a P&E Micro remote connection, be sure to check the checkboxes Use Target Initialization File and Use Memory Configuration File, of the CF Debugger Settings panel.

### ISS Remote Connection

**NOTE**
Special-Edition software does not support the ISS. To use the ISS, you must have Standard- or Professional-Edition software.

Figure 5.20 shows the configuration dialog box for ColdFire Instruction Set Simulator (ISS) remote connections. Table 5.10 explains the elements of this dialog box.

#### Table 5.9 P&E Micro Dialog Box Elements (continued)

<table>
<thead>
<tr>
<th>Speed text box (in USB dialog box)</th>
<th>Integer $N$ that specifies the data stream transfer rate per the expression $(1000000/(N+1))$ hertz.</th>
<th>$0$ specifies 1000000 hertz, or 1 megahertz. $1$ (the default) specifies 0.5 megahertz. $31$ specifies the slowest transfer rate: 0.031 megahertz.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCI card slot list box</td>
<td>Specifies PCI slot that the board uses.</td>
<td>Default: 1</td>
</tr>
<tr>
<td>Speed text box (in Lightning dialog box)</td>
<td>Integer $N$ that specifies the data stream transfer rate per the expression $(33000000/(2*N+5))$ hertz.</td>
<td>$0$ specifies 6600000 hertz, or 6.6 megahertz. $1$ (the default) specifies 4.7 megahertz. $31$ specifies the slowest transfer rate: 0.49 megahertz.</td>
</tr>
<tr>
<td>COM Port list box (in Cyclone Max Serial dialog box)</td>
<td>Specifies the serial port: COM1, COM2, COM3, ... or COM256.</td>
<td>Default: COM1.</td>
</tr>
<tr>
<td>IP Address text box (in Cyclone Max TCP/IP dialog box)</td>
<td>Specifies IP address.</td>
<td>Must be in format 127.0.0.1:1000 or in format host.domain.com:1000.</td>
</tr>
<tr>
<td>Speed text box (in Cyclone Max dialog boxes)</td>
<td>Integer $N$ that specifies the data stream transfer rate per the expression $(50000000/(2*N+5))$ hertz.</td>
<td>$0$ specifies 5000000 hertz, or 5.0 megahertz. $5$ (the default) specifies 3.33 megahertz. $31$ specifies the slowest transfer rate: 0.75 megahertz.</td>
</tr>
</tbody>
</table>
NOTE To use the ISS for V2 and V4e cores, create a CCS remote connection. Alternatively, use the default CCS - SIM connection from the Remote Connection panel list.

Table 5.10 ISS Dialog-Box Elements

<table>
<thead>
<tr>
<th>Elements</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name text box</td>
<td>Identifies the remote connection.</td>
<td>For an existing connection, already has a value.</td>
</tr>
<tr>
<td>Debugger list box</td>
<td>Identifies the debugger.</td>
<td>For an existing connection, already specifies ColdFire CCS</td>
</tr>
<tr>
<td>Show in processes list checkbox</td>
<td>Clear — Leaves the connection off the official list. Checked — Adds connection to the official list (select View &gt; Systems &gt; List); also adds connection to the pop-up list for debugging certain kinds of file.</td>
<td>Default: Clear</td>
</tr>
</tbody>
</table>
## Table 5.10 ISS Dialog-Box Elements (continued)

<table>
<thead>
<tr>
<th>Elements</th>
<th>Purpose</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection Type list box</td>
<td>Specifies CCS Remote Connection</td>
<td>Changing this value changes other elements of the dialog box.</td>
</tr>
<tr>
<td>Use Remote CCS checkbox</td>
<td>Clear — Launches the CCS locally. Checked — Starts debug code on a remote target; activates Server IP Address text box.</td>
<td>ISS must be running and connected to a remote target device.</td>
</tr>
<tr>
<td>Server IP Address text box</td>
<td>Specifies IP address of the remote machine, in format 127.0.0.1:1000 or host.domain.com:1000.</td>
<td>Available only if you check the Use Remote CCS checkbox.</td>
</tr>
<tr>
<td>Port # text box</td>
<td>Specifies the port number the CCS uses</td>
<td>Use only 40969 — the number of the port pre-wired for the simulator.</td>
</tr>
<tr>
<td>Specify CCS Executable checkbox</td>
<td>Clear — Uses the default CCS executable file. Checked — Lets you specify a different CCS executable file, activating the text box and Choose button. To do so, click the Choose button, then use the subordinate dialog box to select the executable file. Clicking OK puts the pathname in the text box.</td>
<td>Does not pertain to the simulator.</td>
</tr>
<tr>
<td>Multi-Core Debugging checkbox</td>
<td>Clear — Does not debug code on a multicore target. Checked — Lets you specify the JTAG chain for debugging on a multicore target, activating the text box and Choose button. To do so, click the Choose button, then use the subordinate dialog box to select the executable file. Clicking OK puts the pathname in the text box.</td>
<td>Does not apply to the simulator.</td>
</tr>
<tr>
<td>CCS Timeout text box</td>
<td>Specifies the number of seconds the CCS should wait for a connection to go through, before trying the connection again.</td>
<td></td>
</tr>
</tbody>
</table>
BDM Debugging

This section shows two examples of connections for Background Debugging Mode (BDM) debugging of a ColdFire target board.

Connecting a P&E Parallel Connector

Figure 5.21 depicts connections for a P&E Parallel Remote Connection.

Follow these steps:

1. Plug the P&E Parallel Connector onto the target-board BDM connector.
2. Connect the parallel cable to the P&E Parallel Connector.
3. Connect the other end of the parallel cable to a parallel port of your PC.
4. This completes P&E Parallel connection. The P&E Parallel Remote Connection automatically installs a default set of drivers and interface dlls on your PC.

**NOTE**

You must have the correct P&E Parallel Connector for your target. If necessary, contact P&E Microsystems for assistance.

The Windows drivers for P&E Microsystems BDM cables are available as well in subdirectory `bin\Plugins\Support\ColdFire\pemicro` of your CodeWarrior installation directory.

Connecting an Abatron BDI Device

Figure 5.22 depicts connections for an Abatron BDI device.
Follow these steps:

1. Connect the BDI device to your computer.
   a. Serial connection: Connect a serial cable between the BDI serial connector and a serial port of the PC, as Figure 5.22 shows.
   b. TCP/IP connection: Connect a TCP/IP cable between the BDI TCP/IP connector and an appropriate port of your PC.

2. Connect the appropriate RCD cable between the BDI JTAG connector and the JTAG connector of your target board. (The board JTAG connector is a 26-pin Berg-type connector.)

**NOTE** Certain target boards, such as the MCF5485, MCF5475, MCF5235, and MCF5271, require a different RCD cable than do other ColdFire boards. To make sure that your cable is correct, see the Abatron reference manual or visit http://www.abatron.ch.

3. Connect the power cable between the BDI power connector and a 5-volt, 1-ampere power supply, per the guidance of the Abatron user manual.

4. This completes cable connections.

**NOTE** Before using an Abatron remote connection, you must
   1. Make sure that you have the correct drivers and configuration utility for your target board.
   2. Use Abatron software to configure the BDI device, per the guidance of the Abatron user manual.
Before you use the BDI for ROM/Flash debugging, you must check the Use Breakpoint Logic checkbox of the BDI Working Mode dialog box.

Debugging ELF Files without Projects

The CodeWarrior debugger can debug an ELF file that you created in a different environment. But before you begin, you must update IDE preferences and customize the default XML project file. (The CodeWarrior IDE uses the XML file to create a project with the same target settings for any ELF file that you open to debug.)

Updating IDE Preferences

Follow these steps:

1. From the main menu bar, select Edit > Preferences. The IDE Preferences window appears.

2. From the IDE Preferences Panels pane, select Build Settings. The Build Settings panel (Figure 5.23) moves to the front of the window.

![Figure 5.23 Build Settings Panel](image)

3. Make sure that the Build before running list box specifies Never.

**NOTE**  Selecting Never prevents the IDE from building the newly created project, which is useful if you prefer to use a different compiler.

4. Select Edit > Preferences > Global Settings. The Global Settings panel (Figure 5.24) moves to the front of the window.
Debugging ELF Files without Projects

Customizing the Default XML Project File

CodeWarrior software creates a new CodeWarrior project for any ELF file that you open to debug. To create the new project, the software uses the target settings of the default XML project file:

```
bin\plugins\support\CF_Default_Project.xml
```

For different target settings, you must customize this default XML file. Follow these steps:

1. Import the default XML project file.
   a. Select File > Import Project — a file-select dialog box appears.
   b. Navigate to subdirectory `bin\plugins\support`.
   c. Select file `CF_Default_Project.xml`.
   d. Click OK — a new project window appears for file `CF_Default_Project.xml`.

2. Change target settings of the new project.
   a. Select Edit > Target Settings — the Target Settings window appears.
   b. From the Target Settings Panels pane, select any panel — that panel moves to the front of the window.
   c. Review/update panel settings.
Debugging

Debugging ELF Files without Projects

d. Repeat substeps b and c for all other appropriate panels.
e. When all settings are correct, click OK — the Target Settings window closes; the system updates project settings.

3. Close the project window.

4. Export the modified target settings.
   a. Select File > Export Project — a file-select dialog box appears.
   b. Navigate to subdirectory bin\plugins\support.
   c. Select the file you just modified: CF_Default_Project.xml.
   d. Click OK — the system saves your modified file CF_Default_Project.xml over the old file.

5. This completes XML-file customization — the new CF_Default_Project.xml file includes your target-settings changes; you are ready to debug an ELF file.

Debugging an ELF File

Once you have updated IDE preferences and customized the default XML file, you are ready to debug an ELF file (that includes symbolics information). Follow these steps:

1. Confirm that a remote connection exists for the ColdFire target.

2. Open Windows Explorer.

3. Navigate to the ELF file.

4. Drag the ELF file to the IDE main window — the IDE uses the default XML file to create a new project, opening a new project window.

   **NOTE** As ELF-file DWARF information does not include full pathnames for assembly (.s) files; the IDE cannot find these files when it creates the project. But when you debug the project, the IDE does find the assembly files that reside in a directory that is a project access path. If any assembly files still lack full pathnames, you can add their directory to the project manually, so that the IDE finds the directory whenever you open the project.

5. Select Project > Debug — the IDE starts the debugger; the debugger window appears.


Additional ELF-Debugging Considerations

Any of these points may make your debugging more efficient:
Once the IDE creates a .mcp project for your ELF file, you can open that project instead of dropping your ELF file onto the IDE.

To delete an old access path that no longer applies to the ELF file, use either of two methods:
- Use the Access Path target settings panel to remove the access path from the project manually.
- Delete the existing project for the ELF file, then drag the ELF file to the IDE to recreate a project.

To have the project include only the current files, you must manually delete project files that no longer apply to the ELF.

To recreate a project from an ELF file:
- If the project is open, close it.
- Delete the project (.mcp) file.
- Drag the ELF file to the IDE — the IDE opens a new project, based on the ELF file.

---

**Special Debugger Features**

This section explains debugger features that are unique to ColdFire-platform targets.

**ColdFire Menu**

To see the unique Coldfire debugger menu, select **Debug > ColdFire.** Table 5.11 lists its selections.

<table>
<thead>
<tr>
<th>Selection</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Target</td>
<td>Sends a reset signal to the target processor. (Not available unless the target processor supports this signal.)</td>
</tr>
<tr>
<td>Save Memory</td>
<td>Saves target-board data to disk, as a binary image file.</td>
</tr>
<tr>
<td>Load Memory</td>
<td>Writes previously saved, binary-file data to target-board memory.</td>
</tr>
<tr>
<td>Fill Memory</td>
<td>Fills a specified area of memory with a specified value.</td>
</tr>
<tr>
<td>Save Registers</td>
<td>Saves contents of specified register to a text file.</td>
</tr>
</tbody>
</table>
Working with Target Hardware

To have the IDE work with target hardware, use Debug-menu selections Connect and Attach.

**Connect**

This selection tells the IDE to read the contents of target-board registers and memory; these contents help you determine the state of the processor and target board. You can use this selection in combination with the Load/Save Memory and Fill Memory selections of the ColdFire menu to create a memory dump, load memory contents, or initialize memory with specific patterns of data.

You can have the IDE connect to a target board that uses ColdFire Abatron or ColdFire P&E Micro protocols.

The Connect selection works with a remote connection that you define in a project:

1. Bring forward the project you want to use. (The project must have at least one remote connection defined for the target hardware.)

2. Select Debug > Connect — a Thread window appears, showing where the IDE stops program execution. The debugger configuration file is executed.

3. Use the Thread window, with other IDE windows, to see register views and memory contents.

---

Table 5.11 ColdFire Debug Menu (continued)

<table>
<thead>
<tr>
<th>Selection</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restore Registers</td>
<td>Writes previously saved register contents back to the registers.</td>
</tr>
<tr>
<td>Watchpoint Type</td>
<td>Specifies the type:</td>
</tr>
<tr>
<td></td>
<td>Read — A read from the specified memory address stops execution.</td>
</tr>
<tr>
<td></td>
<td>Write — A write to the specified memory address stops execution.</td>
</tr>
<tr>
<td></td>
<td>Read/Write — Either a read from or write to the specified memory address stops execution.</td>
</tr>
<tr>
<td></td>
<td>(Not available unless the target processor and debug connection support watchpoints.)</td>
</tr>
</tbody>
</table>
Using the Simple Profiler

NOTE For a detailed explanation of how to instrument your code for profiling and how to interpret the results, see the “Profiler” chapter in the IDE 5.7 User’s Guide.

The following steps for enabling and using the profiling tool make reference to ColdFire-specific features that may not be mentioned in the IDE User’s Guide:

1. Specify profiling, in one of these ways:
   a. In the ColdFire Processor panel, check the Generate code for profiling checkbox.
   b. Use the #pragma profile on directive before the function definition and use the #pragma profile off directive after the function definition.
   c. Use the -profile option or the #pragma directives with the command-line compiler.

2. If you use the #pragma directives, add the profiler libraries to your project. (These libraries are in subdirectory \E68K_Support\Profiler\Lib\ of your CodeWarrior installation directory.)

3. In your source code, use the #include directive to include header file Profiler.h. (This file is in subdirectory \E68K_Support\Profiler\include\ of your CodeWarrior installation directory.)

4. If necessary, instrument your code with the following function calls. These functions are documented in the “Profiler” chapter of the IDE User’s Guide:
   a. ProfilerInit — initializes the profiler.
   b. ProfilerClear — removes existing profiling data.
   c. ProfilerSetStatus — turns profiling on (1) or off (0).
   d. ProfilerDump("filename") — dumps the profile data to a profiler window or to the specified file.
   e. ProfilerTerm — exits the profiler.

The profiler libraries use the external function getTime to measure the actual execution time.

NOTE For a list of ColdFire platforms supported by the profiler, see the subdirectory \E68K_support\Profiler\Support\ of your CodeWarrior installation directory. The files in that directory also contain examples using the getTime() function.
Instruction Set Simulator

This chapter explains how to use the Instruction Set Simulator (ISS). Using the ISS with the CodeWarrior™ debugger, you can debug code for a ColdFire target.

Additionally, if you run the ISS on your host computer, you can share target-board access with remote users of the CodeWarrior debugger.

In the same way, you can access the target board of any remote computer that is running the ISS, provided that you know the IP address and ISS port number of that remote computer.

NOTE Special-Edition software does not support the ISS; to use the ISS, you must have Standard- or Professional-Edition software.

Do not move the ISS folders or files from its location in subdirectory \Bin\Plugins\Support\Sim of your CodeWarrior installation directory. You can start the ISS only from the CodeWarrior debugger.

This chapter consists of these sections:

- Features
- Using the Simulator
- ISS Configuration Commands
- Sample Configuration File
- ISS Limitations

Features

Your CodeWarrior software supports the Instruction Set Simulator (ISS) for V2 and V4e cores.

ColdFire V2

For V2 cores the ISS features are:

- *Instruction set* — modeling only of the original ColdFire v2 instruction set, without ISA+ support of the 5282 processor.
Instruction Set Simulator

Features

- **MAC** — modeling of the MAC *without* the EMAC of the 5282 processor. (This affects register accesses.)
- **Cache** — modeling of the original ColdFire v2 direct-mapped instruction cache, *without* modeling of the 5282 instruction and data cache.
- **Format exceptions** — not implemented.
- **IPSBAR Functionality (5282 Peripherals)** — modeling of the IPSBAR register and Synchronous DRAM Controller (SDRAMC) module. (No modeling of other 5282 peripherals or related behavior.)
- **IPSBAR register fields** — all implemented.
- **SDRAMC registers** — five present:
  - **DCR**
  - **DACR0**
  - **DACR1**
  - **DMR0**
  - **DMR1**
- **DCR** — this model includes reads from and writes to this register, but ignores all internal fields of this register.
- **DACRx** — this model includes reads from and writes to these fields. (The SDRAMC model covers functionality only of DACRx register fields BA and CBM, ignoring other fields.)
- **DMRx** — this model includes reads from and writes to these fields. (The SDRAMC model covers functionality only of DMRx register fields BAM and V, ignoring other fields.)
- **KRAM, KROM** — support as much as 512 kilobytes of memory.
- **Memory wait states** — supported.
- **A-line exceptions** — not generated, as this model includes MAC.

**NOTE** The V2 ISS has pipeline delays that can lead to debugger defects.

ColdFire V4e

For V4e cores the ISS features are:

- **Instruction set** — modeling for all instructions.
- **EMAC** — modeling of the EMAC.
- **FPU** — not supported.
• Cache — modeling of the ColdFire V4e four-way set-associative instruction and data caches. (Caches always are physically tagged and physically addressed.)
• MMU model — partially supported.
• WDEBUG instruction — not supported, as the model does not support the WDEBUG module.
• WDDATA instruction — not supported, as the model does not support the WDDATA module.
• PULSE instruction — not supported, as the model does not support the PULSE debug module.
• IPSBAR Functionality (5282 Peripherals) — modeling of the IPSBAR register. (No modeling of other peripherals or related behavior.)
• A-line exceptions — not generated, as this model includes EMAC.
• F-line exceptions — not generated, as this model includes an FPU.
• Clock multiplier — not supported.
• Memory wait states — not supported.

NOTE Pipeline delay can lead to appearance problems in the debugger variable viewer.

Using the Simulator

When you use a local ISS connection for debugging, the IDE starts the ISS automatically; the ISS icon appears on the taskbar.

Right-click the icon to access the ISS pop-up menu. Its selection are:
• Configure — opens the ISS configuration options dialog box
• Show console — displays the ISS console window. (Another way to open this console window is double-clicking the ISS icon.)
• Hide console — hides the ISS console window
• About CCSSIM2 — displays version information
• Quit CCS — stops the ISS.

Console Window

Use the ISS console window to view and change server connection options. You may type commands at the command line, or select them from the menu bar.
NOTE  Do not use the console window to modify settings during a debug session. This would affect the debug state.

Viewing ISS Registers
To view the ISS registers, select View > Registers — the Registers window (Figure 6.1) appears.

Figure 6.1  Register Window: ISS Register Values

You may edit the ISS register values that this window shows.
- INSTCNT (Instruction Count) is the number of instructions executed in a debug session.
- CYCLCNT (Cycle Count) is the number of elapsed clock cycles in a debug session.

NOTE  These registers are unique to ISS projects; other projects do not have these registers.

ISS Configuration Commands

The ISS reads configuration information from configuration files ColdFire2.cfg (V2 core) and ColdFire4.cfg (V4e core). Both files are in subdirectory Bin\Plugins\Support\Sim\ccssim2\bin of your CodeWarrior installation directory.

NOTE  Do not change the location of the configuration files, or the ISS may not work properly.

If you cannot use the ISS to start a debug session, you probably must reduce the memory that file ColdFire2.cfg or ColdFire4.cfg defines. And for an MCF5282 or other processor core that had IPSBAR, you must use the ipsbar command to configure the settings.

The configuration files consist of text commands, each on a single line:
Some argument values are numerical.
Possible boolean argument values are true (or yes) and false (or no).
Comment lines must start with the # character.

The rest of this section consists of explanations for the ISS configuration commands:

- **bus_dump**
- **cache_size**
- **ipsbar**
- **kram_size**
- **krom_size**
- **krom_valid**
- **mbar**
- **mbus_multiplier**
- **memory**
- **sdram**

### bus_dump

Controls dumping bus signals to the `processor.bus_dump` file.

**Parameter**

`switch`

Boolean value yes (or true) or no (or false).

**Remarks**

If environment variable CF_REG_DUMP is set, a yes or true switch value for this command also dumps the CPU register values to the `processor.reg_dump` file.

**Example**

`bus_dump true`
cache_size

Configures the cache size.

```
cache_size size_parameter
```

**Parameter**

`size_parameter`

Default value 0 (off), or another code number for the size, per Table 6.1.

---

Table 6.1 Cache Size Parameter Conversion

<table>
<thead>
<tr>
<th>size_parameter</th>
<th>Kilobytes</th>
<th>size_parameter</th>
<th>Kilobytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>32</td>
</tr>
</tbody>
</table>

---

**Example**

```
cache_size 7
```

---

ipsbar

Provides beginning address and offset, enabling V4-core IPSBAR registers. (The V4 counterpart command is mbar.)

```
ipsbar switch
```

**Parameter**

`switch`

Boolean value yes (or true) or no (or false).

---

**Example**

```
ipsbar true
```
**kram_size**

Configures the KRAM size.

kram_size size_parameter

**Parameter**

size_parameter

Code number for the size, per Table 6.2.

**Table 6.2 kram Size Parameter Conversion**

<table>
<thead>
<tr>
<th>size_parameter</th>
<th>Kilobytes</th>
<th>size_parameter</th>
<th>Kilobytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>6</td>
<td>16</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>7</td>
<td>32</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>9</td>
<td>128</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>10</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>11</td>
<td>512</td>
</tr>
</tbody>
</table>

**Example**

kram_size 7

**krom_size**

Configures the KROM size.

krom_size size_parameter

**Parameter**

size_parameter

Code number for the size, per Table 6.3.
### krom_valid

Controls KROM mapping to address $0$ at boot-up.

**krom_valid switch**

**Parameter**

switch

Boolean value **yes** (or **true**) or **no** (or **false**).

**Example**

krom_valid true

### mbar

Provides beginning address and offset, enabling V2-core MBAR registers. (The V4 counterpart command is **ipsbar**.)

mbar switch
Parameter

switch

Boolean value yes (or true) or no (or false).

Example

mbar true

mbus_multiplier

For a V2-core processor, multiplies the core clock speed.

mbus_multiplier value

Parameter

value

Any integer between 1 and 10.

Example

mbus_multiplier 10

memory

Configures sections of external memory.

memory start end wait_states line_wait_states

Parameters

start

Starting address of the contiguous section of memory.

end

Ending address of the contiguous section of memory.

wait_states

Number of wait states inserted for normal access (for V2 ISS only).

line_wait_states

Number of wait states inserted for line access (for V2 ISS only).
Remarks
There may be any number of MBUS memories, each with different wait states settings.
You must provide wait_states and line_wait_states values for a V2 ISS, but you should not provide these values for a V4 ISS.

Examples
memory 0x00000000 0x0fffffff 0 0
memory 0x200000000 0x3000ffff 0 0

sdram
Configures SDRAM.

sdram bank_bits num_bytes wait_states line_wait_states

Parameters

bank_bits
Number of bank bits used (only two banks are allowed).

num_bytes
Number of bytes allocated.

wait_states
Number of wait states inserted for normal access (for V2 ISS only).

line_wait_states
Number of wait states inserted for line access (for V2 ISS only).

Example
sdram 2 0x8000 0 0

Sample Configuration File

Listing 6.1 shows configuration file ColdFire2.cfg.

Listing 6.1 ColdFire2.cfg File Example

Example Configuration File
memory 0x0000 0x7fff 0 0
ISS Limitations

These limitations apply to the ISS:

- You cannot set hardware breakpoints, because debugging is not happening on an actual hardware board.
- You cannot set watchpoints in source code.
- You cannot use the Attach feature while you use the ISS.
- The Run Without Debugger button does not work, if you use the ISS to run your application.
Using Hardware Tools

This chapter explains the CodeWarrior IDE hardware tools, which you can use for board bring-up, test, and analysis. These tools are not used with CCS-SIM or other simulators.

This chapter consists of these sections:

- Flash Programmer
- Hardware Diagnostics

Flash Programmer

Use the CodeWarrior flash programmer to program target-board flash memory with code from any CodeWarrior IDE project, or with code from any individual executable files.

The flash programmer runs as a CodeWarrior plug-in, using the CodeWarrior debugger protocol API to communicate with the target boards. The CodeWarrior flash programmer lets you use the same IDE to program the flash of any of the embedded target boards.

**NOTE**

For Special-Edition software, the CodeWarrior flash programmer is limited to 128 kilobytes. There is no such limitation for Standard-Edition or Professional-Edition software.

Each software edition also comes with an optional ColdFire flash programmer, available in subdirectory `\bin\Plugins\Support\Flash_Programmer` of the CodeWarrior installation directory.

Follow these steps:

1. Make sure to build the application you want to program into flash memory.
2. From the IDE main menu bar, select **Tools > Flash Programmer** — the **Flash Programmer** window (Figure 7.1) appears.
3. If the Target Configuration panel is not visible, select it from the list at the left — the panel moves to the front of the Flash Programmer window.

4. Verify Target Configuration settings.
   a. If the Default Project field specifies your project, skip ahead to substep c.
   b. Otherwise, from the main menu bar, select **Project > Set as default project** to specify your project.
   c. If the Default Target field specifies the correct Flash target, skip ahead to substep e.
   d. Otherwise, from the main menu bar, select **Project > Set as default target** to specify the correct Flash target.
   e. Make sure that the Use Custom Settings checkbox is clear.
   f. Click the **Load Settings** button. A file browser will appear.
   g. Browse to `bin\Pugins\Support\Flash_Programmer\ColdFire` and select the appropriate xml file, then press **Open** — the system updates other settings for the default project and target.

5. Configure the flash device.
Using Hardware Tools

Flash Programmer

a. From the pane list at the left of the **Flash Programmer** window, select **Flash Configuration** — the Flash Configuration panel moves to the front of the window, as Figure 7.2 shows.

Figure 7.2 Flash Programmer Window: Flash Device Configuration Panel

b. Make sure that the Device list box specifies your external flash device, or the on-chip flash of your ColdFire-derivative processor.

c. Make sure that the Flash Memory Base Address text box specifies the appropriate base address.

d. The Organization and Sector Address Map boxes display appropriate additional information.

6. Erase the destination flash-memory sectors.
Using Hardware Tools
Flash Programmer

a. From the pane list at the left of the Flash Programmer window, select Erase/Blank Check — the Erase/Blank Check panel moves to the front of the window, as Figure 7.3 shows.

Figure 7.3 Flash Programmer Window: Erase/Blank Check Flash Panel

b. In the panel’s list box, select the sectors you want to erase. (To select them all, check the All Sectors checkbox.)

c. Click the Erase button — the flash programmer erases the sectors.

d. (Optional) To confirm erasure, select the same sectors, then click the Blank Check button — a message reports the status of the sectors.

7. Flash your application.
a. From the pane list at the left of the Flash Programmer window, select Program/Verify — the Program/Verify Flash panel moves to the front of the window, as Figure 7.4 shows.

Figure 7.4 Flash Programmer Window: Program/Verify Flash Panel

b. Make sure that the Use Selected File checkbox is clear.

c. Click the Program button — the flash programmer programs your application into the target sectors of flash memory.

d. (Optional) To confirm programming, click the Verify button — the flash programmer compares the data now in flash sectors to the image file on disk.

8. (Optional) For an additional test of programmed flash sectors, run a checksum.
Using Hardware Tools

Hardware Diagnostics

a. From the pane list at the left of the Flash Programmer window, select Checksum — the Checksum panel moves to the front of the window, as Figure 7.5 shows.

Figure 7.5 Flash Programmer Window: Checksum Panel

b. In the Compute Checksum Over area, select the appropriate option button: File on Target, File on Host, Memory Range on Target, or Entire Flash.

c. If this selection activates the Address Range text boxes, enter the appropriate Start and Size values.

d. Click the Calculate Checksum button — the flash programmer runs the checksum calculation; a message tells you the result.

9. This completes flash programming.

Hardware Diagnostics

Use the CodeWarrior hardware diagnostics tool to obtain several kinds of information about the target board. The Hardware Diagnostics feature is not supported by the ISS.

Select Tools > Hardware Diagnostics from the IDE main menu bar — the Hardware Diagnostics window (Figure 7.6) appears.
Figure 7.6  Hardware Diagnostics window: Configuration Panel

Figure 7.6 shows the Configuration panel. Click any name in the list pane to bring the corresponding panel to the front of the window:

- Memory Read/Write Test — which Figure 7.7 shows.
- Scope Loop Test — which Figure 7.8 shows.
- Memory Tests — which Figure 7.9 shows.

NOTE  In Figure 7.7, Figure 7.8, and Figure 7.9, be sure to use addresses which are in the memory map for your hardware configuration (which are defined through the configuration file).
Using Hardware Tools

Hardware Diagnostics

Figure 7.7 Hardware Diagnostics window: Memory Read/Write Test Panel

Figure 7.8 Hardware Diagnostics window: Scope Loop Test Panel
The **Hardware Diagnostics** window lists global options for the hardware diagnostic tools; these preferences apply to every open project file. For more information about each hardware-diagnostics panel, see the *IDE User’s Guide*.
Using Hardware Tools

Hardware Diagnostics
Using Debug Initialization Files

This appendix explains background debugging mode (BDM) support for the ColdFire reference boards. BDM controls the processor, accessing both memory and I/O devices via a simple serial, wiggler interface. BDM can be very useful during initial debugging of control system hardware and software; it also can simplify production-line testing and end-product configuration.

Specifically, this appendix explains how to use debug initialization files with the P&E Micro wiggler. Debug initialization files contain commands that initialize the target board to write the program to memory, once the debugger starts the program.

Each time you start the debugger, and each time you select **Debug > Reset Target**, the system processes a debug initialization file. Such a file perform such functions as initializing registers and memory in targets that do not yet have initialization code in ROM.

This appendix consists of these sections:

- Common File Uses
- Command Syntax
- Command Reference

You specify whether to use a debug initialization file — and which to use — via the **ColdFire Target Settings** panel.

Common File Uses

The most common use for debug initialization files is configuring the essential set of memory-control registers, so that downloads and other memory operations are possible. This is appropriate if your target system or evaluation board does not yet have initialization code in target ROM. It also can be an appropriate way to override an existing initialization after a reset.

To create this section of the debug initialization file, you mirror the values that the processor chip-select, pin-assignment, and other memory control registers should have after execution of initialization code. However, the set of registers that need initialization...
Using Debug Initialization Files

Common File Uses

varies by processor. For details, see your processor data book, as well as the sample files in the CodeWarrior subdirectory \E68K_Tools\Initialization_Files.

Other uses and guidance items are:

- Sample files are specific to processor, and, in most cases, evaluation board. Use the sample templates for your own board.
- Use a debug initialization file only to initialize memory setup. Trying to use such a file for additional initialization, such as for on-board peripherals or setup ports, would prevent these other initializations during normal execution. As the program does not use BDM in normal execution, it would not initialize such peripherals, so the program could fail to execute properly.
- Put non-memory and non-exception-setup initialization instructions in the init_hardware function of processor startup code instead of in a debug initialization file. Another valid place for such instructions is your own start routine. These methods take care of initialization.
- Once debugging is done, your startup code must initialize the memory management unit, setting up the memory appropriately for non-debugger program execution.
- Listing 8.1 is a sample BDM initialization file for the MCF5272C3 board.

Listing 8.1  Sample BDM Initialization file

; Set VBR to start of future SDRAM
; VBR is an absolute CPU register
; SDRAM is at 0x00000000+0x0400000
writecontrolreg 0x0801 0x00000000

; Set MBAR to 0x10000001
; MBAR is an absolute CPU register, so if
; you move MBAR, you must change all subsequent
; writes to MBAR-relative locations
writecontrolreg 0x0C0F 0x10000001

; Set SRAMBAR to 0x20000001
; SRAMBAR is an absolute CPU register, the
; location of chip internal 4k of SRAM
writecontrolreg 0x0C04 0x20000001

; Set ACR0 to 0x00000000
writecontrolreg 0x04 0x00000000

; Set ACR1 to 0x00000000
writecontrolreg 0x05 0x00000000

; 2MB FLASH on CS0 at 0xFFF00000
writemem.1 0x10000040 0xFFF00201
writemem.1 0x10000044 0xFFF00014

; CS7 4M byte SDRAM

; Unlike 5307 and 5407 Cadre 3 boards, the 5272 uses CS7 to access SDRAM
writemem.1 0x10000078 0x00000701
writemem.1 0x1000007C 0xFFF0007C

; Set SDRAM timing and control registers
; SDCTR then SDCCR
writemem.1 0x10000184 0x0000F539
writemem.1 0x10000180 0x00004211

; Wait a bit
delay 600

writemem.1 0x10000000 0xDEADBEEF

; Wait a bit more
delay 600

---

**Command Syntax**

The syntax rules for commands in a debug initialization file are:

- The system ignores space characters and tabs.
- The system ignores character case in commands.
- Numbers may be in hexadecimal, decimal, or octal format:
  - Hexadecimal values must start with 0x, as in 0x00002222, 0xA, or 0xCAfeBeaD.
  - Decimal values must start with a numeral 1 through 9, as in 7, 12, 526, or 823643.
  - Octal values must start with a zero, as in 0123, or 0456.
• Start comments with a colon (;), or pound sign (#). Comments end at the end of the line.

# Command Reference

This section explains the commands valid for debug initialization files:

- **Delay**
- **ResetHalt**
- **ResetRun**
- **Stop**
- **writeaddressreg**
- **writecontrolreg**
- **writedatareg**
- **writemem.b**
- **writemem.l**
- **writemem.w**

## NOTE

Old data initialization files that worked with a Macraigor interface may not work with a P&E interface because command `writereg SPRnn` changed to `writecontrolreg 0xNNNN`. Please update files accordingly.

## Delay

Delays execution of the debug initialization file for the specified time.

**Delay <time>**

**Parameter**

- **time**

  Number of milliseconds to delay.

**Example**

This example creates a half-second pause in execution of the debug initialization file:

```
Delay 500
```
ResetHalt

Resets the target, putting the target in debug state.

ResetHalt

ResetRun

Resets the target, letting the target execute from memory.

ResetRun

Stop

Stops program execution, putting the target in a debug state.

Stop

writeaddressreg

Writes the specified value to the specified address register.

writeaddressreg <registerNumber> <value>

Parameters

registerNumber

Any integer, 0 through 7, representing address register A0 through A7.

value

Any appropriate register value.

Example

This example writes hexadecimal ff to register A4:

writeaddressreg 4 0xff
writecontrolreg

Writes the specified value to the address of a control register.

writecontrolreg <address> <value>

*address* is the address of the control register.

**Parameters**

*address*

Address of any control register.

*value*

Any appropriate value.

**Example**

This example writes hexadecimal c0f to control-register address 20000001:

```
writecontrolreg 0xc0f 0x20000001
```

writedatareg

Writes the specified value to the specified data register.

writedatareg <registerNumber> <value>

**Parameters**

*registerNumber*

Any integer, 0 through 7, representing data register D0 through D7.

*value*

Any appropriate register value.

**Example**

This example writes hexadecimal ff to register D3:

```
writedatareg 3 0xff
```
writemem.b

Writes the specified byte value to the specified address in memory.

\texttt{writemem.b\ <address>\ <value>}

\textbf{Parameters}

address
One-byte memory address.

value
Any one-byte value.

\textbf{Example}

This example writes decimal 255 to memory decimal address 2345:

\texttt{writemem.b\ 2345\ 255}

writemem.l

Writes the specified longword value to the specified address in memory.

\texttt{writemem.l\ <address>\ <value>}

\textbf{Parameters}

address
Four-byte memory address.

value
Any four-byte value.

\textbf{Example}

This example writes hexadecimal 00112233 to memory hexadecimal address 00010000:

\texttt{writemem.l\ 0x00010000\ 0x00112233}
writesmem.w

Writes the specified word value to the specified address in memory.

writesmem.w <address> <value>

**Parameters**

- **address**
  - Two-byte memory address.
- **value**
  - Any two-byte value.

**Example**

This example writes hexadecimal 12ac to memory hexadecimal address 00010001:

writesmem.w 0x00010001 0x12ac
Memory Configuration Files

In your overall memory map, there can be gaps or holes between physical memory devices. If the debugger tries a read or write to an address in such a hole, the system would issue an error message, and debugging might not even be possible.

To prevent such developments, use a memory configuration file (MCF). An MCF identifies valid memory address ranges to the debugger, and even specifies valid access types.

**NOTE**  The memory configuration file for your project should be updated to the latest memory configuration file for your hardware configuration. Up-to-date memory files can be found in \E68K\Support\Initialization_Files.

A sample memory configuration file is \E68K\Support\Initialization_Files\MCF5485EVB.mem of the CodeWarrior installation directory.

This appendix consists of these sections:

- Command Syntax
- Command Explanations

**Command Syntax**

The syntax rules for commands in a memory configuration file are:

- The system ignores space characters and tabs.
- The system ignores character case in commands.
- Numbers may be in hexadecimal, decimal, or octal format:
  - Hexadecimal values must start with 0x, as in 0x00002222, 0xA, or 0xCAfeBeaD.
  - Decimal values must start with a numeral 1 through 9, as in 7, 12, 526, or 823643.
  - Octal values must start with a zero, as in 0123, or 0456.
- Comments can be in standard C or C++ format.
Command Explanations

This section explains the commands you can use in a memory configuration file:

- range
- reserved
- reservedchar

**range**

Specifies a memory range for reading or writing.

```plaintext
range <loAddr> <hiAddr> <sizeCode> <access>
```

**Parameters**

- `loAddr`
  - Starting address of memory range.
- `hiAddr`
  - Ending address of memory range.
- `sizeCode`
  - Size, in bytes, for memory accesses by the debug monitor or emulator.
- `access`
  - Read, Write, or ReadWrite.

**Example**

These range commands specify three adjacent ranges: the first read-only, with 4-byte access; the second write-only, with 2-byte access; and the third read/write, with 1-byte access.

```
range 0xFF000000 0xFF0000FF 4 Read
range 0xFF000100 0xFF0001FF 2 Write
range 0xFF000200 0xFFFF0000 1 ReadWrite
```
reserved

Reserves a range of memory, preventing reads or writes.
reserved <loAddr> <hiAddr>

Parameters

loAddr

Starting address of reserved memory range.

hiAddr

Ending address of reserved memory range.

Remarks

If the debugger tries to write to any address in the reserved range, no write takes place.
If the debugger tries to read from any address in the reserved range, the system fills the memory buffer with the reserved character. (Command reservedchar defines this reserved character.)

Example

This command prevents reads or writes in the range 0xFF00024 — 0xFF00002F:
reserved 0xFF000024 0xFF00002F

reservedchar

Specifies a reserved character for the memory configuration file.
reservedchar <char>

Parameter

char

Any one-byte character.

Remarks

If an inappropriate read occurs, the debugger fills the memory buffer with this reserved character.
Example

reservedchar 0xBA
Index

A
Abatron
   BDI connection 83, 85
   remote connections 71–73
application development diagram 16
application tutorial 21–32

B
batchrunner postlinker settings panel 37
batchrunner prelinker settings panel 37
BDM debugging 83, 85
build settings panel 85
building 18, 19
building a project 25, 27
bus_dump simulator configuration command 95

C
cache_size simulator configuration command 96
CF
debugger settings panel 27, 56–59
   exceptions panel 63–66
   interrupt panel 68
checksum panel 108
CodeWarrior
development process 15–20
   IDE 14
ColdFire
   assembler panel 38–41
   linker panel 48–53
   processor panel 45–48
   settings panels 34–53
   target panel 38
commands
debug initialization files 116–120
   memory configuration files 122–124
   simulator configuration 94–100
compiling 18, 19
configuration files, memory 121–124
configuration panel 109
connections
   Abatron BDI 83, 85
   remote 61–63
   wiggler 83
console window 32
creating a project 21–24

D
debug initialization files 113–120
commands 116–120
   Delay 116
   ResetHalt 117
   ResetRun 117
   Stop 117
   writeaddressreg 117
   writecontrolreg 118
   writedatereg 118
   writemem.b 119
   writemem.l 119
   writemem.w 120
syntax 115
uses 113–115
debugger PIC settings 53
debugger settings panel 66, 68
debugger window 28
debugging 19, 55–90
   Abatron remote connections 71–73
   an application 27–32
   BDM debugging 83, 85
      connecting a wiggler 83
      connecting Abatron BDI 83, 85
   ELF files 85–88
      customizing XML file 86, 87
      IDE preferences 85, 86
   Freescale remote connections 73–76
   ISS remote connection 80–82
   P&E Micro remote connections 76–80
   remote connections 69–82
   simple profiler 90
   special features 88–90
   target settings 55–69
      CF debugger settings panel 56–59
      CF exceptions panel 63–66
      CF interrupt panel 68
debugger settings panel 66, 68
remote connections 61–63
remote debugging panel 59–61
Delay debug initialization command 116
development process, CodeWarrior 15–20
dialog boxes
new 22
new connection 62
new project 23
disassembling 19
documentation 10, 11

E
editing 18
editions 8, 9
editor window 18
ELF
disassembler panel 41–44
files
customizing XML file 86, 87
debugging 85–88
IDE preferences 85, 86
erase/blank check flash panel 106

F
disassembler panel 41–44
files
customizing XML file 86, 87
debugging 85–88
IDE preferences 85, 86
erase/blank check flash panel 106

G
global settings panel 86

H
hardware diagnostics 108–111
hardware diagnostics window 109–111
hardware tools 103–111
flash programmer 103–108
hardware diagnostics 108–111
hardware diagnostics window 109–111
hardware tools 103–111
flash programmer 103–108
hardware diagnostics 108–111
IDE
CodeWarrior 14
preferences, updating 85, 86
IDE preferences window 62
instruction set simulator 91–101
limitations 101
sample configuration file 100
introduction 7–11
ipsbar simulator configuration command 96
ISS
configuration commands 94–100
bus_dump 95
cache_size 96
ipsbar 96
kram_size 97
krom_size 97, 98
krom_valid 98
mbar 98, 99
mbus_multiplier 99
memory 99, 100
sdram 100
features 91–93
remote connection 80–82
kram_size simulator configuration command 97
krom_size simulator configuration command 97, 98
krom_valid simulator configuration command 98
linking 18, 19
main window 22
mbar simulator configuration command 98, 99
mbus_multiplier simulator configuration command 99
memory configuration files 121–124
commands 122–124
range 122
reserved 123
reservedchar 123, 124
syntax 121
memory read/write test panel 110
memory simulator configuration command 99, 100
memory tests panel 111

N
new connection dialog box 62
new dialog box 22
new project dialog box 23

O
overview, target settings 33, 34

P
P&E Micro remote connections 76–80
panels
batchrunner postlinker 37
batchrunner prelinker 37
build settings 85
CF debugger settings 27, 56–59
CF exceptions 63–66
CF interrupt 68
checksum 108
ColdFire assembler 38–41
ColdFire linker 48–53
ColdFire processor 45–48
ColdFire settings 34–53
ColdFire target 38
configuration 109
destructor settings 66, 68
ELF disassembler 41–44
erase/blank check flash 106
flash device configuration 105
global settings 86
memory read/write test 110
memory tests 111
program/verify flash 107
remote connections 62
remote debugging 26, 59–61
scope loop tests 110
target configuration 104
target setting 35–36
target settings 25
PIC
settings 53
profiler 90
program/verify flash panel 107
project
building 25, 27
creating 21–24
debugging 27–32
files 17
window 17
project window 24

R
range memory configuration command 122
registers window 30, 94
remote connections 61–63
debugging 69–82
panel 62
remote debugging panel 26, 59–61
requirements, system 13, 14
reserved memory configuration command 123
reservedchar memory configuration command 123, 124
ResetHalt debug initialization command 117
ResetRun debug initialization command 117

S
sample code
transitive closure 52, 53
sample ISS configuration file 100
scope loop tests panel 110
sdram simulator configuration command 100
settings
panels 34–53
target 33–53
target, debugging 55–69
simple profiler 90
simulator 91–101
simulator configuration commands 94–100
simulator, using 93, 94
software editions 8, 9
special features, debugging 88–90
starting 13–20
Stop debug initialization command 117
syntax
debug initialization files 115
memory configuration files 121
system requirements 13, 14

target configuration panel 104
target settings 33–53
debugging 55–69
overview 33, 34
panel 35–36
window 34, 56
target settings panel 25
target settings window 25, 26
tools
flash programmer 104–108
hardware diagnostics 109–111
IDE preferences 62
main 22
project 17, 24
register 30, 94
target settings 25, 26, 34, 56
view memory 31
writeaddressreg debug initialization command 117
writecontrolreg debug initialization command 118
writedatereg debug initialization command 118
writemem.b debug initialization command 119
writemem.l debug initialization command 119
writemem.w debug initialization command 120

XML file, customizing 86, 87