

WIND RIVER

Wind River[®] Workbench for On-Chip Debugging

USER TUTORIALS

2.6.1

Copyright © 2007 Wind River Systems, Inc.

All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means without the prior written permission of Wind River Systems, Inc.

Wind River, the Wind River logo, Tornado, and VxWorks are registered trademarks of Wind River Systems, Inc. Any third-party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see:

<http://www.windriver.com/company/terms/trademark.html>

This product may include software licensed to Wind River by third parties. Relevant notices (if any) are provided in your product installation under the following directory:
installDir\product_name\3rd_party_licensor_notice.pdf.

Corporate Headquarters

Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501-1153
U.S.A.

toll free (U.S.): (800) 545-WIND
telephone: (510) 748-4100
facsimile: (510) 749-2010

For additional contact information, please visit the Wind River URL:

<http://www.windriver.com>

For information on how to contact Customer Support, please visit the following URL:

<http://www.windriver.com/support>

Contents

1	Introduction	1
1.1	Document Overview	1
1.2	Other Resources	2
2	Basic Operation: Debugging with a Project	5
2.1	Introduction	5
2.2	Connecting to the Target	6
2.3	Creating a Project	18
2.3.1	Downloading the Sample Code	23
2.4	Debugging Code	25
2.4.1	Monitoring Processes	25
2.4.2	Stepping Through Code	25
2.4.3	Setting a Software Breakpoint	27
2.4.4	Running a Program	28
2.4.5	Stepping Through a Program	29
2.4.6	Setting a Hardware Breakpoint	29
2.4.7	Disconnecting and Terminating Processes	32

2.5	Moving On	32
3	Basic Operation: Debugging Without a Project	33
3.1	Introduction	33
3.2	Connecting to the Target	34
3.3	Downloading Code	46
3.4	Debugging Code	53
3.4.1	Monitoring Processes	54
3.4.2	Stepping Through Code	54
3.4.3	Setting a Software Breakpoint	55
3.4.4	Running a Program	56
3.4.5	Stepping Through a Program	57
3.4.6	Setting a Hardware Breakpoint	58
3.4.7	Disconnecting and Terminating Processes	61
3.5	Moving On	61
4	Using the OCD Standalone Project Wizard	63
4.1	Introduction	63
4.2	Creating an OCD Standalone Project	64
4.3	Building an OCD Standalone Project	71
4.4	Setting Standalone Project Defaults	72
5	Defining a Launch Configuration	75
5.1	Introduction	75
5.2	Creating a Launch Configuration	76
5.2.1	Specifying Files	78

5.3	Other Options	85
6	Using Board Descriptor Files	87
6.1	Introduction	87
6.2	Board Descriptor Files	88
6.3	Creating a New Board Descriptor File	89
	Using the Predefined Layouts in JTAG Editor	91
	Using the Custom Option in the JTAG Editor View	95
	Editing Your Board Layout	97
6.4	XML Board Files	98
6.4.1	XML Board File Fields	100
	<DEVICE_TABLE> Fields	100
	<DEVICE> Fields	100
6.5	Manually Creating XML Board Files	101
7	Debugging Multi-Core Targets	105
7.1	Introduction	105
7.2	JTAG Server	106
7.3	Multi-Core Debugging	107
7.3.1	Establishing Communications with Multiple Devices	107
	Configuring Communication Settings Manually	110
7.4	Initializing the Targets	119
7.5	Creating a Project	125
7.5.1	Downloading the Sample Code	130
7.6	Configuring Options for Multi-Core Debugging	131
	CF HRESET	133
	CF CMDRST	133

7.7	Commands for Multi-Core Debugging	134
8	Configuring Target Registers	137
8.1	Introduction	137
8.2	Downloading a Register File	138
8.3	Saving Register Settings from a Target	139
8.4	Enabling and Disabling Register Groups	141
	Enabling and Disabling Register Groups with Low-Level Commands	
	142	
8.5	Configuring Registers Manually	143
8.6	Working With Custom Register Groups	147
	Creating a New Set of Registers	147
	Creating New Registers With Low-Level Commands	149
	SCGA Options	151
	Using Your New Register File	153
	Modifying an Existing Register File	154
8.7	System Configuration (SC) Commands	155
9	Programming Flash Memory	157
9.1	Introduction	157
9.2	Connecting to a Target	158
9.3	Testing Flash Workspace	163
	Reading and Writing Memory	163
9.4	Configuring Registers	164
9.5	Using the Flash Programmer View	165
9.6	Flash Configuration Tab	166
	9.6.1 Selecting a Flash Driver	166

9.6.2	Configuring Flash Memory Bounds	167
9.6.3	Configuring RAM Workspace	168
9.6.4	Setting Timeouts	168
9.7	Flash Programming Tab	168
9.7.1	Erasing and Programming Flash	169
9.7.2	Verifying Flash Contents	169
9.7.3	Running a Pre- or Post-Flash Script	170
9.7.4	Selecting Flash Sectors for Erasure	170
9.7.5	Manually Configuring Flash Memory Erasure Bounds	170
9.7.6	Adding Files	170
9.7.7	Removing Files	171
9.7.8	Converting Files To Wind River Flash Binary Format	171
9.7.9	Setting The Download Offset Of A File	173
9.7.10	Enabling A File For Download	174
9.8	Flash Memory/Diagnostics Tab	174
9.8.1	Viewing Memory	175
9.8.2	Running Diagnostic Tests	175
10	On-Chip Debugging for Linux	177
10.1	Introduction	177
10.2	Linux Virtual Memory Management Architecture	178
10.3	Connection Parameters	179
10.4	Emulator Configuration	183
10.5	MMUL Settings	183
10.6	Booting a Linux System with OCD	185
10.6.1	Standard Boot	185

10.6.2	OCD Boot	189
10.7	Boot Line Commands	192
10.8	Reverse-Engineering the Boot Line Parameters	195
10.9	Debugging the Linux Kernel	196
10.9.1	Debugging Linux Kernel Modules	196
	Kernel Module Detection	196
	Debugging the init() Function of a Module	197
10.10	Kernel Configuration	198
10.11	Debugging User Space Applications with OCD	198
10.11.1	Attaching to a Process	199
10.11.2	Debugging a Process	199
10.11.3	Setting Breakpoints	200
10.11.4	Thread-Qualified Breakpoints	200
10.11.5	Debugging the Beginning of a Process	200
10.11.6	Limitations	200
10.12	Linux Troubleshooting	201
11	Using the WDB Transparent Mode Driver	203
11.1	Introduction	203
11.2	Connecting Through the Transparent Mode Driver	204
11.3	Using the TMD With the Wind River ICE SX	206
11.3.1	Configuring Wind River ICE SX	206
11.3.2	Configuration Options	207
	Setting CF Options in the CF Options View	207
	Setting CF Options with Low-Level Commands	208
11.4	Configuring the Target Server	209

11.5	Moving On	218
12	Internal Software Trace	219
12.1	Overview	219
12.2	The Trace View	220
12.2.1	Trace View Buttons	220
	Collapsing and Expanding Fields	220
	Toggle Trace/Source view Auto-Sync	221
	Clear Trace Buffer	221
	Refresh Trace View	222
	Open Trace Rules Dialog	222
	Filter Visible Trace Events	224
	Save Output to File	224
12.3	Configuring Trace	225
12.3.1	PowerPC Trace Configuration Options	225
12.3.2	PA Semi Trace Configuration	228
12.4	Tracing Execution	231
12.4.1	Setting a Tracepoint	231
12.4.2	Tracing Execution	231
13	Using the CF Options View	233
13.1	Introduction	233
13.2	Connecting to a Target	234
13.3	Configuring the Target Connection	238
13.4	Changing CF Options in the CF Options View	240
13.5	Changing CF Options With Low-Level Commands	241
13.6	Resetting CF Options	242

14	Using Hardware Diagnostics	243
14.1	Introduction	243
14.2	Connecting to Your Target	244
14.3	Setting a Workspace	248
14.4	Hardware Diagnostic Tests	249
14.4.1	Simple RAM Test	249
14.4.2	Full RAM Tests	251
14.4.3	CRC Calculation	251
14.4.4	Scope Tests	253
	Read From Location	253
	Write To Location	253
	Write and Complement	253
	Write Rotating Value	253
	Write Then Read	253
14.4.5	Bus Tests	253
	Address Bus Test	253
	Data Bus Test	254
15	OCD Statistical Code Profiling	255
15.1	Introduction	255
15.2	Connecting to the Target	256
15.3	Creating a Project	266
15.3.1	Downloading the Sample Code	271
15.4	Profiling Your Code	272
15.4.1	Profiling Selected Functions	278
15.4.2	Browsing Functions in Source	279
15.4.3	Updating the Profile Data	279
15.4.4	Removing Functions	279

16	Using the Cache View	281
16.1	Introduction	281
16.2	Connecting to the Target	282
16.3	Creating a Project	293
16.3.1	Downloading the Sample Code	298
16.4	Examining Cache	299
16.4.1	Instruction Cache	299
16.4.2	Data Cache	301
16.4.3	Interpreting the Cache View	301
16.5	Viewing Cache Source	303
16.6	Comparing Memory	303
16.7	Reconfiguring the Cache	305
16.8	Exporting Cache Information	305
16.9	Using Processors Without Cache Lines	306
	Instruction Cache	306
	Data Cache	307
17	Uploading Target Memory to a Binary File	309
17.1	Introduction	309
17.2	Uploading Memory	309
17.3	Comparing Memory	311
18	Using the Instruction Set Simulator	313
18.1	Introduction	313
18.2	Connecting to the Simulator	314

19	Programming a VxWorks Boot ROM into Flash Memory	321
19.1	Introduction	321
19.2	Configuring The Target	322
19.2.1	Making Physical Connections	322
19.2.2	Testing Memory and Breakpoints	323
	Reading and Writing Memory	323
	Testing Breakpoints	324
19.3	Flashing the Boot ROM	324
19.3.1	Playing a Register File	325
19.3.2	Setting Up Chip Select 0 and Programming the Reset Configuration Word	325
19.3.3	Unlocking Flash	326
19.3.4	Programming Flash	326
20	Programming a Linux Bootloader into Flash Memory	333
20.1	Introduction	333
20.2	Installing the Bootloader	334
20.3	Configuring and Building the Bootloader	334
20.3.1	Configuring and Building the Bootloader Manually	335
	Modifying the boardConfig.h File	335
	Building a Downloadable U-Boot File	336
20.4	Configuring the Target	338
20.4.1	Making Physical Connections	338
20.4.2	Testing Memory and Breakpoints	339
	Reading and Writing Memory	339
	Testing Breakpoints	339
20.5	Flashing the Bootloader	340
20.5.1	Playing a Register File	340

20.5.2	Setting Up Chip Select 0 and Programming the Reset Configuration Word	341
20.5.3	Unlocking Flash	341
20.5.4	Programming Flash	342
21	Downloading a Kernel Image Using a JTAG Connection	345
21.1	Introduction	345
21.2	Bypassing the Boot Line Address -- VxWorks	347
21.2.1	Manually Setting the BOOT_LINE_ADRS Location	348
21.2.2	Forcing the DEFAULT_BOOT_LINE	348
21.3	Bypassing the Boot Line Setup -- Linux	349
21.4	Downloading the Kernel Image	351
22	Kernel-Aware Debugging	357
22.1	Introduction	357
22.1.1	VxWorks 5.5	357
22.1.2	Linux	357
22.1.3	ThreadX	358
	Index	359

1

Introduction

[1.1 Document Overview](#) 1

[1.2 Other Resources](#) 2

1.1 Document Overview

This document is designed to help you understand the Wind River On-Chip Debugging solution. For On-Chip Debugging, Wind River provides the Wind River ICE SX and Wind River Probe emulators and the Wind River Workbench development suite. Together, these products provide a fully integrated hardware and software solution for board bring-up, flash programming, production, and testing.

Wind River emulators allow you to perform source-level debug activities such as watching memory and controlling large numbers of registers.

Wind River emulators let you control a target by using the On-Chip Debugging (OCD) services embedded in the microprocessor of that target. An emulator operates effectively as a standalone system, communicating with the OCD services resident in the microcode of the chip.

When you access the OCD services in a chip, you gain complete control of the microprocessor, and all interaction between the emulator and the target runs exclusively through the OCD connection. This means that the emulation system is

effective for the entire development process, even before board-level peripherals are stable.

This document describes common use cases for the Wind River ICE SX and Wind River Probe emulators, including:

- Setting up OS-independent projects.
- Debugging Linux targets.
- Working with registers and register groups.
- Programming flash memory.
- Creating, editing, and using board descriptor files.
- Debugging multiple cores.
- Tracing executing code.
- Performing statistical profiling analysis on executing code.
- Examining cache on your target.
- Flashing a Linux boot loader on your target.
- Flashing a VxWorks boot ROM on your target.
- Downloading an image to your target without using a boot ROM or boot loader.

This document provides a collection of tutorials for the operations described above, and provides step-by-step instructions on how to perform them using Wind River Workbench.

1.2 Other Resources

For information on the Wind River ICE SX and Wind River Probe, including hardware information, establishing communications with Wind River Workbench, and defining launch configurations, see the *Wind River ICE SX for Wind River Workbench Hardware Reference* or the *Wind River Probe for Wind River Workbench Hardware Reference*.

For information on low-level commands available for the Wind River ICE SX and Wind River Probe, see the *Wind River Workbench for On-Chip Debugging Command Reference*.

For information on configuration options for the Wind River ICE SX and Wind River Probe, see the *Wind River Workbench for On-Chip Debugging Configuration Options Reference*.

For information on Wind River Workbench, see the *Wind River Workbench User's Guide*.

2

Basic Operation: Debugging with a Project

- 2.1 Introduction 5
- 2.2 Connecting to the Target 6
- 2.3 Creating a Project 18
- 2.4 Debugging Code 25
- 2.5 Moving On 32

2.1 Introduction

This chapter provides a tutorial on basic operation of Wind River Workbench for On-Chip Debugging (OCD).

You can use Wind River Workbench to run and debug code either in combination with the Workbench project management utility, or without a project. This chapter assumes you are debugging with a Workbench project. For a tutorial on debugging without a Workbench project, see [3. *Basic Operation: Debugging Without a Project*](#).

This tutorial includes the following topics:

- Launching Wind River Workbench.
- Connecting to a Wind River emulator and a target processor.
- Creating a sample project.

- Building a sample project.
- Downloading code to the target.
- Debugging code running on the target.

2.2 Connecting to the Target

First, open Workbench according to the method for your host computer.

Linux/Solaris Hosts

From your installation directory, issue the command

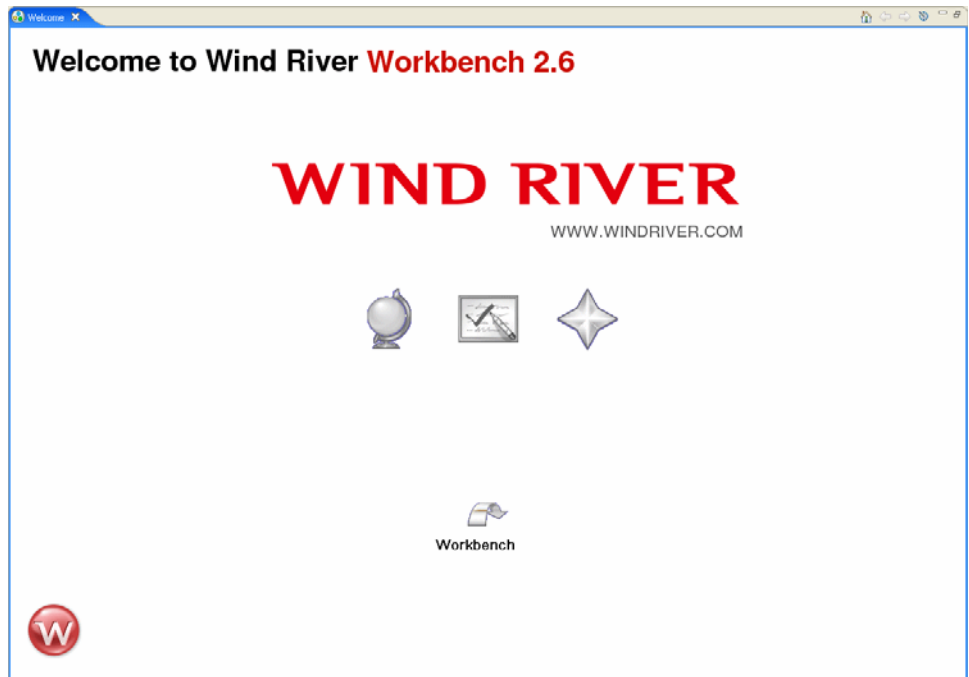
```
$ ./startWorkbench.sh
```

Windows Hosts

Select **Start > All Programs > Wind River > Wind River Workbench** *version*.

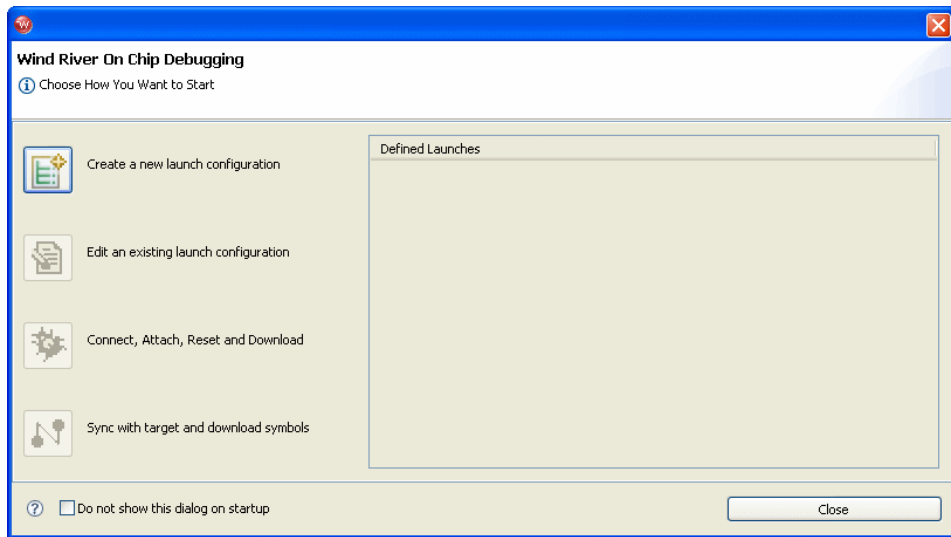
On Windows hosts, Workbench prompts you to specify a workspace location. Linux and Solaris hosts use the default location *installDir/workspace*.

The **Welcome** screen appears.



1. Click **Workbench**.

Workbench opens, displaying the **Quick Target Launch** dialog.



Workbench saves all information regarding a particular emulator-target connection in a *launch configuration*. The launch configuration includes such configuration information as emulator type, target processor family, target CPU, and host-PC interface port, plus any port parameters such as IP address (if using a Wind River ICE SX), serial number (if using a Wind River Probe) or baudrate.

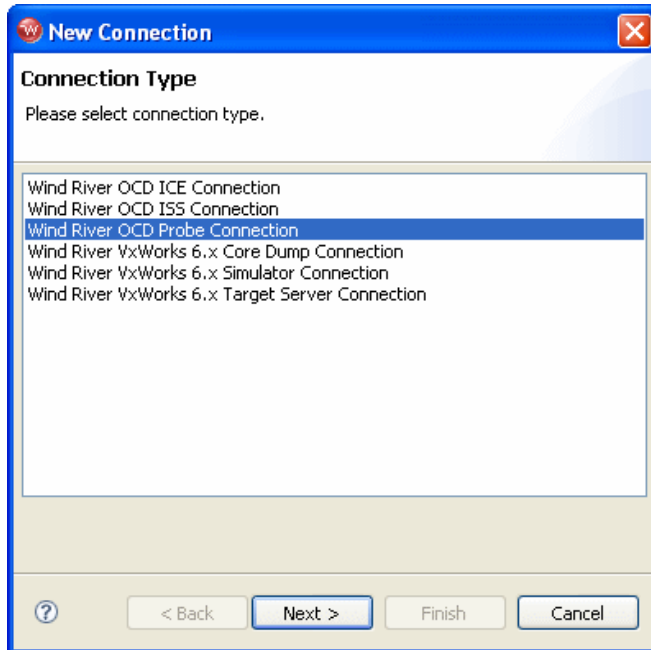
Once you have defined an emulator-target connection, Workbench saves it in the **workspace** folder. The next time you open Workbench, that launch configuration will appear in the **Defined Launches** area of the **Quick Target Launch** dialog, and you can return to it by highlighting it and clicking **Connect, Attach, Reset and Download**.

The **Quick Target Launch** dialog opens automatically any time you launch Workbench. If you do not want to use the **Quick Target Launch**, select the **Do not show this dialog on startup** checkbox and click **Close**. You can open the **Quick Target launch** dialog at any time by clicking the **OCD Quick Launch** button in the Workbench toolbar.

Since this is the first time you have opened Workbench, there are no existing launch configurations, and you must create one.

2. Select **Create a new launch configuration**.

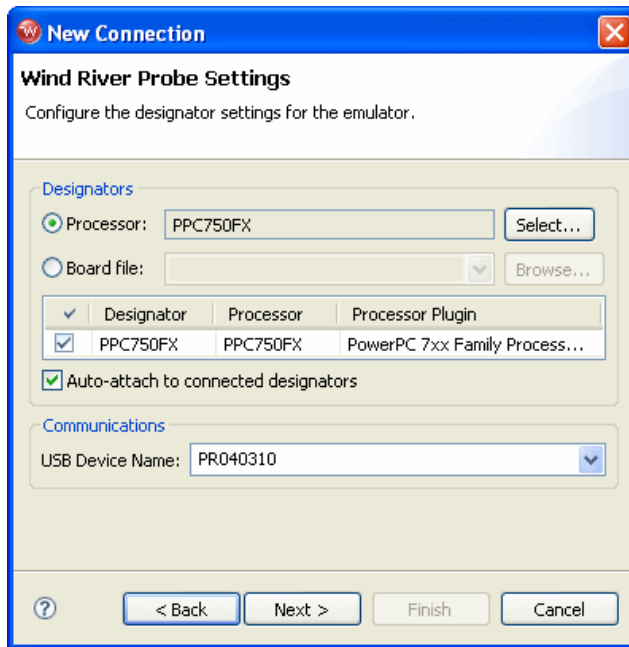
The **Connection Type** dialog appears.



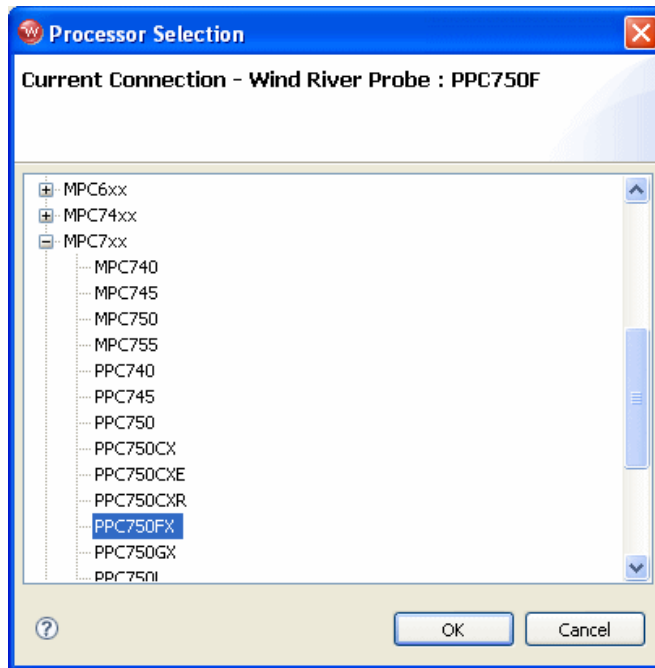
3. Select your connection type (Wind River ICE SX, Wind River Probe, or Wind River Instruction Set Simulator) and click **Next**.

For instance, the examples in this tutorial show a Wind River Probe emulator connected to a Wind River PPMC750FX target, so you would select **Wind River OCD Probe connection**.

The **Processor Selection** dialog appears.



4. Click **Select**. From the list that appears, expand **MPC7xx** and select **PPC750FX**.

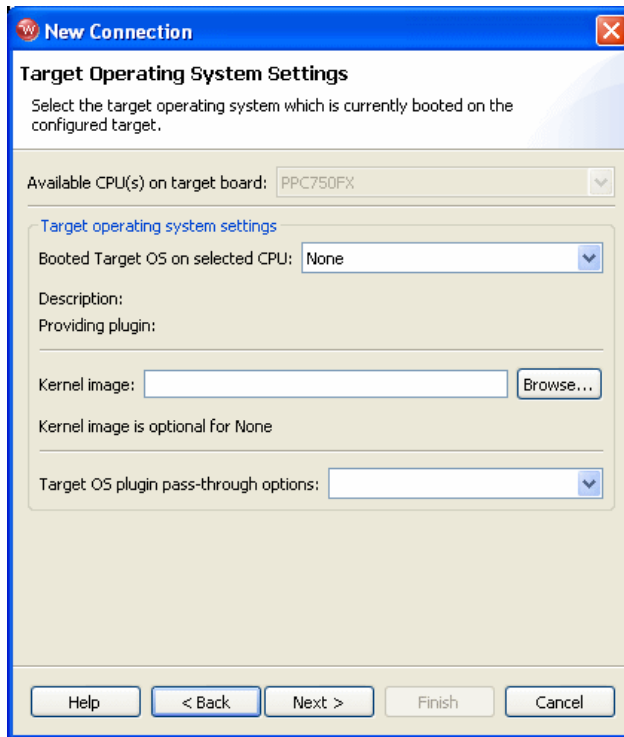


5. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

6. Click **Next**.

The **Target Operating System Settings** dialog appears.



7. In the **Booted Target OS on selected CPU** field, select the operating system that is running on your target processor. The default is **None**.
8. Next to the **Kernel Image** field, click **Browse** to navigate to the kernel image you wish to specify. If you selected **None** in the previous step, you do not need to specify a kernel image.
9. If you are using a Linux plug-in, specify the pass-through options in the **Target OS Pass-Through Options** field. If you are not using a Linux plug-in, skip this step.

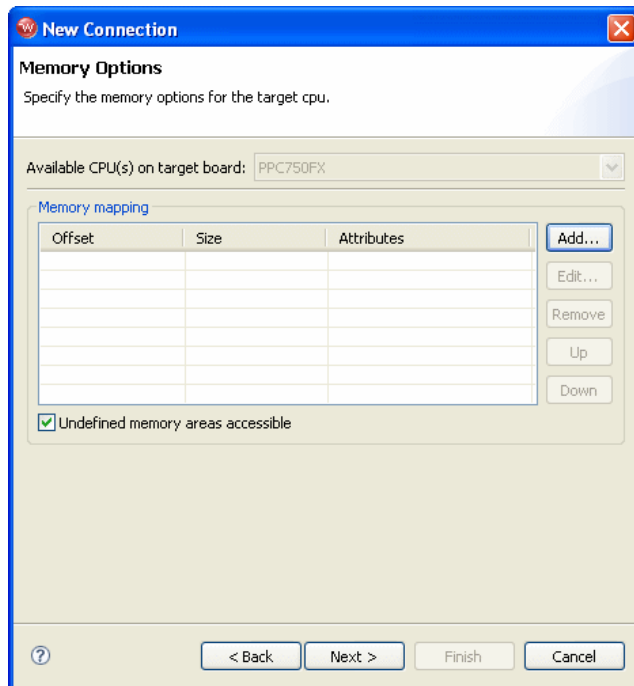
Options are passed as pairs in the format *name='value'*. Separate options with a comma. The following options are available:

- **notasklist=1** : Never fetch process list.
- **noautomodules=1** : Do not plant internal breakpoints to do automatic kernel module load/unload detection. When this option is specified, you must manually refresh to see an updated module list.

- **noloadcheck=1** : Do not issue gophers until the hardware breakpoint is used to detect kernel load triggers. This option is for “sensitive” boards that don’t accept access until the kernel loads and sets up memory mapping.
- **loaddetectloc=symbol or address**: Set the hardware breakpoint used to detect kernel load at *symbol* (for example, **loaddetectloc=start_kernel**) or *address* (for example, **loaddetectloc=0x1000**). If you do not specify a symbol or address, Workbench uses a default. For most architectures the default is **start_kernel**; for PowerPC targets, the default is **0x0**.

10. Click **Next**.

The **Memory Options** dialog appears.

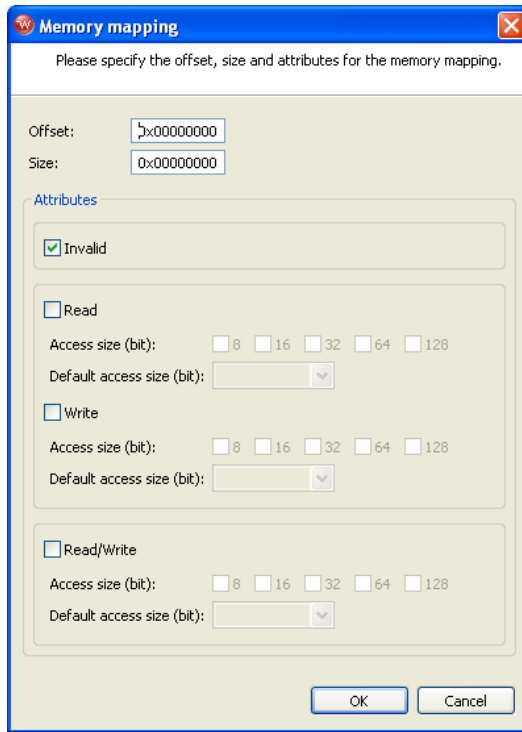


Use the **Memory Options** dialog to specify how memory on the target is partitioned, and what the attributes of the particular memory regions are.



NOTE: The **Memory Options** dialog is only necessary for Linux or other non-VxWorks target operating systems.

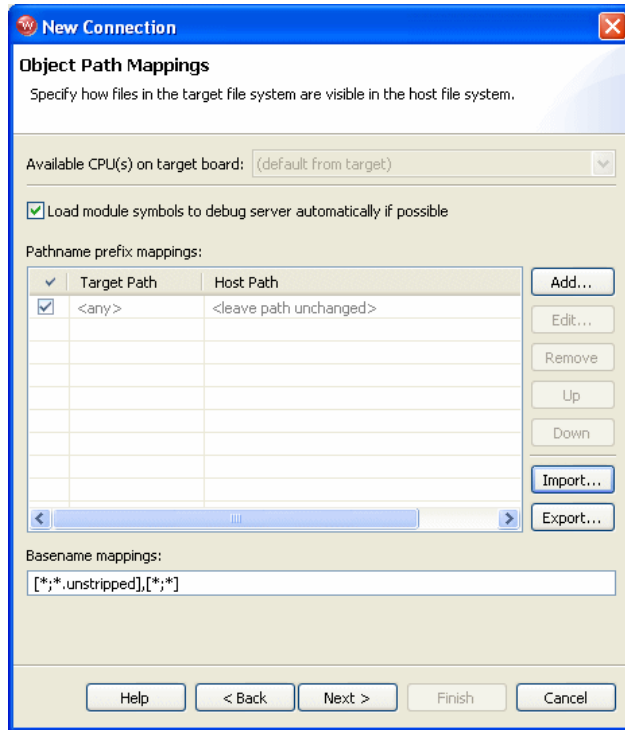
To specify an area of memory, click **Add**.
The **Set Memory Map** dialog appears.



Use the **Set Memory Map** dialog to specify which memory areas are read-only, read-write, or write-only, and to specify the access width Workbench should use to read the data from those regions.

11. Click Next.

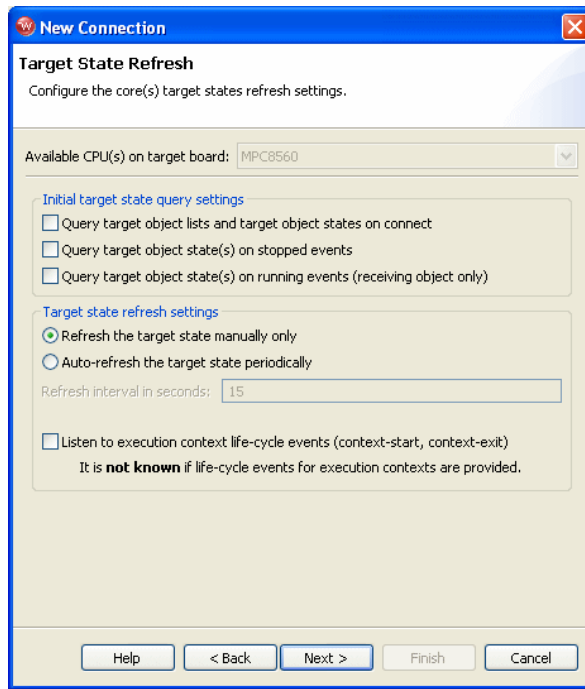
The **Object Path Mappings** dialog appears.



Use the **Object Path Mappings** dialog to specify how files in the target file system are visible in the host file system.

12. To add a host or target path, click **Add...** and type the path in the dialog that appears.
13. Click **Next**.

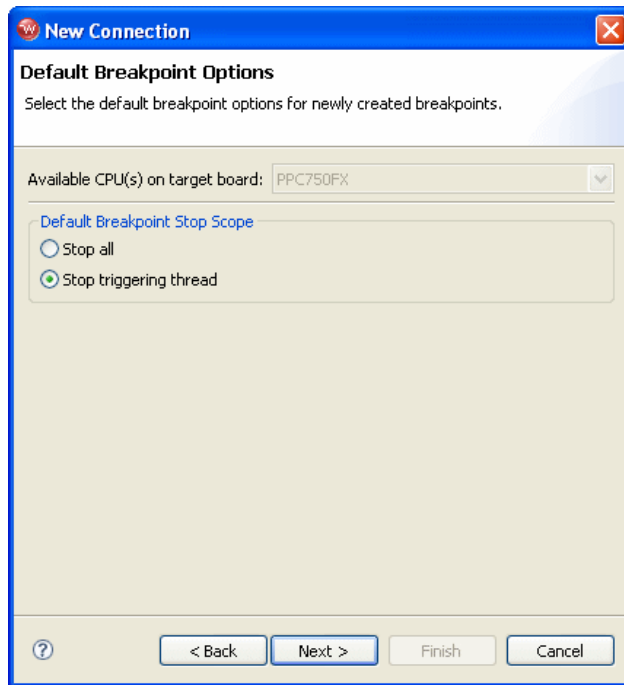
The **Target State Refresh** dialog appears.



Use the **Target State Refresh** dialog to configure the target state query and target state refresh settings on your target processor.

14. Click **Next**.

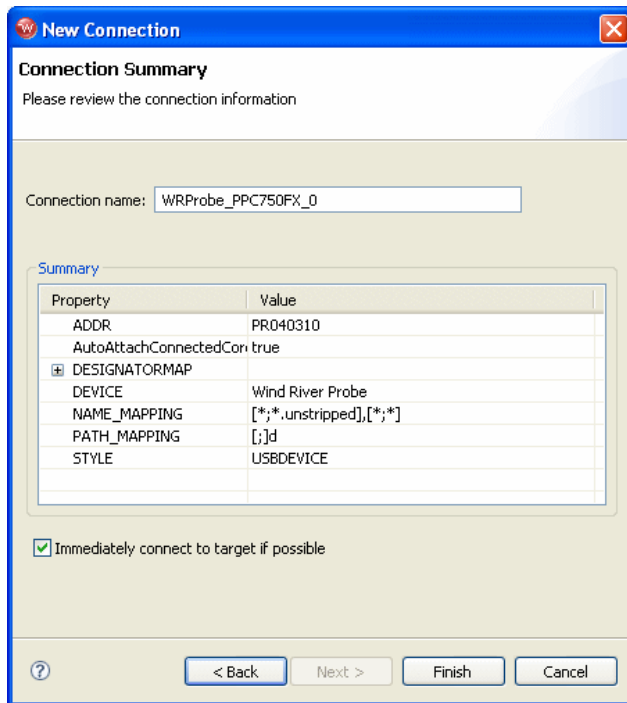
The **Default Breakpoint Options** dialog appears.



Use this dialog to set default breakpoint options for newly created breakpoints.

15. Click **Next**.

The **Connection Summary** dialog appears. Inspect the displayed values to make sure they are correct.



16. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

Workbench creates a target connection called **WRProbe_PPC750FX** in the **Target Manager** view.

The **Reset and Download** view appears.

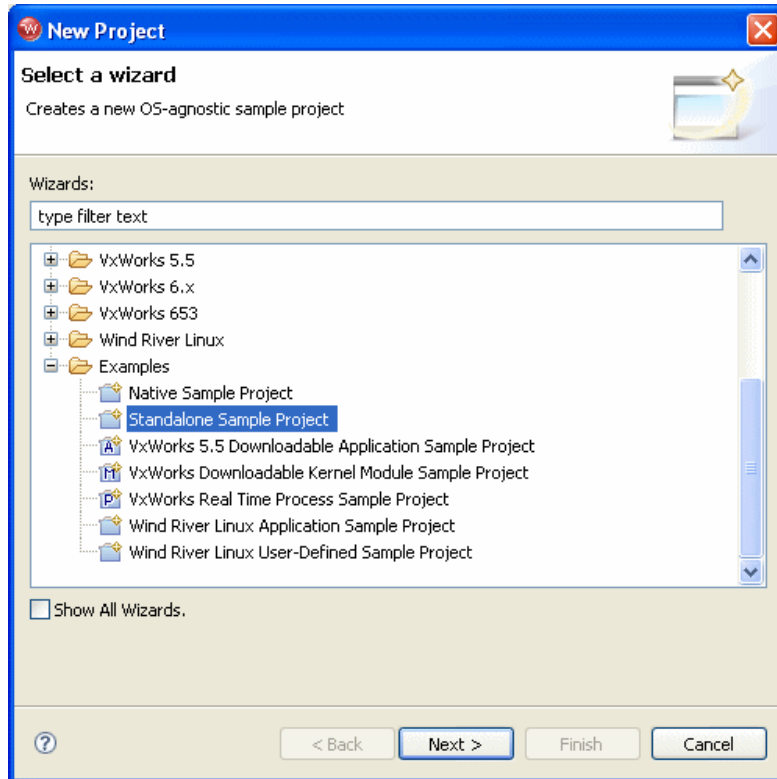
17. Click **Close** in the **Reset and Download** view.

2.3 Creating a Project

This tutorial uses the C Demonstration Program, which is included in your Workbench installation.

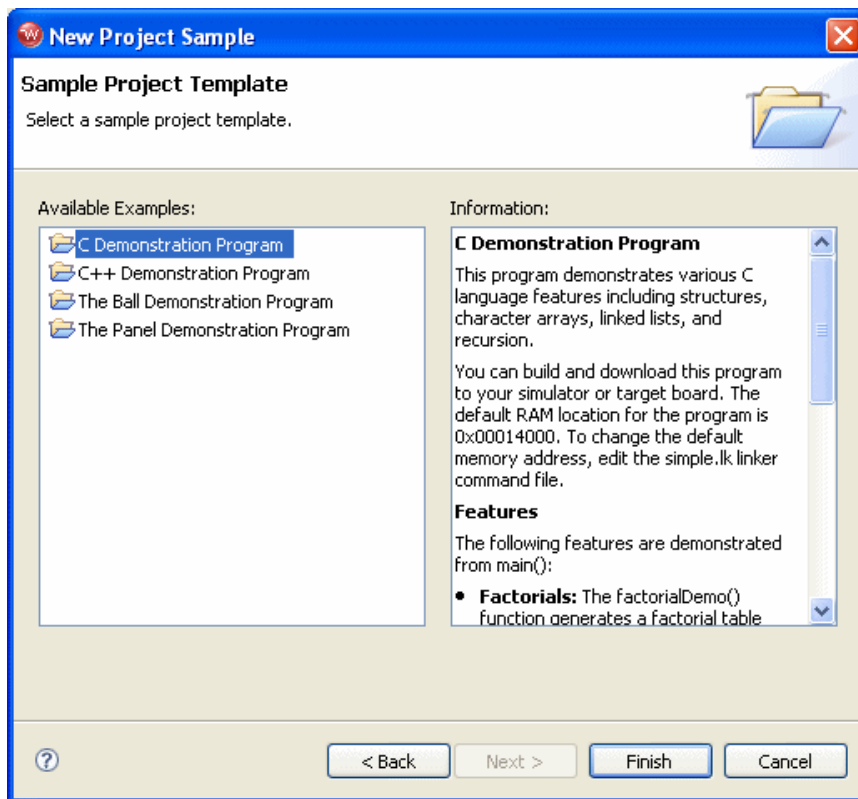
1. In the Workbench toolbar, select **File > New > Project**.

The **New Project** wizard appears.



2. Expand the **Examples** folder and select **Standalone Sample Project**.
3. Click **Next**.

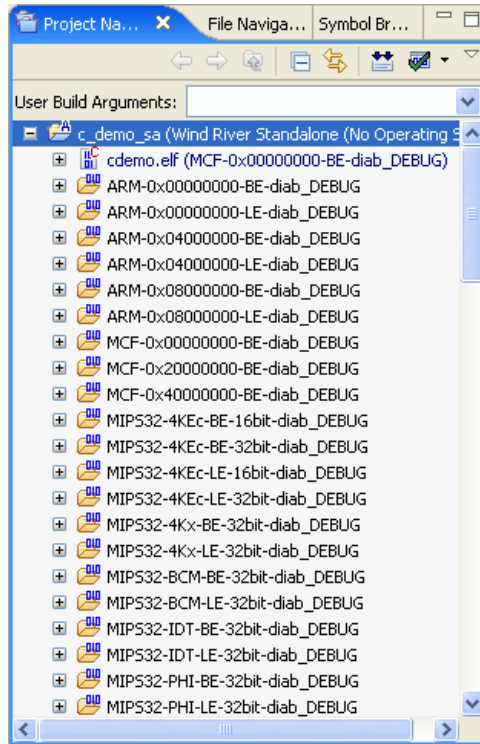
A sample project template appears.



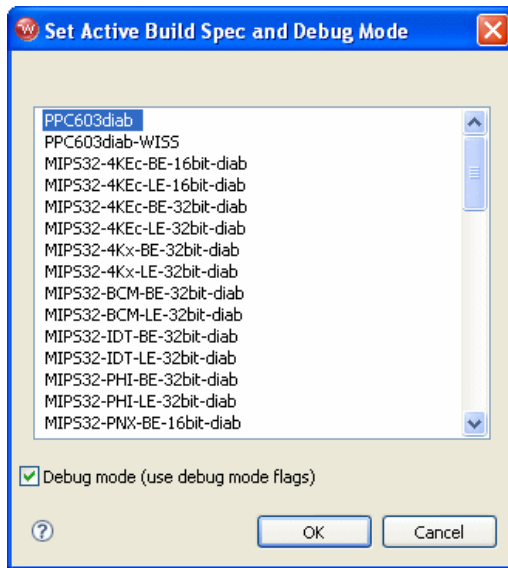
4. Select **C Demonstration Program** and click **Finish**.

Workbench creates the sample project in the default workspace folder and opens the **Application Development** perspective.

5. In the **Project Navigator** view, expand the **c_demo_sa** project.

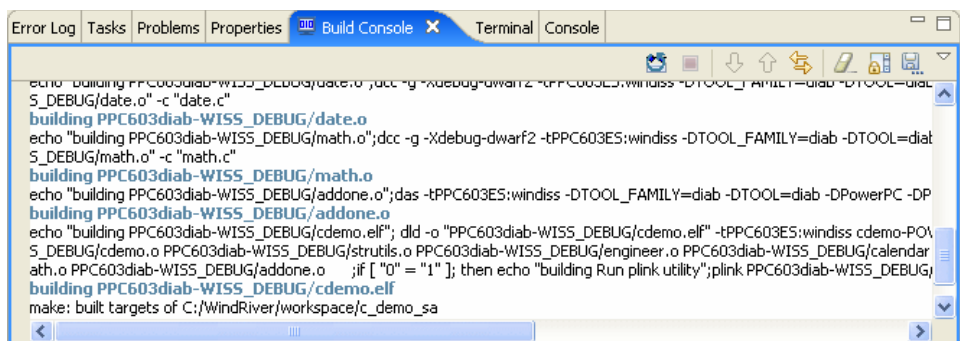


6. To build the sample project for use with a PowerPC target, right-click on the `c_demo_sa` top-level folder and select **Build Options > Set Active Build Spec**. The **Set Active Build Spec and Debug Mode** dialog appears.



7. Scroll to the top and highlight **PPC603diab**.
8. Select **Debug mode (use debug mode flags)** so Workbench will generate symbolic debug information.
9. Click **OK**.
10. Right-click on the project name and select **Rebuild Project**.

Workbench builds the sample project. The results of the project build appear in the **Build Console** view.

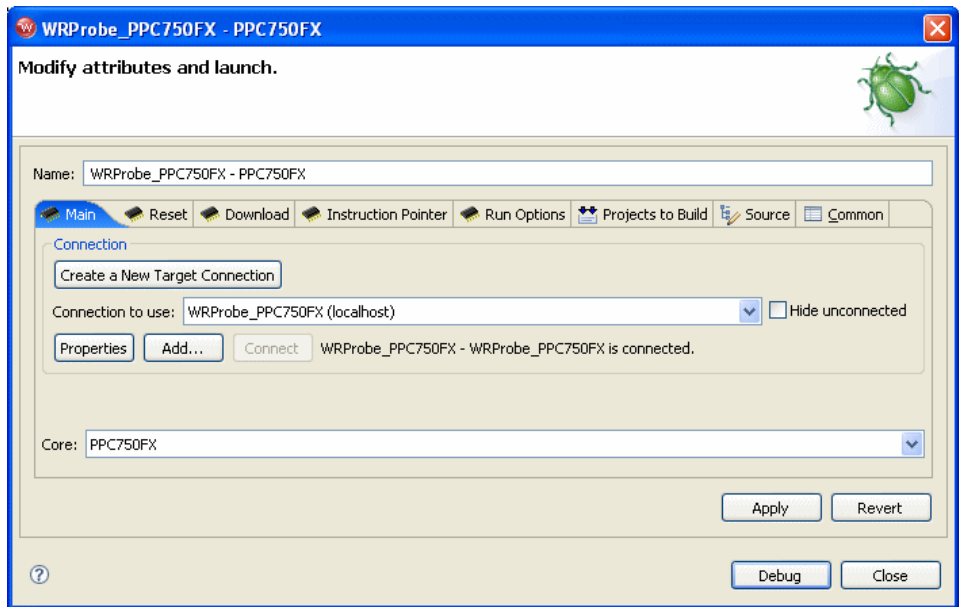


2.3.1 Downloading the Sample Code

To run the sample code, use the following steps:

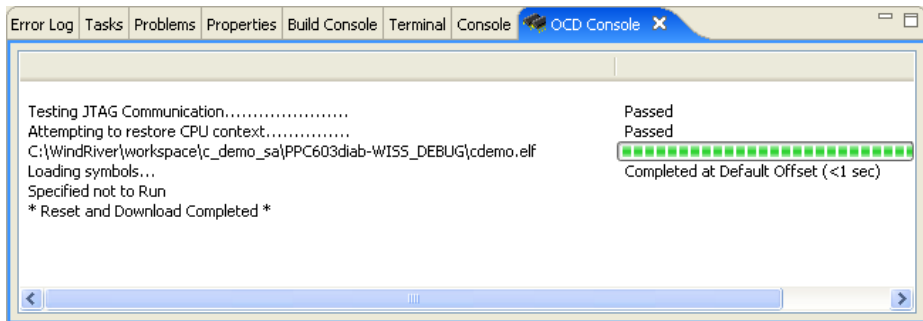
1. In the **Target Manager**, highlight the target connection name **WRProbe_PPC750FX**.
2. In the **Project Navigator** view, right-click on **cdemo.elf** and select **Reset and Download**.

The **Reset and Download** view appears.

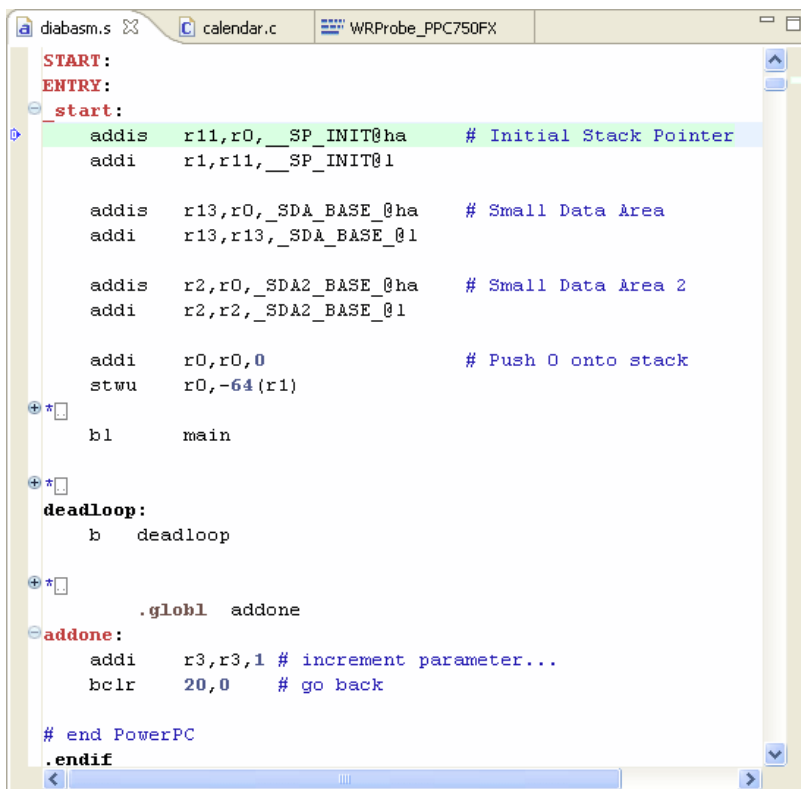


3. Leave all settings at their defaults and click **Debug**.

The **OCD Console** view opens.



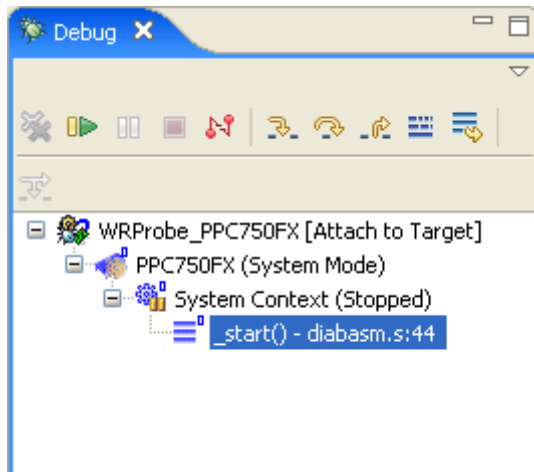
The **OCD Console** view shows the progress of the download operation.
The Editor opens showing the Program Counter set at the beginning of the application code.



You are now ready to run and debug the application.

2.4 Debugging Code

Use the **Debug** view to monitor, control, and manipulate the processes and tasks that you are actively debugging. The **Debug** view shows only the processes that are currently under debugger control.



2.4.1 Monitoring Processes

When you start processes under debugger control, or attach the debugger to running processes, they appear in the **Debug** view labeled with unique colors and numbers. You can change the color assigned to a process or thread by right-clicking the process or thread and selecting **Color** > *specific color*.

2.4.2 Stepping Through Code

The Editor shows the source file **diabasm.s**, showing the C Demonstration Project initialization assembly.

In the **Debug** view, click the **Step Into** button.

The Program Counter moves to the second assembly instruction. If you open the **Memory** view or the **Registers** view, you can see them update memory and register values as you step through instructions.

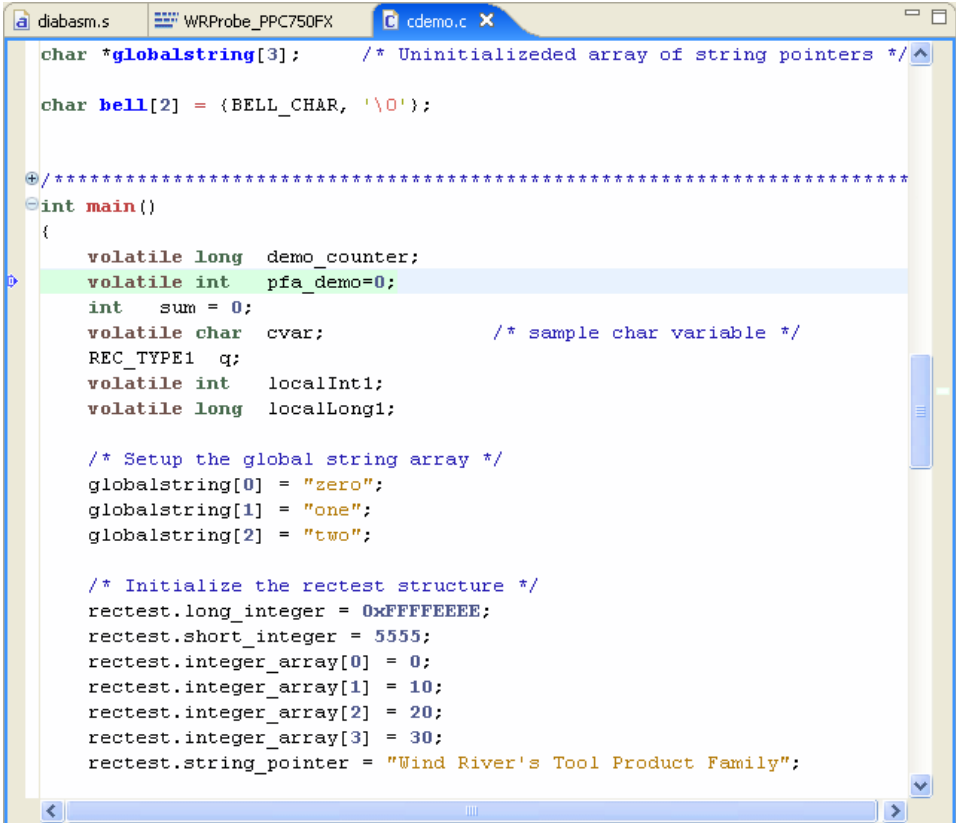
Click the **Step Into** button seven more times, to step through all the initialization code and reach the first branch instruction:

```
bl main
```

This is where the application branches out of assembly into C code.

Click the **Step Into** button again.

The application branches into **main()** and the Editor opens the source file **cdemo.c**.



```
diabasm.s  WRProbe_PPC750FX  cdemo.c x
char *globalstring[3]; /* Uninitialized array of string pointers */
char bell[2] = {BELL_CHAR, '\0'};

/* *****
int main()
{
    volatile long demo_counter;
    volatile int pfa_demo=0;
    int sum = 0;
    volatile char cvar; /* sample char variable */
    REC_TYPE1 q;
    volatile int localInt1;
    volatile long localLong1;

    /* Setup the global string array */
    globalstring[0] = "zero";
    globalstring[1] = "one";
    globalstring[2] = "two";

    /* Initialize the rectest structure */
    rectest.long_integer = 0xFFFFFFFF;
    rectest.short_integer = 5555;
    rectest.integer_array[0] = 0;
    rectest.integer_array[1] = 10;
    rectest.integer_array[2] = 20;
    rectest.integer_array[3] = 30;
    rectest.string_pointer = "Wind River's Tool Product Family";
```

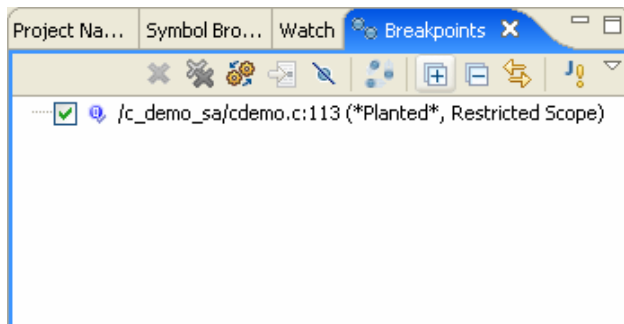

2.4.3 Setting a Software Breakpoint

Breakpoints allow you to stop a running program at particular places in the code or when specific conditions exist.

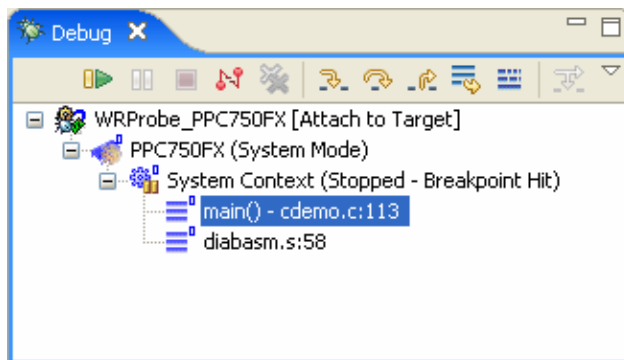
In the left ruler of the Editor (the gutter), double-click to the left of the source line

```
globalstring[2] = "two";
```

This sets a software breakpoint on that source line. The breakpoint appears in the **Breakpoints** view.



In the **Debug** view, click the **Resume** button. The program runs until it hits the breakpoint. The **System Context** changes to **Stopped -- Breakpoint Hit**.



Breakpoint information also appears in the **OCD Command Shell**:

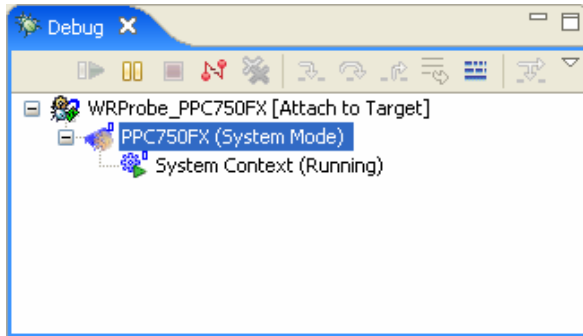
```
>RUN>
```

```
!BREAK! - [msg12000] Software breakpoint; PC = 0x00014074 [EVENT Taken]  
>BKM>
```

2.4.4 Running a Program

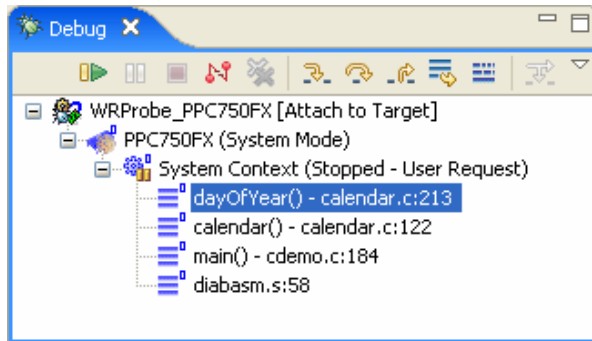
To run your downloaded program, click **Resume** in the **Debug** view. The program will run until it hits a breakpoint. If there are no breakpoints or interrupts, the program will run to completion or until you click **Suspend**.

When the program is running, the System Context changes to **Running**, and a **>RUN>** prompt appears in the **OCD Command Shell**.



If there are no breakpoints, you can stop the program by clicking the **Suspend** button in the **Debug** view or by entering the **HA** command at the **>RUN>** prompt in the **OCD Command Shell**.

The Editor updates to show the current location of the Program Counter and the **System Context** in the **Debug** view changes to **Stopped -- User Request**.



2.4.5 Stepping Through a Program

To single-step without going into other subroutines, click **Step Over** instead of **Step Into**.

While stepping through a program, you may conclude that the problem you are interested in lies in the current subroutine's caller, rather than at the stack level where your process is suspended. In this situation, if you click **Step Return**, execution continues until the current subroutine completes, then the debugger regains control in the calling statement.

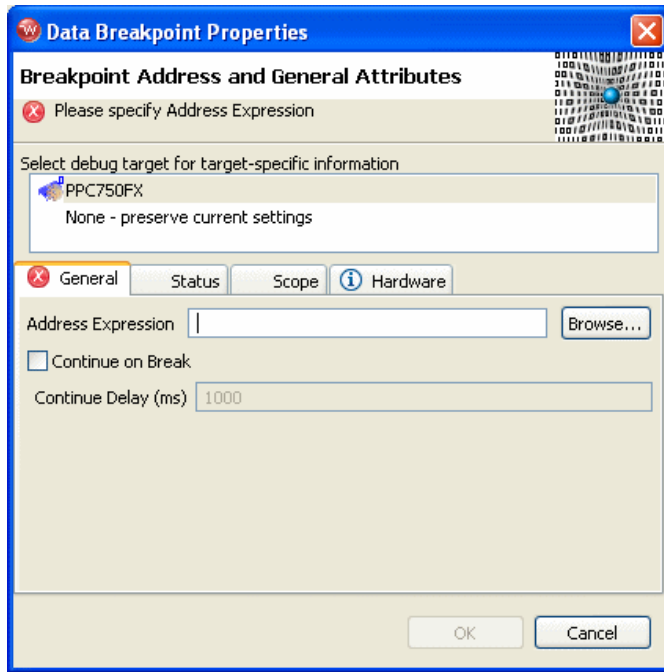
2.4.6 Setting a Hardware Breakpoint

The availability of hardware breakpoints varies by architecture. You can only set as many hardware breakpoints as there are debug registers available on your target.

Once a hardware breakpoint is trapped, the debugger will behave in the same way as for a standard breakpoint and stop for user interaction.

In the **Breakpoints** view, click on the **Menu** button and select **Add Data Breakpoint**.

The **Data Breakpoint** dialog appears.



If an error message appears, you may have exceeded the number of allowed hardware breakpoints (four for most targets). Right-click in the **Breakpoints** view and select **Remove All**. Then select **Menu > Add Data Breakpoint** again.

If an error message still appears, your target may not support hardware breakpoints.

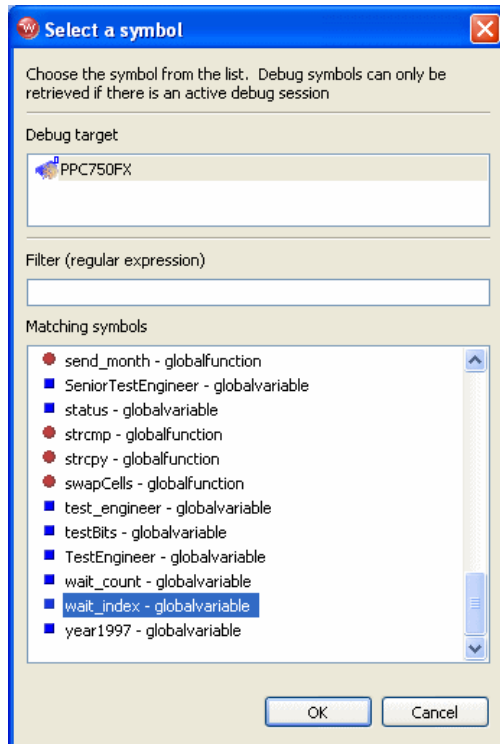
You can use data hardware breakpoints to find out which routines are modifying a specific variable.

The **Address Expression** can be a symbol or a specific address in hex. You can use the address **0x0** in the **Address Expression** field to set a data hardware breakpoint to catch null pointers. You can set the **Address Expression** field to an address in the stack area to set a data hardware breakpoint to find out if the stack grew to that point.

The following example sets a symbol in the **Address Expression** field.

1. Click **Browse**.

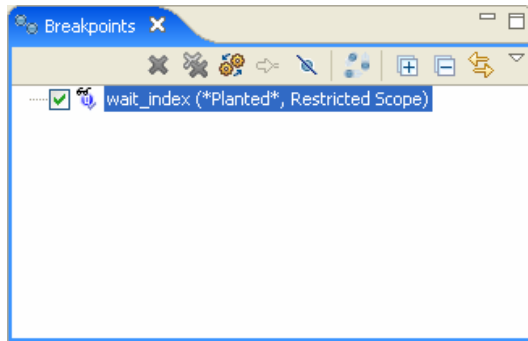
The **Select Symbol** dialog appears, showing a list of available symbols that can take a hardware breakpoint.



2. Scroll down and highlight the symbol **wait_index**.
3. Click **OK**.

The global variable **wait_index** is now the address for the data hardware breakpoint.

The hardware breakpoint on **wait_index** appears in the **Breakpoints** view.



In the **Debug** view, click **Resume**.

The program runs until it hits the hardware breakpoint. Workbench halts the processor when it locates **wait_index** and displays that source line in the Editor.

2.4.7 Disconnecting and Terminating Processes

Disconnecting from a process or core detaches the debugger, but leaves the process or core in its current state.

Terminating a process actually kills the process on the target.



NOTE: If the selected target supports terminating individual threads, you can select a thread and terminate only that thread.

2.5 Moving On

For descriptions of other features of Wind River Workbench for On-Chip Debugging, such as code profiling, code tracing, and so on, see the relevant chapters in this document.

3

Basic Operation: Debugging Without a Project

- 3.1 Introduction 33
- 3.2 Connecting to the Target 34
- 3.3 Downloading Code 46
- 3.4 Debugging Code 53
- 3.5 Moving On 61

3.1 Introduction

This chapter provides a tutorial on basic operation of Wind River Workbench for On-Chip Debugging (OCD).

You can use Wind River Workbench to run and debug code either in combination with the Workbench project management utility, or without a project. This chapter assumes you are debugging without using a Workbench project. For a tutorial on debugging without a Workbench project, see [2. *Basic Operation: Debugging with a Project*](#).

This tutorial includes the following topics:

- Launching Wind River Workbench.
- Connecting to a Wind River emulator and a target processor.
- Downloading code to the target.

- Debugging code running on the target.

3.2 Connecting to the Target

First, open Workbench according to the method for your host computer.

Linux/Solaris Hosts

From your installation directory, issue the command

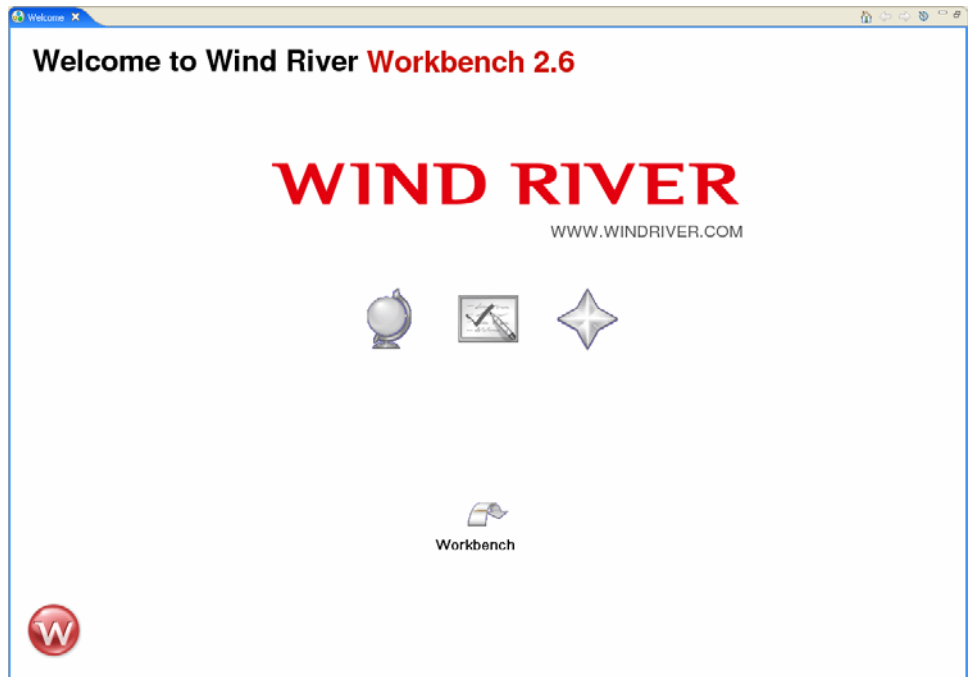
```
$ ./startWorkbench.sh
```

Windows Hosts

Select **Start > All Programs > Wind River > Wind River Workbench** *version*.

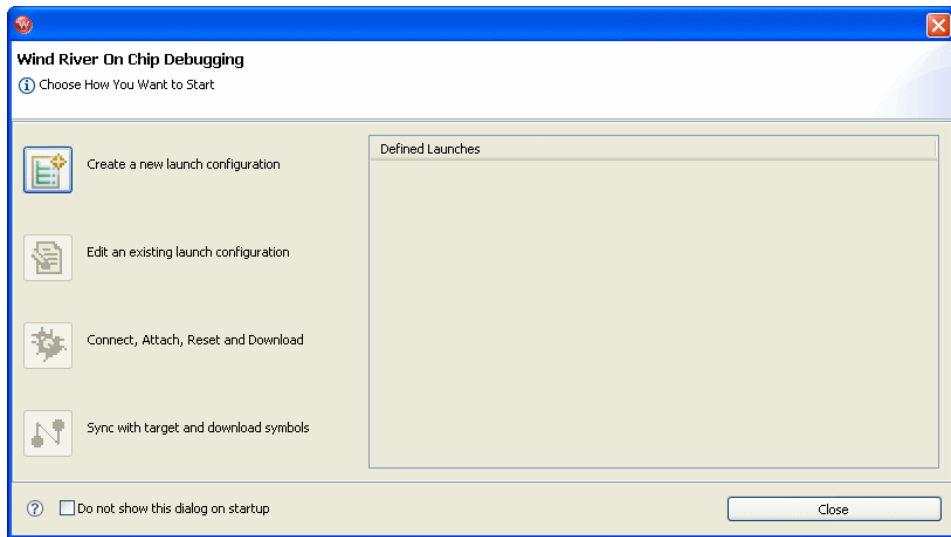
On Windows hosts, Workbench prompts you to specify a workspace location. Linux and Solaris hosts use the default location *installDir/workspace*.

The **Welcome** screen appears.



1. Click **Workbench**.

Workbench opens, displaying the **Quick Target Launch** dialog.



Workbench saves all information regarding a particular emulator-target connection in a *launch configuration*. The launch configuration includes such configuration information as emulator type, target processor family, target CPU, and host-PC interface port, plus any port parameters such as IP address (if using a Wind River ICE SX), serial number (if using a Wind River Probe) or baudrate.

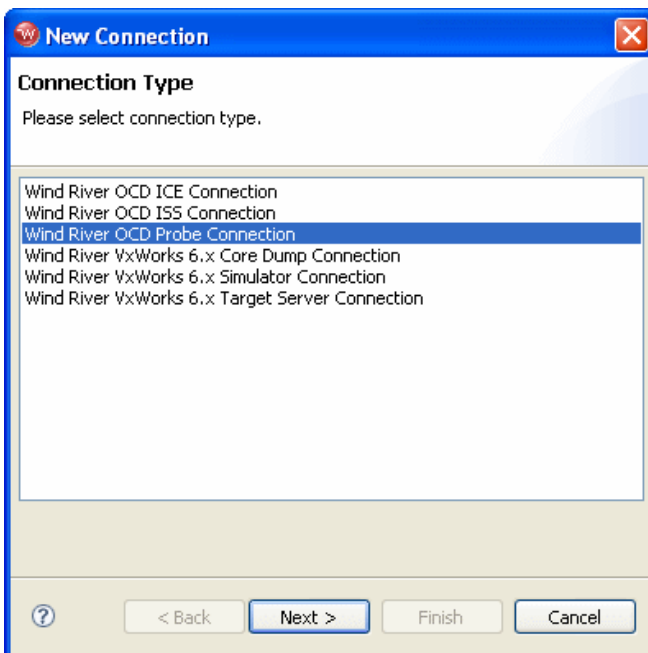
Once you have defined an emulator-target connection, Workbench saves it in the **workspace** folder. The next time you open Workbench, that launch configuration will appear in the **Defined Launches** area of the **Quick Target Launch** dialog, and you can return to it by highlighting it and clicking **Connect, Attach, Reset and Download**.

The **Quick Target Launch** dialog opens automatically any time you launch Workbench. If you do not want to use the **Quick Target Launch**, select the **Do not show this dialog on startup** checkbox and click **Close**. You can open the **Quick Target launch** dialog at any time by clicking the **OCD Quick Launch** button in the Workbench toolbar.

Since this is the first time you have opened Workbench, there are no existing launch configurations, and you must create one.

2. Select **Create a new launch configuration**.

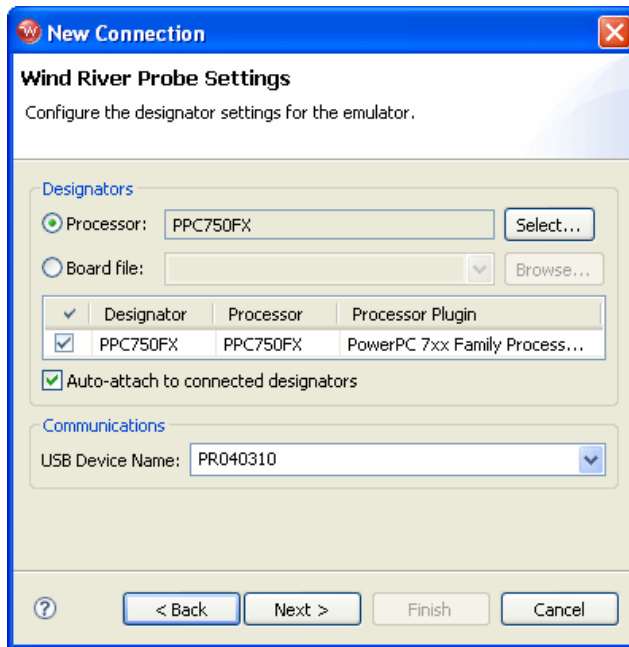
The **Connection Type** dialog appears.



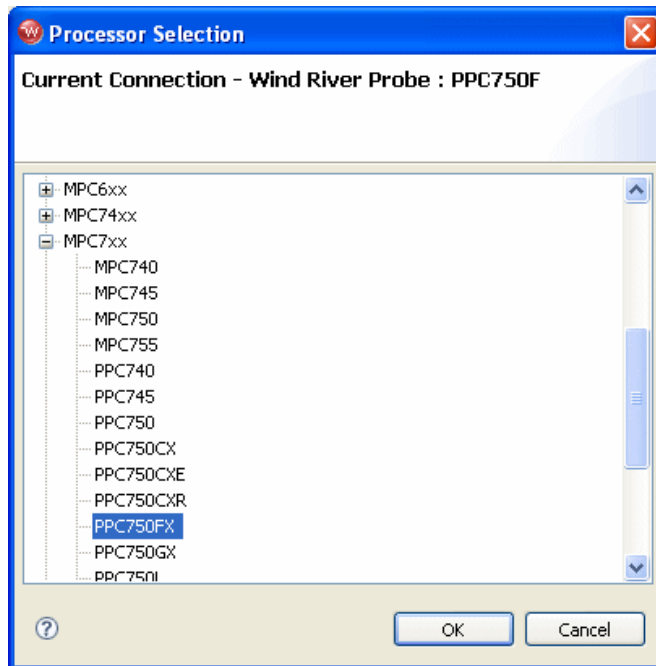
3. Select your connection type (Wind River ICE SX, Wind River Probe, or Wind River Instruction Set Simulator) and click **Next**.

For instance, the examples in this tutorial show a Wind River Probe emulator connected to a Wind River PPMC750FX target, so you would select **Wind River OCD Probe connection**.

The **Processor Selection** dialog appears.



4. Click **Select**. From the list that appears, expand **MPC7xx** and select **PPC750FX**.

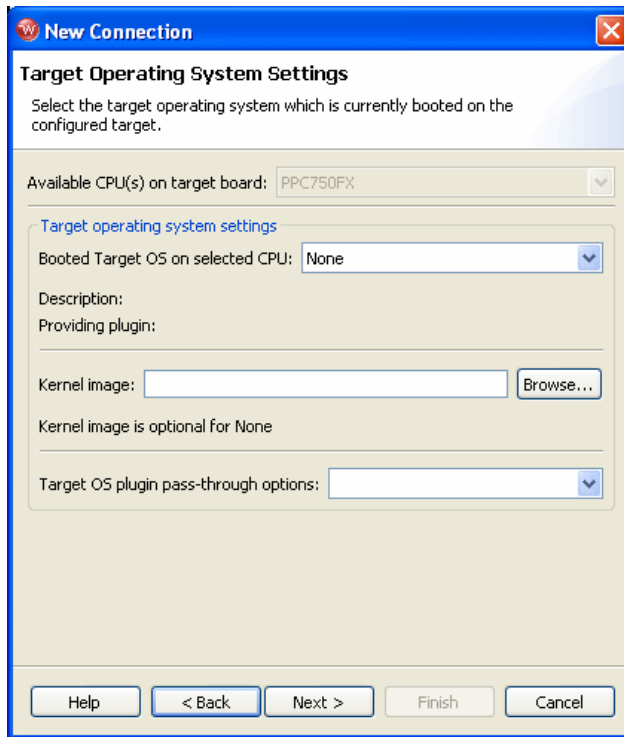


5. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

6. Click **Next**.

The **Target Operating System Settings** dialog appears.



7. In the **Booted Target OS on selected CPU** field, select the operating system that is running on your target processor. The default is **None**.
8. Next to the **Kernel Image** field, click **Browse** to navigate to the kernel image you wish to specify. If you selected **None** in the previous step, you do not need to specify a kernel image.
9. If you are using a Linux plug-in, specify the pass-through options in the **Target OS Pass-Through Options** field. If you are not using a Linux plug-in, skip this step.

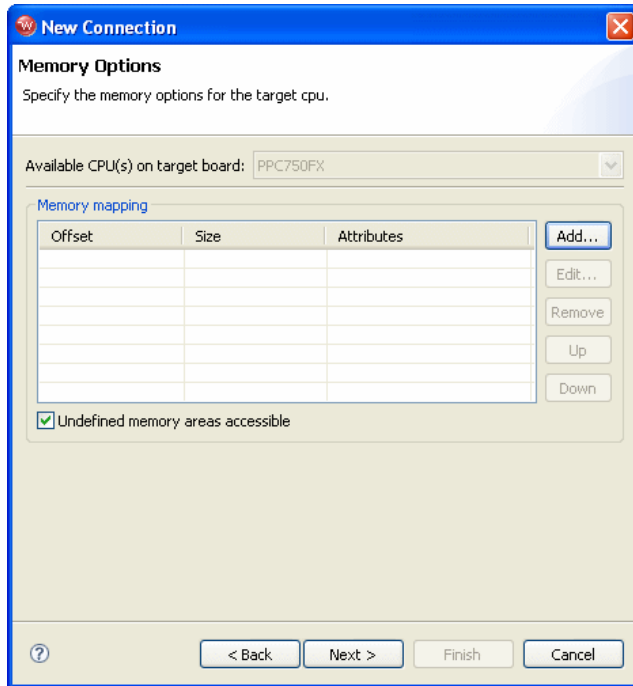
Options are passed as pairs in the format *name='value'*. Separate options with a comma. The following options are available:

- **notasklist=1** : Never fetch process list.
- **noautomodules=1** : Do not plant internal breakpoints to do automatic kernel module load/unload detection. When this option is specified, you must manually refresh to see an updated module list.

- **noloadcheck=1** : Do not issue gophers until the hardware breakpoint is used to detect kernel load triggers. This option is for “sensitive” boards that don’t accept access until the kernel loads and sets up memory mapping.
- **loaddetectloc=symbol or address**: Set the hardware breakpoint used to detect kernel load at *symbol* (for example, **loaddetectloc=start_kernel**) or *address* (for example, **loaddetectloc=0x1000**). If you do not specify a symbol or address, Workbench uses a default. For most architectures the default is **start_kernel**; for PowerPC targets, the default is **0x0**.

10. Click **Next**.

The **Memory Options** dialog appears.

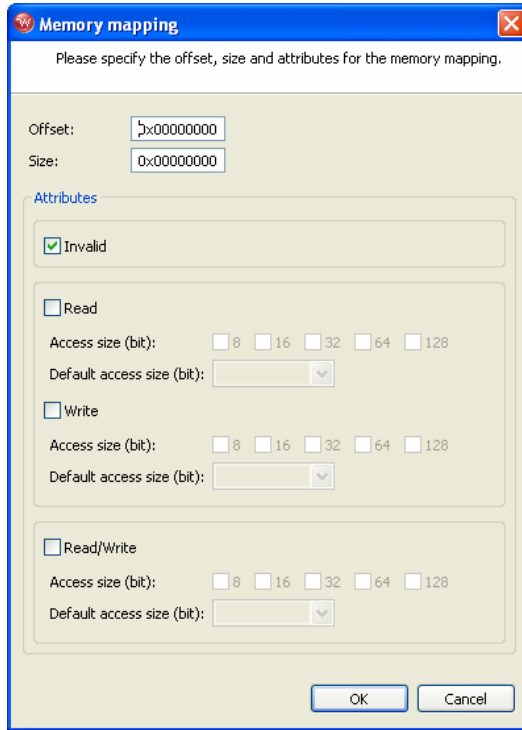


Use the **Memory Options** dialog to specify how memory on the target is partitioned, and what the attributes of the particular memory regions are.



NOTE: The **Memory Options** dialog is only necessary for Linux or other non-VxWorks target operating systems.

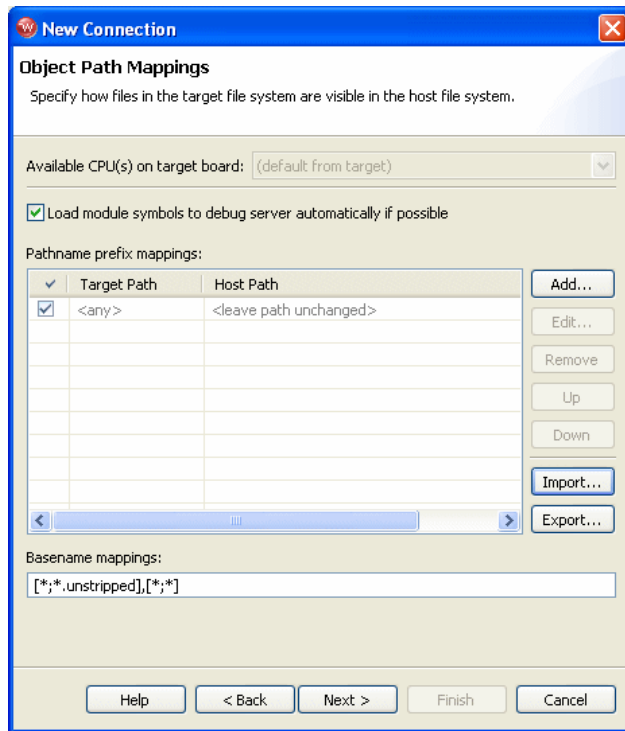
To specify an area of memory, click **Add**.
The **Set Memory Map** dialog appears.



Use the **Set Memory Map** dialog to specify which memory areas are read-only, read-write, or write-only, and to specify the access width Workbench should use to read the data from those regions.

11. Click Next.

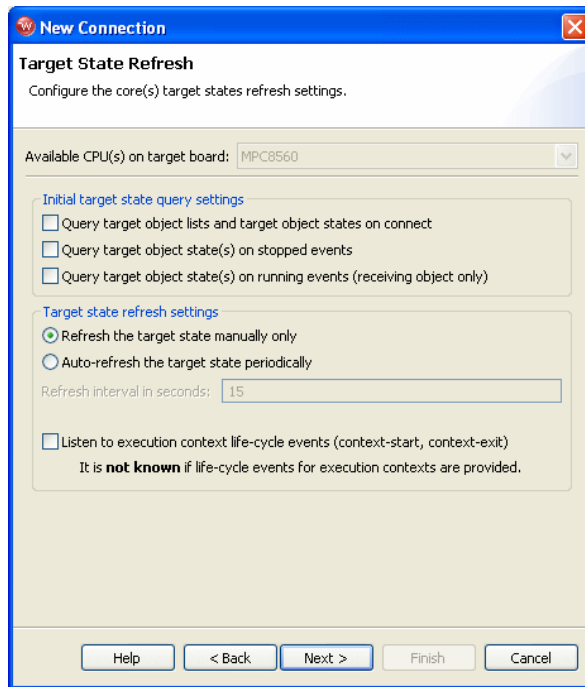
The **Object Path Mappings** dialog appears.



Use the **Object Path Mappings** dialog to specify how files in the target file system are visible in the host file system.

12. To add a host or target path, click **Add...** and type the path in the dialog that appears.
13. Click **Next**.

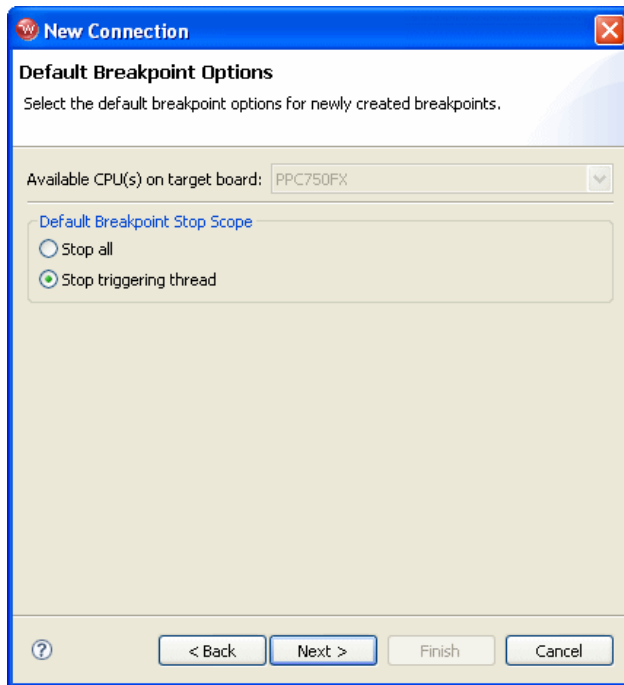
The **Target State Refresh** dialog appears.



Use the **Target State Refresh** dialog to configure the target state query and target state refresh settings on your target processor.

14. Click **Next**.

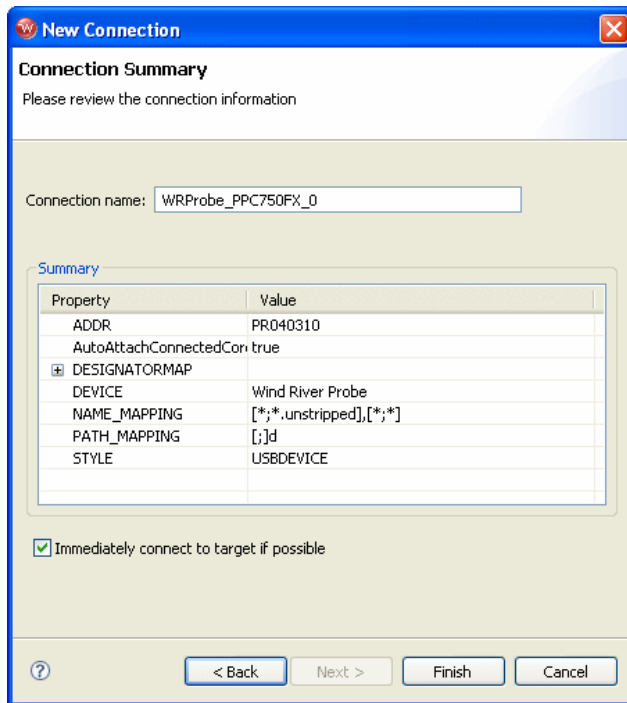
The **Default Breakpoint Options** dialog appears.



Use this dialog to set default breakpoint options for newly created breakpoints.

15. Click **Next**.

The **Connection Summary** dialog appears. Inspect the displayed values to make sure they are correct.



16. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

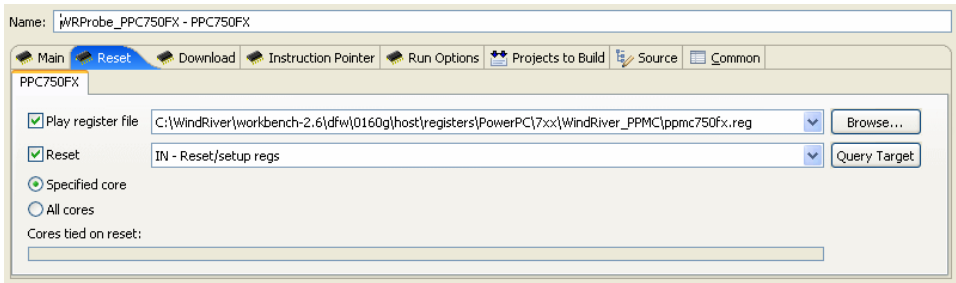
Workbench creates a target connection called **WRProbe_PPC750FX** in the **Target Manager** view.

The **Reset and Download** view appears.

3.3 Downloading Code

Use the steps in this section to download symbols and code to your target.

1. In the **Reset and Download** view, select the **Reset** tab.



2. If you want to configure the target register values with a register file, select **Play Register File** and browse for the file you want to use.

Register files for many Wind River-supported targets are located in *installDir\workbench-2.x\dfw\build\host\registers*.

If you do not want to reconfigure your target registers, leave this box unchecked.

3. Choose the type of reset initialization you want to perform.

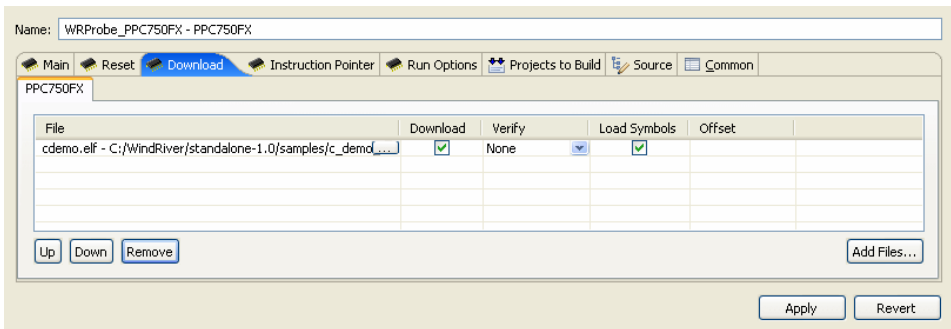
You can use the IN or INN initialization commands. For a full discussion of these two commands, see the *Wind River Workbench for On-Chip Debugging Command Reference*.

You can also choose not to perform an initialization by clearing the **Reset** box.



CAUTION: If you are manually changing registers on your target, be aware that issuing an IN or INN initialization command will overwrite your changes.

4. Select the **Download** tab.



5. Click **Add Files**.

In the browser window that appears, navigate to the executable file you want to run.

The file you select appears in the **Filename** field. Repeat this process as many times as necessary.

The file at the top of the list will download to the target first, followed by the others from the top down. You can edit the order of the list by clicking on any filename to highlight it and using the **Up**, **Down**, and **Delete** buttons.

6. Use the other fields to configure the download.

Download

The **Download** field is checked by default. If you clear it, the file will remain on the list but will not download data to the target. This is useful if, for example, you only want to download symbol information and not data.

Verify

The **Verify** field configures the extent to which the file you are downloading will be compared to a file that may already be on the target. There are three options: **Full**, **Compare**, and **None**.

When this field is set to **Full**, a write/read verify will occur for every download. Workbench writes to the target and then verifies that the write to the target and the read from the target are identical. This is slower than a normal download, but it is a useful security option.

When the field is set to **Compare**, Workbench will verify that the image has been downloaded correctly (that is, that the image on the host is the same as the image on the target.) This is useful for programming flash.



NOTE: You should only set the **Verify** field to **Compare** if an image already exists on the target. If you set the field to **Compare** when there is no image on the target, Workbench will look for a file to compare and not find one, and the reset and download operation will fail.

When the field is set to **None**, Workbench will perform no verification.

The **Verify** field is set to **None** by default.

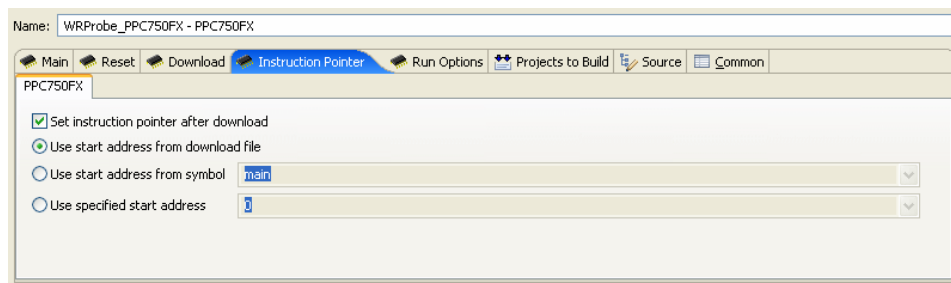
Load Symbols

The **Load Symbols** field, which is checked by default, determines whether the file's symbol information is downloaded to the target.

Offset

In the **Offset** field, you can enter a value in hex to set a memory offset bias for your application file. If you do not enter a value, Workbench uses the default value **0x00000000**.

7. Select the **Instruction Pointer** tab.



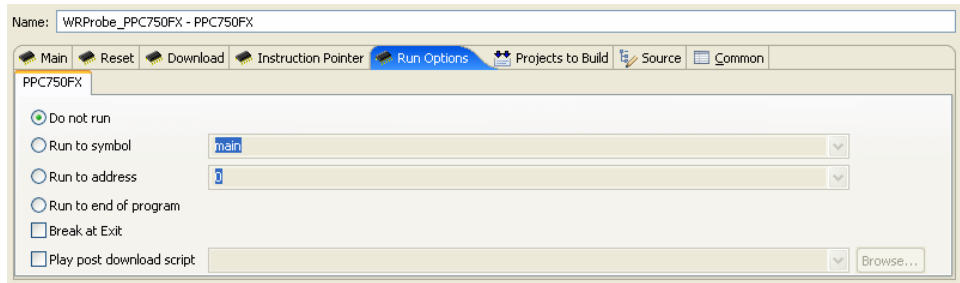
8. Set the starting point for your file.

By default, the instruction pointer is set to use the starting address from the download file.

You can set the instruction pointer to start the file from the first occurrence of a particular symbol (for example, **main**) or you can just specify a starting address by entering the address value in hex in the **Use Specified Start Address** field.

If you do not want to set a starting point, clear the **Set Instruction Pointer After Download** box.

9. Select the **Run Options** tab.



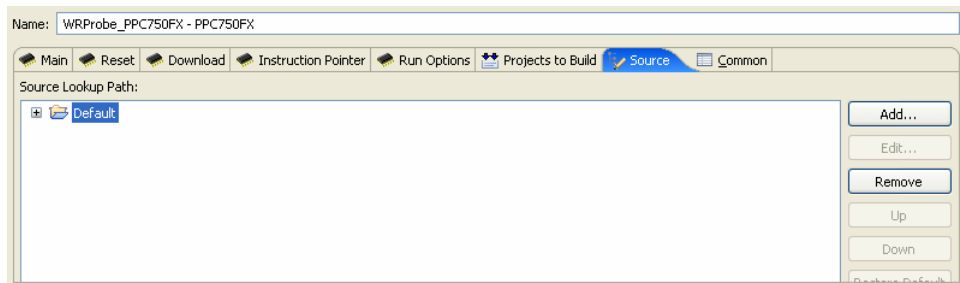
10. Determine how you want your file to run.

By default, the **Reset and Download** view is set not to run the file after downloading. If you want the file to run, you have several options to determine where it should break:

- You can set it to break at the first occurrence of a symbol (for example, **main**) by selecting **Run to Symbol** and entering the symbol in that field.
- You can set it to break at the end of your program by selecting **Run to end of program**.
- You can set it to break at a given memory address by selecting the **Run to Address** box and entering the address in hex in that field.
- You can set it to break at an **_exit** routine by selecting the **Break at Exit** box.

If you need to perform a post-initialization, you can define it here. Select the **Play post download script** box and click **Browse**. In the browser window that appears, navigate to your initialization file.

11. Select the **Source** tab.

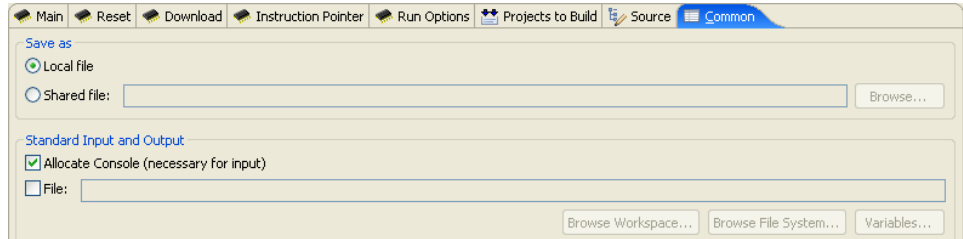


12. Use the **Source** tab to configure the source path of your file.

Workbench uses the input path of the local file system by default. Unless you need to use a different path, you do not need to do anything in the **Source** tab.

If you need to use a different path, click **Add...** and use the **Add Source** dialog to configure the appropriate search path for your project.

13. Select the **Common** tab.



14. Specify whether your launch configuration is local or shared.

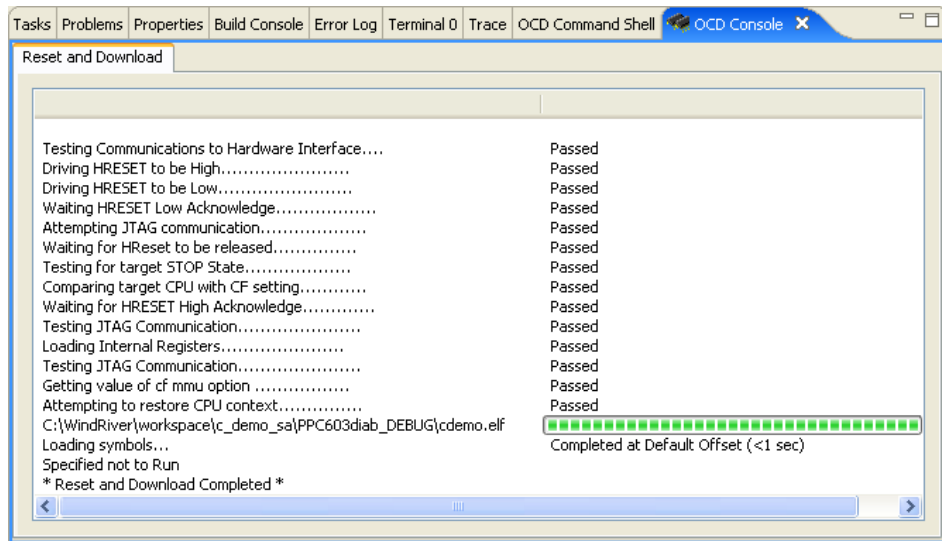
The configuration is local by default. To make it shared, click **Shared file:** and browse to the shared directory where you want the configuration to be located.

You have now fully defined your reset and download operation.

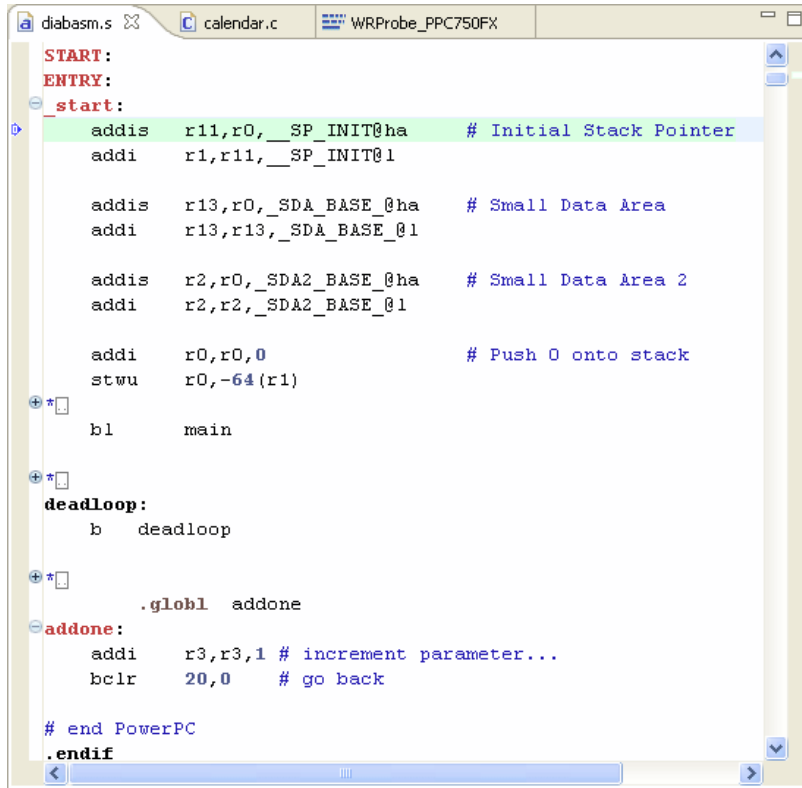
15. Click **Debug**.

Workbench initializes the target board, then downloads the file, then runs the file.

The **OCD Console** view opens to show the progress of the reset and download operation.



The Editor opens showing the Program Counter set at the beginning of the application code.



```
diabasm.s | calendar.c | WRProbe_PPC750FX
START:
ENTRY:
start:
    addis r11,r0,__SP_INIT@ha # Initial Stack Pointer
    addi  r11,r11,__SP_INIT@l

    addis r13,r0,_SDA_BASE_@ha # Small Data Area
    addi  r13,r13,_SDA_BASE_@l

    addis r2,r0,_SDA2_BASE_@ha # Small Data Area 2
    addi  r2,r2,_SDA2_BASE_@l

    addi  r0,r0,0 # Push 0 onto stack
    stwu  r0,-64(r1)

    bl    main

deadloop:
    b    deadloop

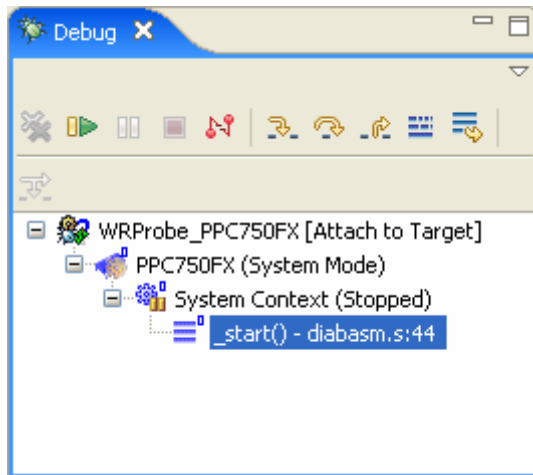
.globl addone
addone:
    addi  r3,r3,1 # increment parameter...
    bclr  20,0 # go back

# end PowerPC
#endif
```

You are now ready to run and debug the application.

3.4 Debugging Code

Use the **Debug** view to monitor, control, and manipulate the processes and tasks that you are actively debugging. The **Debug** view shows only the processes that are currently under debugger control.



3.4.1 Monitoring Processes

When you start processes under debugger control, or attach the debugger to running processes, they appear in the **Debug** view labeled with unique colors and numbers. You can change the color assigned to a process or thread by right-clicking the process or thread and selecting **Color** > *specific color*.

3.4.2 Stepping Through Code

The Editor shows the source file **diabasm.s**, showing the C Demonstration Project initialization assembly.

In the **Debug** view, click the **Step Into** button.

The Program Counter moves to the second assembly instruction. If you open the **Memory** view or the **Registers** view, you can see them update memory and register values as you step through instructions.

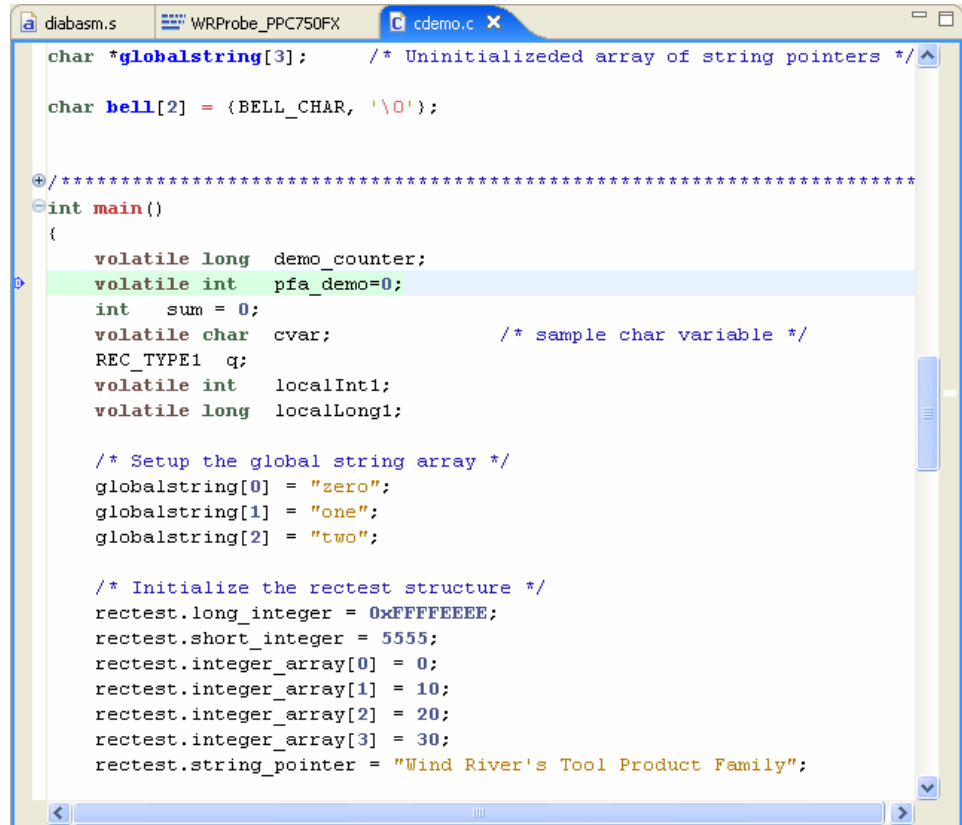
Click the **Step Into** button seven more times, to step through all the initialization code and reach the first branch instruction:

```
b1 main
```

This is where the application branches out of assembly into C code.

Click the **Step Into** button again.

The application branches into `main()` and the Editor opens the source file `cdemo.c`.



```
char *globalstring[3]; /* Uninitialized array of string pointers */

char bell[2] = {BELL_CHAR, '\0'};

/*****
int main()
{
    volatile long demo_counter;
    volatile int pfa_demo=0;
    int sum = 0;
    volatile char cvar; /* sample char variable */
    REC_TYPE1 q;
    volatile int localInt1;
    volatile long localLong1;

    /* Setup the global string array */
    globalstring[0] = "zero";
    globalstring[1] = "one";
    globalstring[2] = "two";

    /* Initialize the rectest structure */
    rectest.long_integer = 0xFFFFFFFF;
    rectest.short_integer = 5555;
    rectest.integer_array[0] = 0;
    rectest.integer_array[1] = 10;
    rectest.integer_array[2] = 20;
    rectest.integer_array[3] = 30;
    rectest.string_pointer = "Wind River's Tool Product Family";
}
```

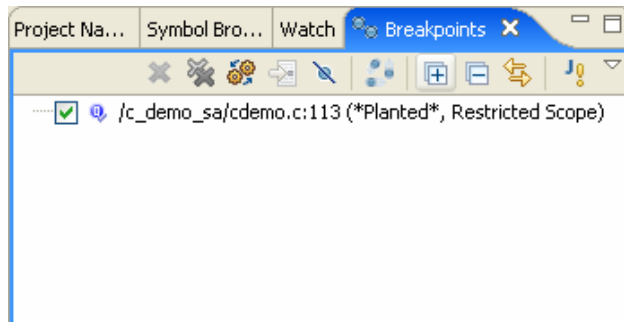
3.4.3 Setting a Software Breakpoint

Breakpoints allow you to stop a running program at particular places in the code or when specific conditions exist.

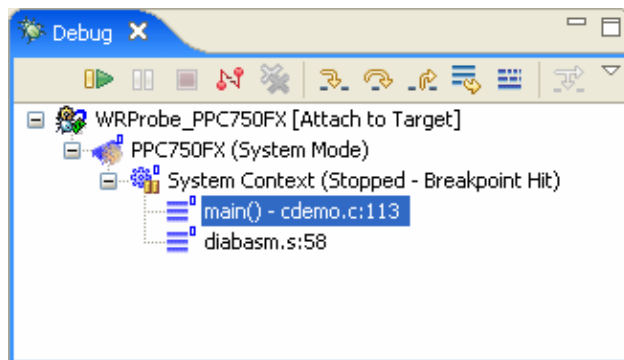
In the left ruler of the Editor (the gutter), double-click to the left of the source line

```
globalstring[2] = "two";
```

This sets a software breakpoint on that source line. The breakpoint appears in the **Breakpoints** view.



In the **Debug** view, click the **Resume** button. The program runs until it hits the breakpoint. The **System Context** changes to **Stopped -- Breakpoint Hit**.



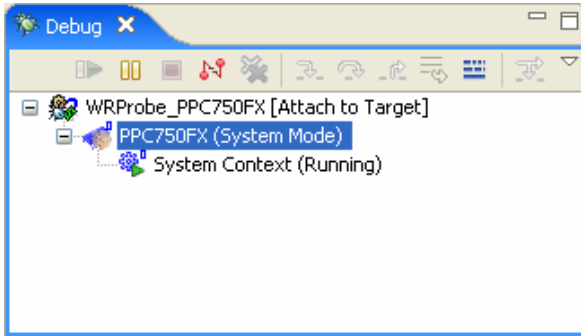
Breakpoint information also appears in the **OCD Command Shell**:

```
>RUN>  
  
!BREAK! - [msg12000] Software breakpoint; PC = 0x00014074 [EVENT Taken]  
>BKM>
```

3.4.4 Running a Program

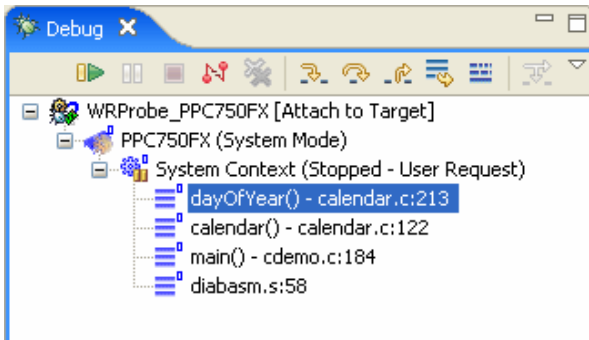
To run your downloaded program, click **Resume** in the **Debug** view. The program will run until it hits a breakpoint. If there are no breakpoints or interrupts, the program will run to completion or until you click **Suspend**.

When the program is running, the System Context changes to **Running**, and a **>RUN>** prompt appears in the **OCD Command Shell**.



If there are no breakpoints, you can stop the program by clicking the **Suspend** button in the **Debug** view or by entering the **HA** command at the **>RUN>** prompt in the **OCD Command Shell**.

The Editor updates to show the current location of the Program Counter and the **System Context** in the **Debug** view changes to **Stopped -- User Request**.



3.4.5 Stepping Through a Program

To single-step without going into other subroutines, click **Step Over** instead of **Step Into**.

While stepping through a program, you may conclude that the problem you are interested in lies in the current subroutine's caller, rather than at the stack level

where your process is suspended. In this situation, if you click **Step Return**, execution continues until the current subroutine completes, then the debugger regains control in the calling statement.

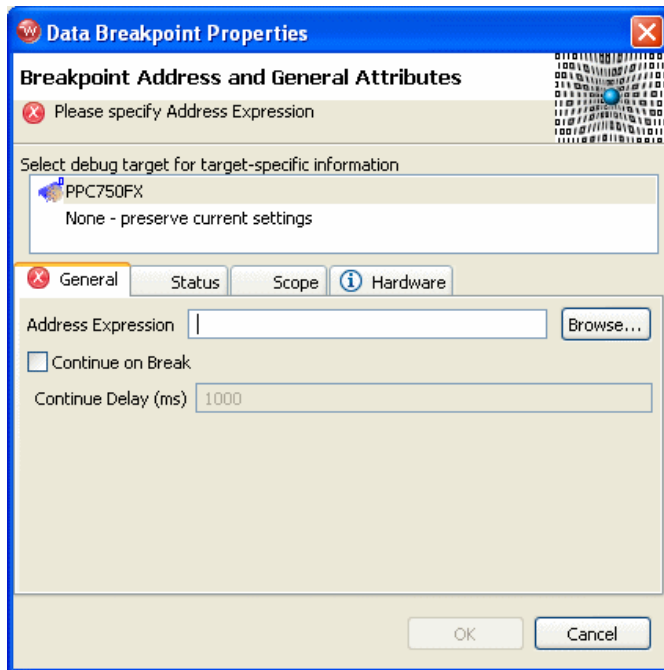
3.4.6 Setting a Hardware Breakpoint

The availability of hardware breakpoints varies by architecture. You can only set as many hardware breakpoints as there are debug registers available on your target.

Once a hardware breakpoint is trapped, the debugger will behave in the same way as for a standard breakpoint and stop for user interaction.

In the **Breakpoints** view, click on the **Menu** button and select **Add Data Breakpoint**.

The **Data Breakpoint** dialog appears.



If an error message appears, you may have exceeded the number of allowed hardware breakpoints (four for most targets). Right-click in the **Breakpoints** view and select **Remove All**. Then select **Menu > Add Data Breakpoint** again.

If an error message still appears, your target may not support hardware breakpoints.

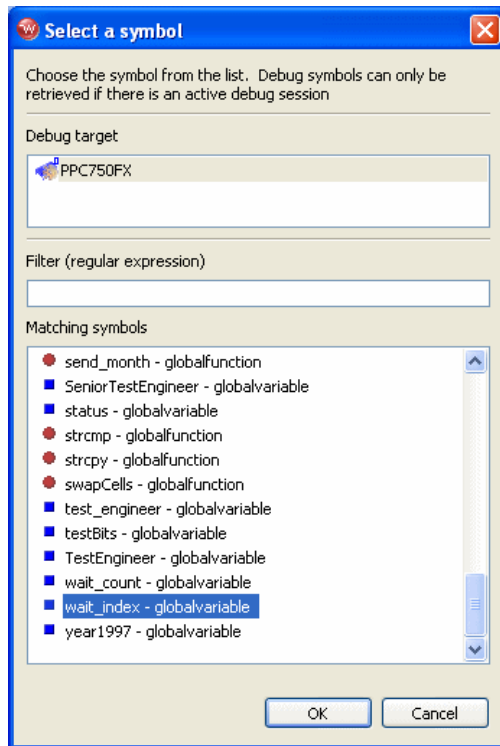
You can use data hardware breakpoints to find out which routines are modifying a specific variable.

The **Address Expression** can be a symbol or a specific address in hex. You can use the address **0x0** in the **Address Expression** field to set a data hardware breakpoint to catch null pointers. You can set the **Address Expression** field to an address in the stack area to set a data hardware breakpoint to find out if the stack grew to that point.

The following example sets a symbol in the **Address Expression** field.

1. Click **Browse**.

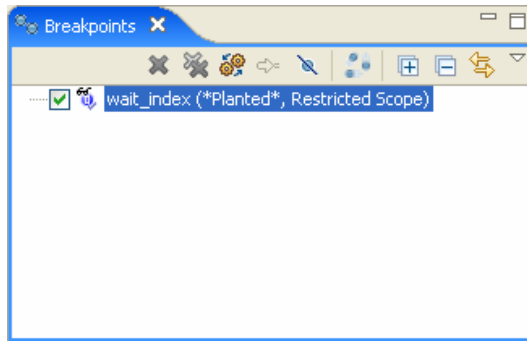
The **Select Symbol** dialog appears, showing a list of available symbols that can take a hardware breakpoint.



2. Scroll down and highlight the symbol **wait_index**.
3. Click **OK**.

The global variable **wait_index** is now the address for the data hardware breakpoint.

The hardware breakpoint on **wait_index** appears in the **Breakpoints** view.



In the **Debug** view, click **Resume**.

The program runs until it hits the hardware breakpoint. Workbench halts the processor when it locates `wait_index` and displays that source line in the Editor.

3.4.7 Disconnecting and Terminating Processes

Disconnecting from a process or core detaches the debugger, but leaves the process or core in its current state.

Terminating a process actually kills the process on the target.



NOTE: If the selected target supports terminating individual threads, you can select a thread and terminate only that thread.

3.5 Moving On

For descriptions of other features of Wind River Workbench for On-Chip Debugging, such as code profiling, code tracing, and so on, see the relevant chapters in this document.

4

Using the OCD Standalone Project Wizard

- 4.1 Introduction 63
- 4.2 Creating an OCD Standalone Project 64
- 4.3 Building an OCD Standalone Project 71
- 4.4 Setting Standalone Project Defaults 72

4.1 Introduction

Every Workbench project has a set of build specs from which you can select, depending on your target processor. For example, there may be a build spec for a PowerPC 603 target and a different build spec for an ARM 920T target, as well as many other variations.

For operating system-independent (standalone) applications, rather than simply providing all possible variations, Workbench provides a Standalone Project Wizard, from which you can create a build spec dynamically.

Standalone application projects can only be debugged using OCD functionality.

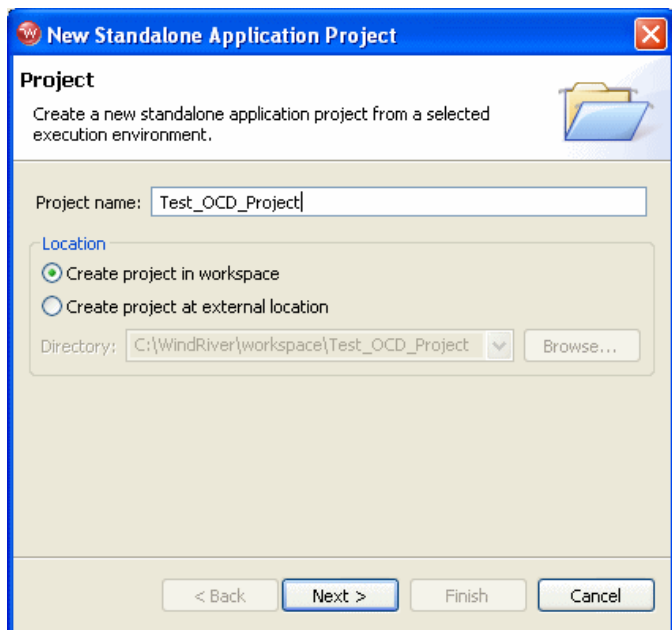
4.2 Creating an OCD Standalone Project

To create a standalone project:

1. In Workbench, select **File > New > Standalone Application Project**.

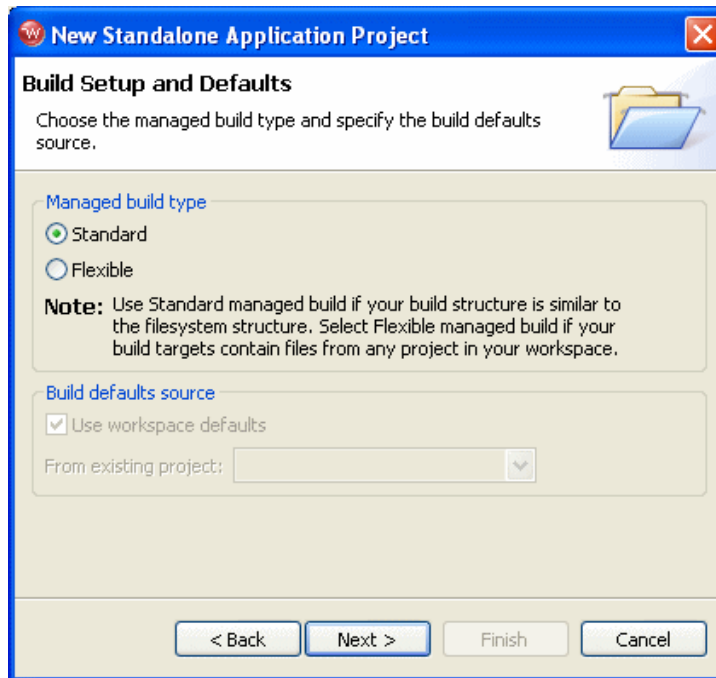
The **Standalone Project** wizard appears, as shown in [Figure 4-1](#).

Figure 4-1 **Standalone Wizard -- Project Name**



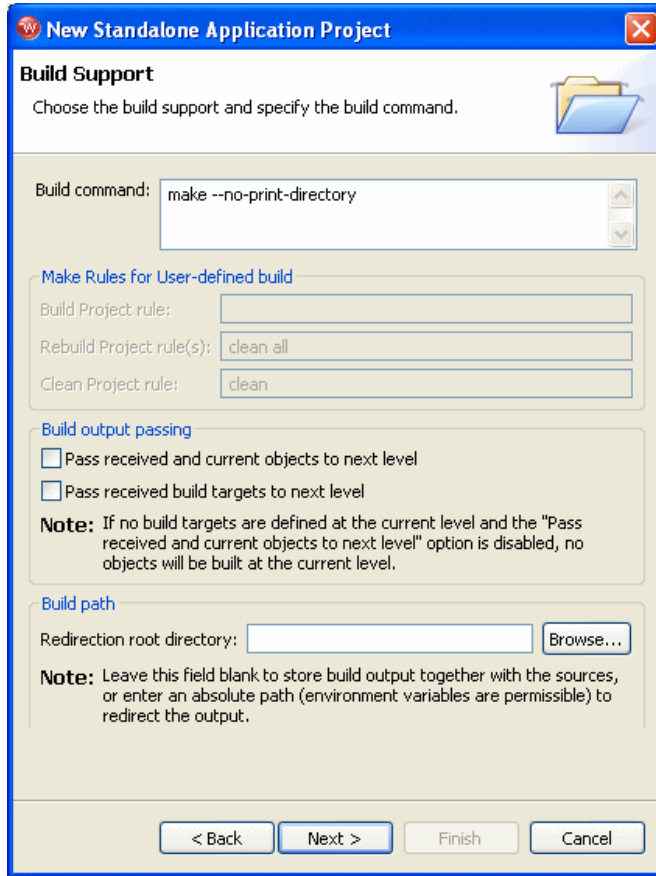
2. In the **Project Name** field, assign a name to your project. In the example shown in [Figure 4-1](#), the project name is **Test_OCD_Project**.
3. Click **Next**.

Figure 4-2 Standalone Wizard -- Build Defaults



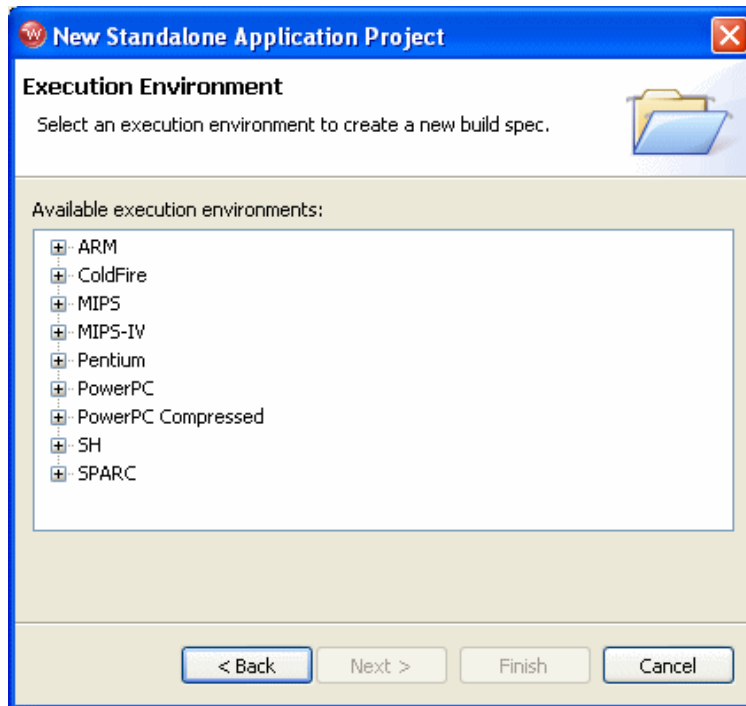
4. Specify the build defaults for your project.
If your build structure is similar to the filesystem structure, select **Standard**.
If your build targets contain files from any project in your workspace, select **Flexible**.
5. Click **Next**.

Figure 4-3 Standalone Wizard -- Build Support



6. Specify the build support for your project.
You can also use this step to determine how build output is passed.
7. Click **Next**.
The **Build Spec** wizard appears, as shown in [Figure 4-4](#).

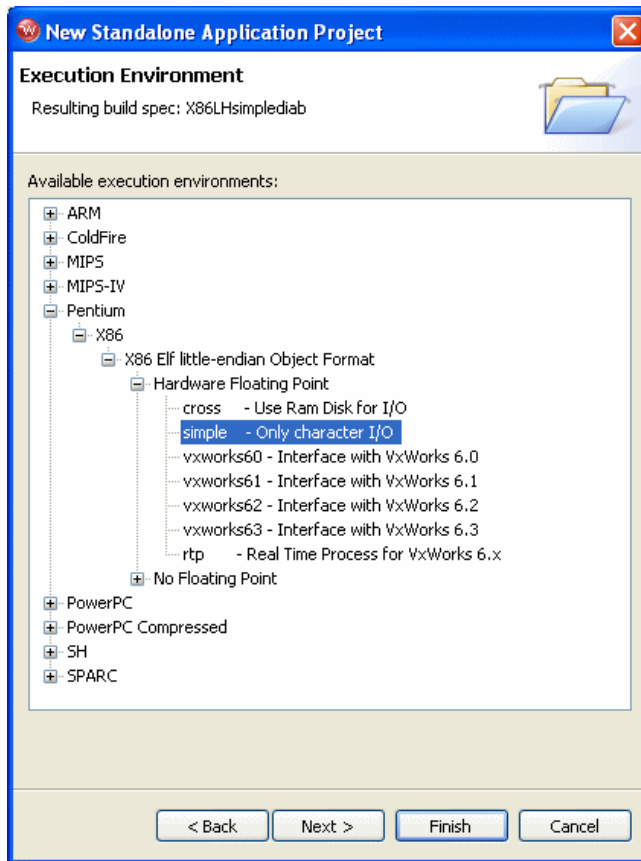
Figure 4-4 Build Spec Wizard



Use the **Build Spec** wizard to create a set of build tool commands for a specific target environment. You will need to create a build spec for each target you want to build for.

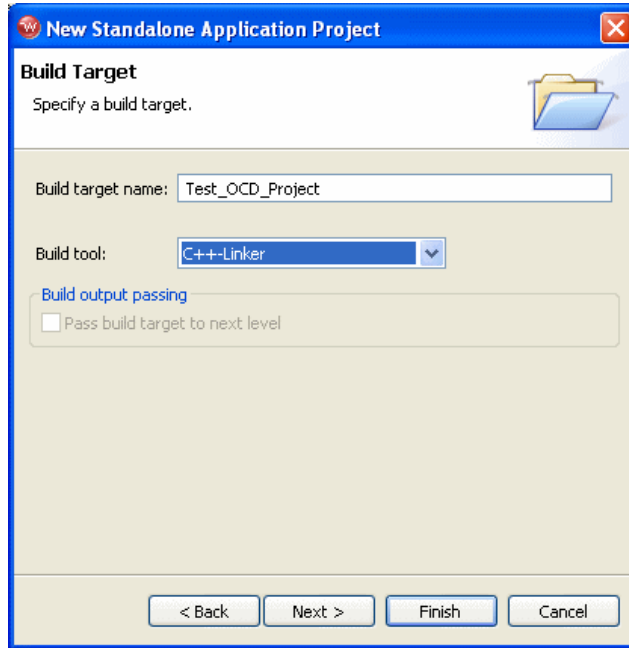
To complete the wizard you must fully expand at least one node of the tree and select an innermost node, as shown in [Figure 4-5](#).

Figure 4-5 Build Spec Wizard -- Innermost Node



8. Click **Next**.

Figure 4-6 Standalone Wizard -- Build Target



Specify the build tool for your project.

The **Build Tool:** field has five options: **Linker**, **C-Linker**, **C++Linker**, **Librarian**, or **(User-Defined)**. You can switch the build tool to build static libraries with this project type, or to use the C or C++ compiler for linking.

If the project is created as a root project, the default is **Linker**.

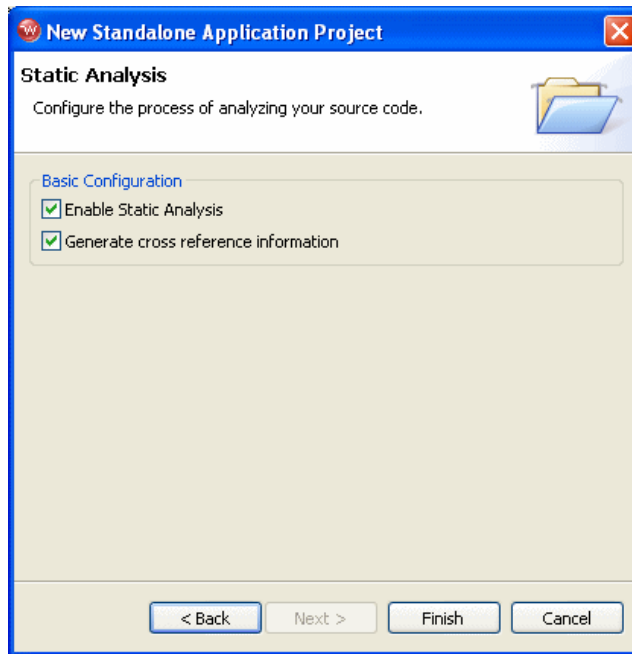
If the project is created as a subproject, the default is **Librarian**.



NOTE: If you selected an existing project before starting the wizard, you may select a superproject for the new project.

9. Click **Next**.

Figure 4-7 **Standalone Wizard -- Static Analysis**



Configure the process of analyzing your source code.

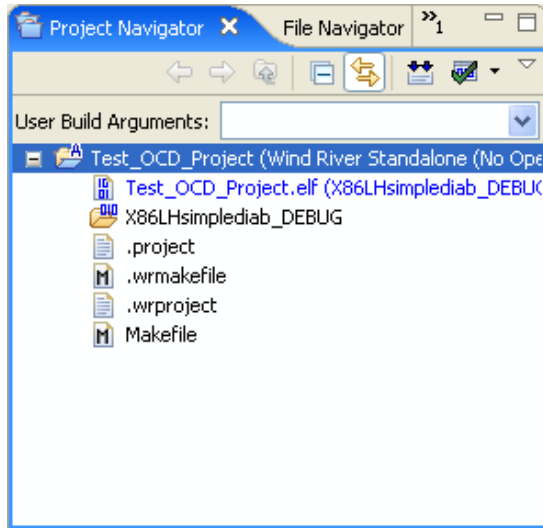
By default, the wizard is set to enable static analysis and to generate cross-reference information. To disable either of these options, clear the checkbox in the wizard.

10. Click **Finish**.

Your project appears in the **Project Navigator** view, as shown in [Figure 4-8](#).

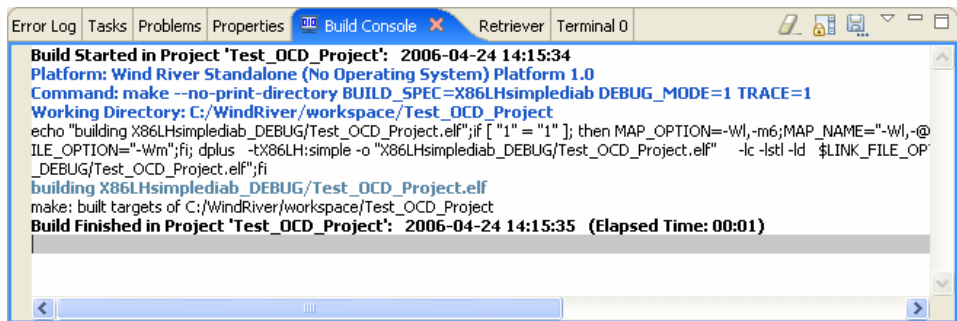
4.3 Building an OCD Standalone Project

Figure 4-8 Test Project



To build your project, right-click on the test name in the **Project Navigator** view and select **Build Project**. Build output is displayed in the **Build Console** view, as shown in [Figure 4-9](#).

Figure 4-9 Build Console View



You can now run and debug your standalone project.

4.4 Setting Standalone Project Defaults

You can set workspace build defaults in Workbench and specify default build specs for all new standalone projects.

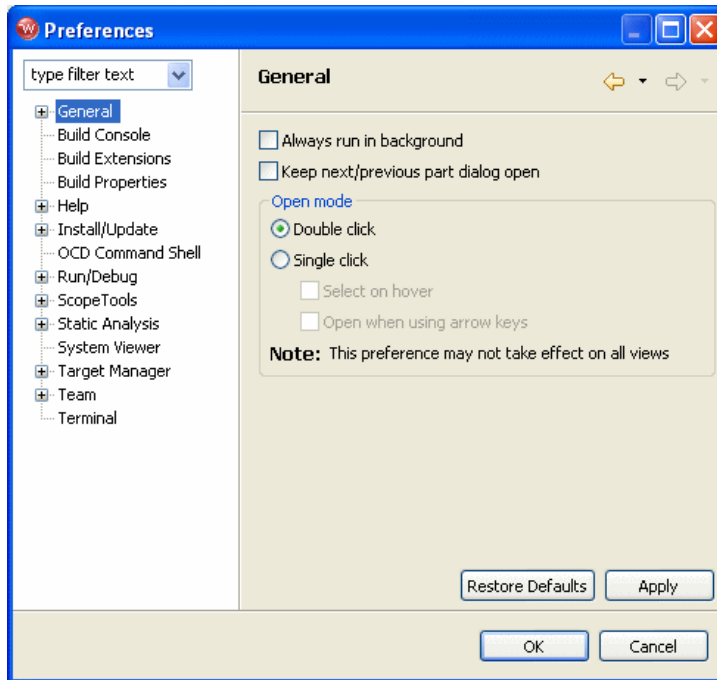
→ **NOTE:** Setting new standalone defaults will not affect already-existing standalone projects.

To set standalone project defaults:

1. In Workbench, select **Window > Preferences**.

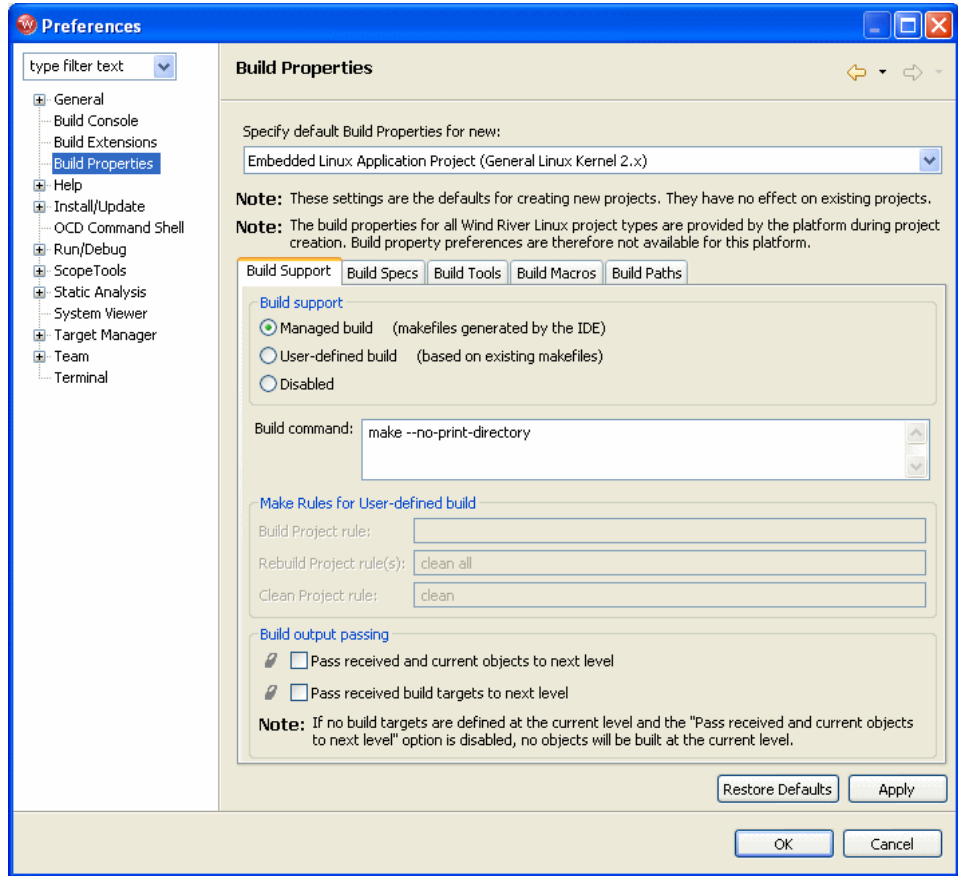
The **Preferences** dialog appears, as shown in [Figure 4-10](#).

Figure 4-10 **Preferences Dialog**



2. Select **Build Properties**.

Figure 4-11 Build Properties



3. In the **Specify Default Build Properties for New:** field, select **Standalone Application Project (Wind River Standalone (No Operating System) Platform 1.0)**.
4. Select the **Build Support** tab.
5. Use the settings in the **Build Support** tab to configure the build support for standalone projects and click **Apply**.
6. Repeat this procedure for the **Build Specs**, **Build Tools**, **Build Macros**, and **Build Paths** tabs.
7. Click **OK**.

Any new standalone projects you create will now use the settings you specified in the **Preferences** dialog as defaults.

5

Defining a Launch Configuration

- 5.1 Introduction 75
- 5.2 Creating a Launch Configuration 76
- 5.3 Other Options 85

5.1 Introduction

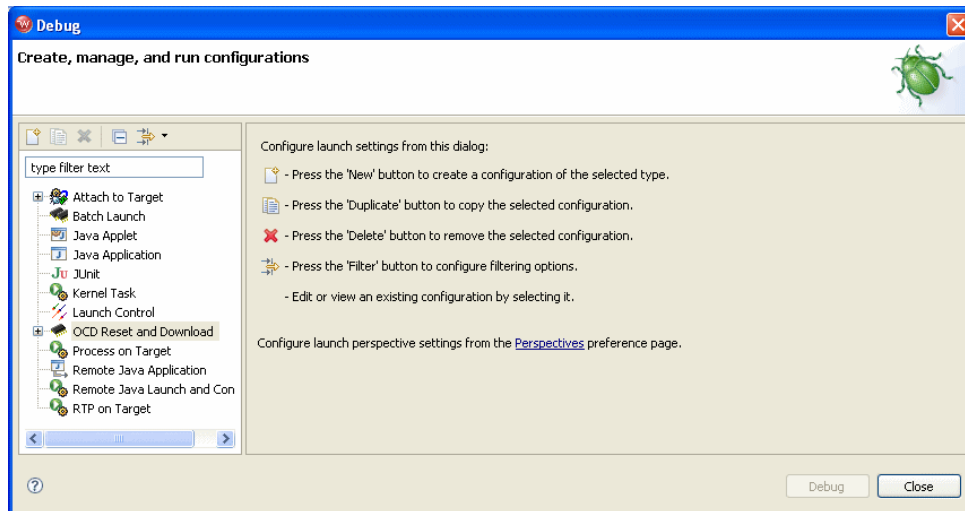
Use the **Launch Configuration** dialog to edit your defined emulator-target connections and their associated actions, such as initializing your target board and downloading a file to run on your target.

The **Launch Configuration** dialog is very similar to the **Reset and Download** view, which is described in the *Establishing Communications* chapter of your emulator's Hardware Reference. The difference is that the Launch Configuration, once defined, is persistent, and you can launch it at any time with one click without having to re-enter your values.

The values you enter in the **Reset and Download** view are not persistent to the Launch Configuration you defined for your emulator and target, so using the **Reset and Download** view will not affect your Launch Configuration.

In the Workbench toolbar, select **Run > Debug**.

The **Launch Configuration** dialog appears.



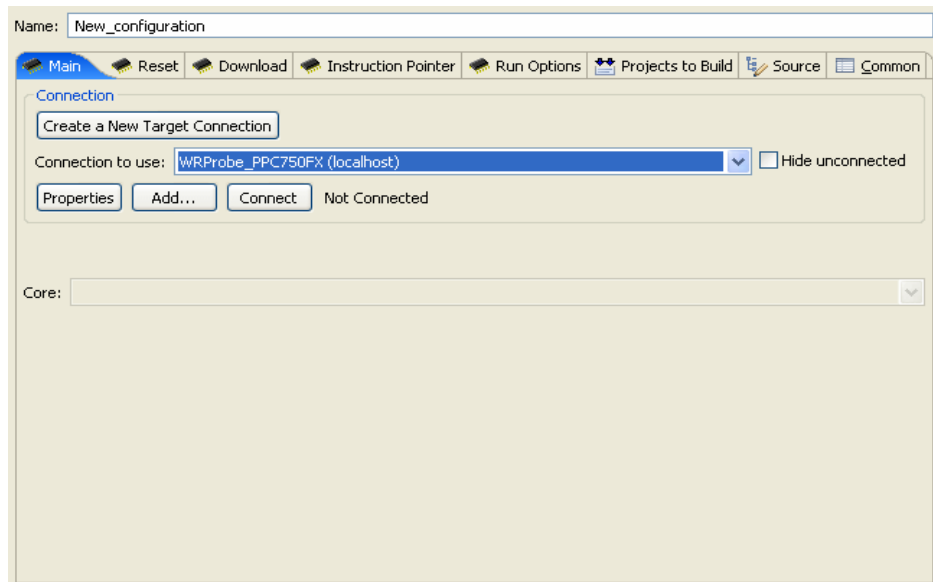
You can use the **Launch Configuration** dialog to create a new launch configuration or to edit, delete, or duplicate an existing launch configuration.

5.2 Creating a Launch Configuration

To create a new OCD launch configuration, use the following steps.

1. Highlight the configuration type **OCD Reset and Download** and click the **New** button.

The **Main** tab appears.



2. Assign a name to the launch configuration.

By default, the **Name** field will populate with the name of the most recently used target connection. If you want to use a different name, select the **Name** field and enter a name.

The connection registry is set to **localhost** by default.

3. Connect to an emulator and target.

To connect using the default target connection, click **Connect**.

If you want to create a new target connection, click **Create a New Target Connection** to open the **New Connection wizard**. Create a target connection following the procedure described in [2.2 Connecting to the Target](#), p.6.

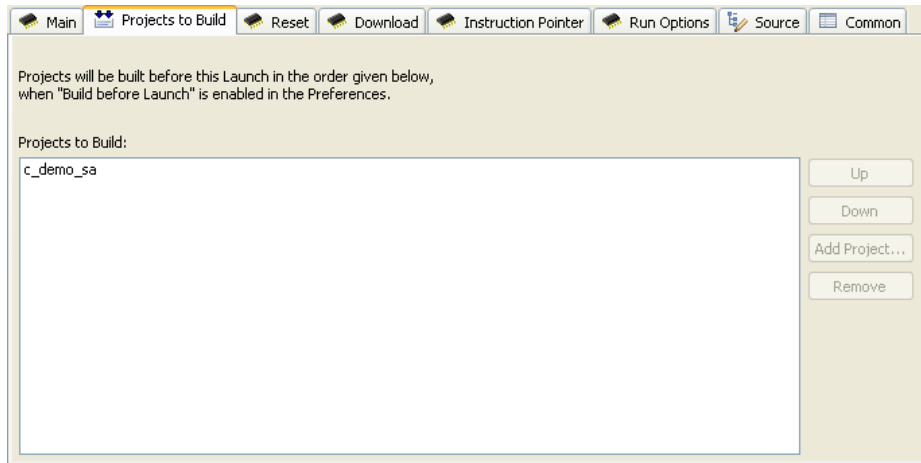
If you decide you want to change your emulator settings, you can return to the **Settings** dialog box by clicking **Properties**.

Your emulator is now connected to the host computer and your target.

5.2.1 Specifying Files

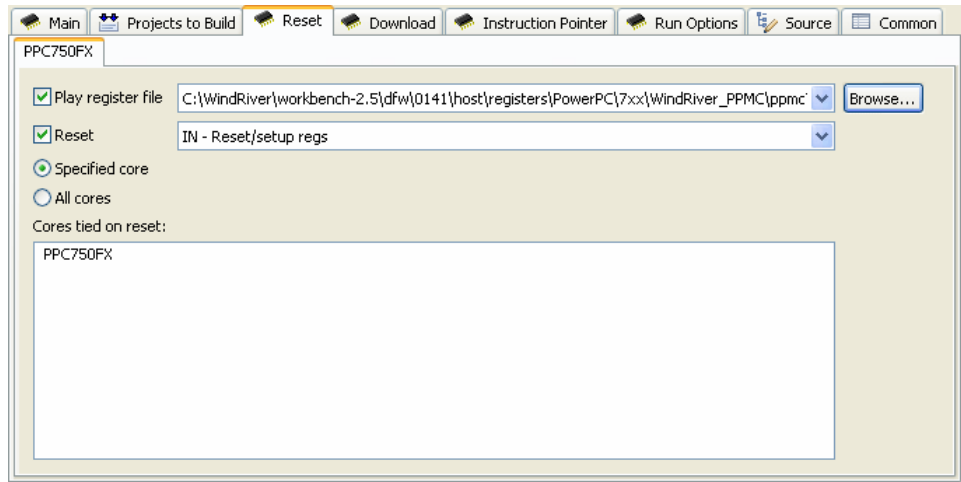
To download and run a file on your target, use the other tabs on the **Launch Configuration** dialog.

1. Select the **Projects to Build** tab.



By default, the project you specify will build before the configuration is launched. If you do not want the project to build first, select **Window > Preferences > Launching** and clear the **Build (if required) before launching** check box.

2. To specify a project to build, click **Add Project...** and select the project name from the list of available projects in the dialog that appears.
You can add more than one project. Edit your project list with the **Up**, **Down**, and **Remove** buttons.
3. Select the **Reset** tab.



4. If you want to play a register file, select **Play Register File** and browse for the register file you want to use.

This example shows a Wind River PPMC750FX target; the Wind River register file for this target is **ppmc750fx.reg**, located in *installDir/workbench-2.x/dfw/build/host/registers* in the directory **PowerPC/7xx/WindRiver_PPMC**.

If you do not want to reconfigure your target registers, leave this box unchecked.

5. Choose the type of reset initialization you want to perform.

You can use the **IN** or **INN** initialization commands. For a full discussion of these two commands, see the *Wind River Workbench for On-Chip debugging Command Reference*.

You can also choose not to perform an initialization by clearing the **Reset** box.



CAUTION: If you are manually changing registers on your target, be aware that issuing an **IN** or **INN** initialization command will overwrite your changes.

6. Select **Specified Core**.

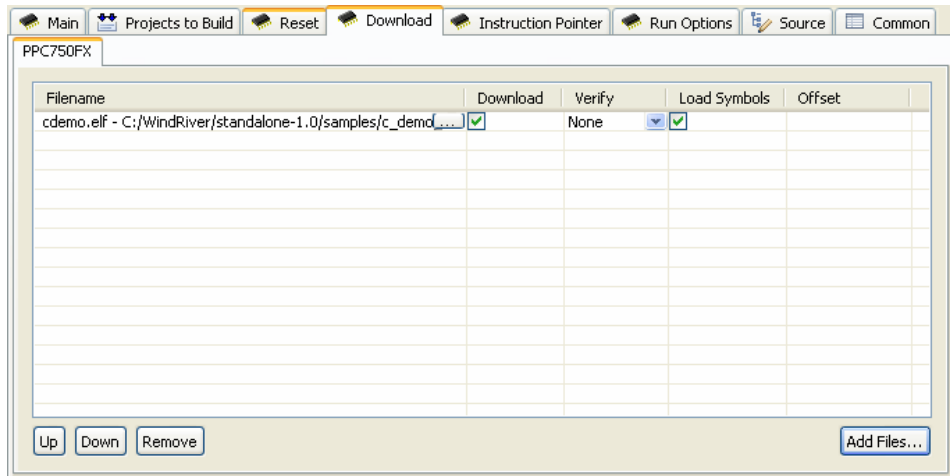
In the **Cores Tied on Reset** field, you will see a list of all the cores on your JTAG scan chain. If you want your reset and download to affect only one core, click on that core in the **Cores Tied on Reset** field and check **Specified Core**. If you

want your reset and download to affect all your target boards, click on **All Cores**.

The Wind River Probe does not support multi-core debugging, so if you are using a Wind River Probe, you do not need to set the **Cores Tied on Reset** field.

7. Select the **Download** tab.

Figure 5-1 **Download Tab**



8. Click **Add Files**.

In the browser window that appears, navigate to the executable file you want to run. This example shows the PowerPC version of the executable **cdemo.elf** file from the sample C Demonstration Project.

The file you select appears in the **Filename** field. Repeat this process as many times as necessary.

The file at the top of the list will download to the target first, followed by the others from the top down. You can edit the order of the list by clicking on any filename to highlight it and using the **Up**, **Down** and **Delete** buttons.

9. Use the other fields to configure the download.

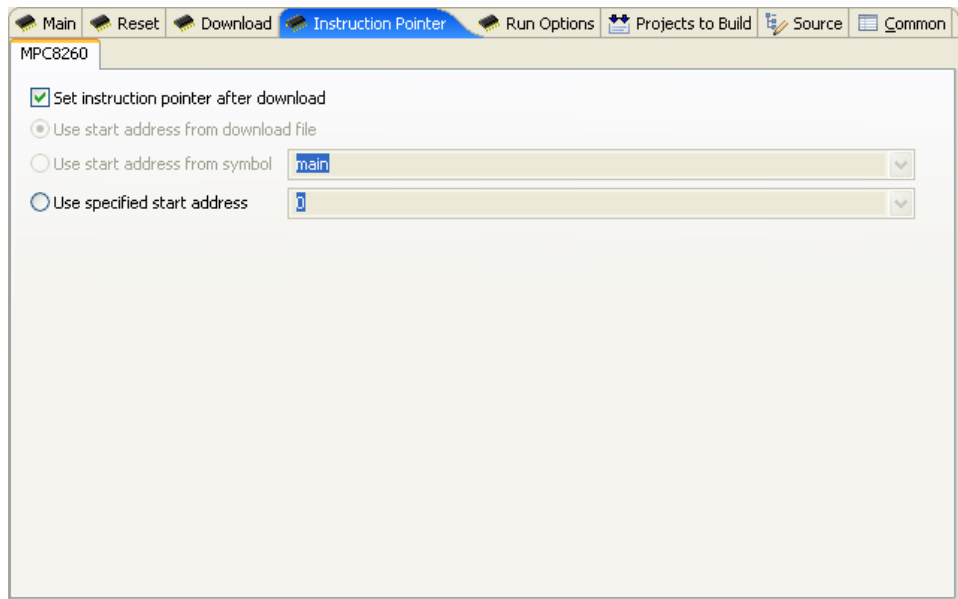
The **Download** field is selected by default. If you clear it, the file will remain on the list but will not download data to the target. This is useful if, for example, you only want to download symbol information and not data.

The **Verify** field configures the extent to which the file you are downloading will be compared to a file that may already be on the target. By default this field is set to **None**.

The **Load Symbol** field, which is selected by default, determines whether the symbol information from the file is downloaded to the target.

In the **Offset** field, you can enter a value in hex to set a memory offset bias for your application file. If you do not enter a value, Workbench will use the default value 0x00000000.

10. Select the **Instruction Pointer** tab.

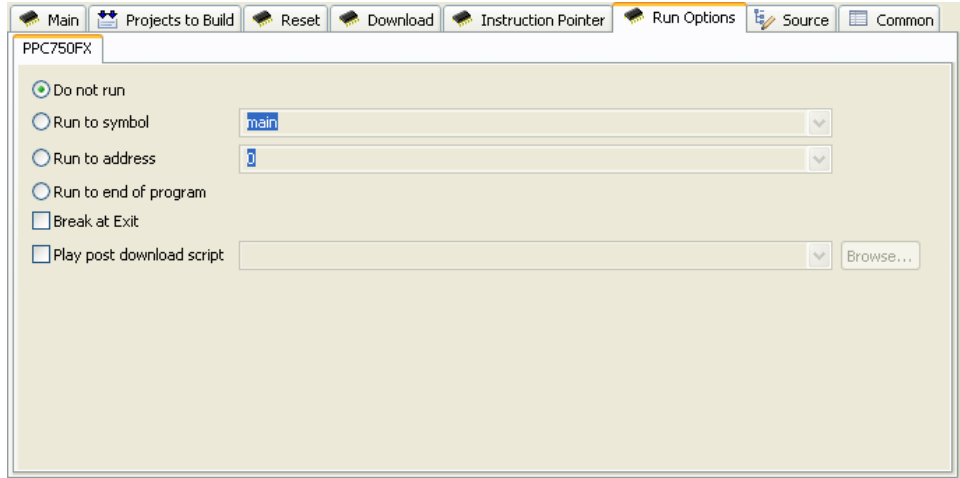


11. Set the starting point for your file.

By default, the instruction pointer is set to use the starting address from the download file. However, you can set it to start the file from the first occurrence of a particular symbol (for example, **main**) or you can just specify a starting address by typing the address value in hex in the **Use Specified Start Address** field.

If you do not want to set a starting point, clear the **Set Instruction Pointer After Download** box.

12. Select the **Run Options** tab.



13. Determine how you want your file to run.

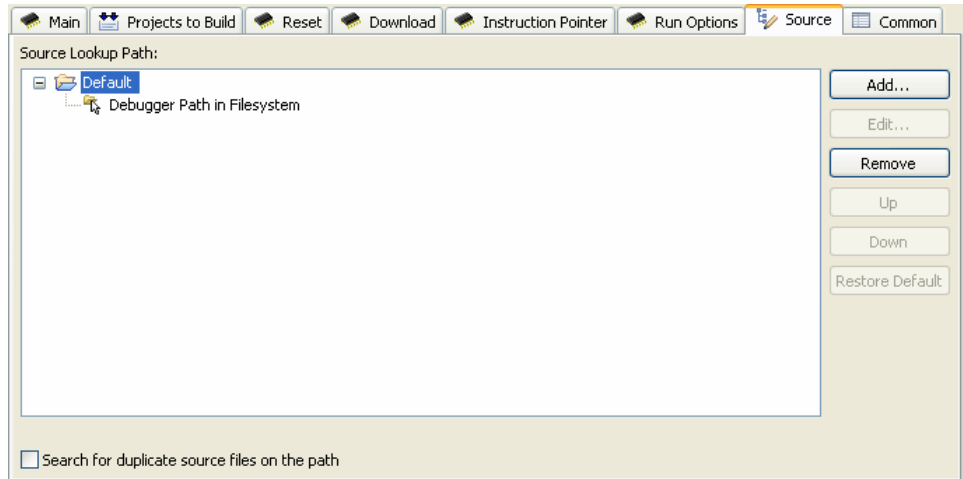
By default, the launch configuration is set not to run the file after downloading. If you want the file to run, you have several options to determine where it should break:

- You can set it to break at the first occurrence of a symbol (for example, **main**) by checking the **Run to Symbol** box and entering the symbol in that field.
- You can set it to break at a given memory address by checking the **Run to Address** box and entering the address in hex in that field.
- You can set it to break at an **_exit** routine by checking the **Break at Exit** box.

If you need to perform a post-initialization, you can define it here. Select the **Play Post Download Script** box and click **Browse**. In the browser window that appears, navigate to your initialization file.

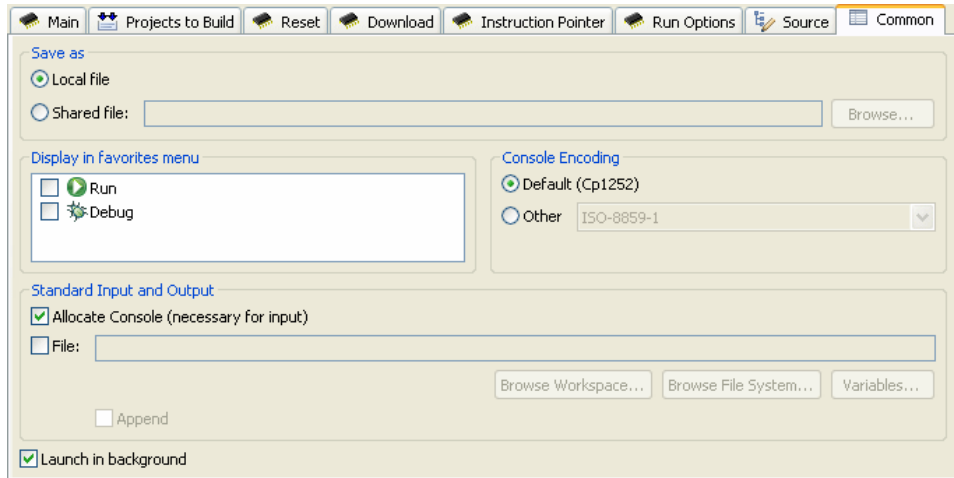
14. Select the **Source** tab.

Figure 5-2 Source Tab



15. Use the **Source** tab to configure the source path of your file.
Workbench uses the input path of the local file system by default. Unless you need to use a different path, you do not need to do anything in the **Source** tab.
If you need to use a different path, click **Add...** and use the **Add Source** dialog to configure the appropriate search path for your project.
16. Select the **Common** tab.

Figure 5-3 Common Tab



17. Specify whether your launch configuration is local or shared.

The configuration is local by default. To make it shared, click **Shared** and browse to the shared directory where you want the configuration to be located.

18. Select **Debug**.

This will make this launch configuration visible in the **Debug** menu in the Workbench toolbar, so you can return to it and launch it at any time.

You have now fully defined your launch configuration.

19. Launch or close the configuration.

To launch the reset and download operation now, click **Debug**.

Workbench will first initialize the target board, then download the file, then run the file. You can proceed to step through instructions and debug the file as explained in the *Wind River Workbench User's Guide*.

To save the launch configuration without downloading and running the file, click **Close**.

The name you gave this configuration is now visible in the **Debug** menu in the Workbench toolbar. To launch it at any time, click the menu arrow next to the Debug icon. A list of launch configurations will appear; choose the one you want to launch.



NOTE: Clicking on the **Debug** icon itself will automatically launch the most recently used launch configuration.

5.3 Other Options

To create a duplicate of an existing launch configuration, highlight the configuration name and click the **Duplicate** button.

To delete an existing launch configuration, highlight the configuration name and click the **Delete** button.

To set which launch configurations the **Launch Configuration** dialog shows, click the **Filter** button.

6

Using Board Descriptor Files

- 6.1 Introduction 87
- 6.2 Board Descriptor Files 88
- 6.3 Creating a New Board Descriptor File 89
- 6.4 XML Board Files 98
- 6.5 Manually Creating XML Board Files 101



NOTE: This chapter applies only to applications that involve multi-core debugging. For single-core debugging, you do not need to use a board descriptor file. Multi-core debugging is not supported for the Wind River Probe. This chapter applies only to the Wind River ICE SX.

6.1 Introduction

Wind River emulators use the Joint Test Action Group (JTAG) interface to communicate to the target microprocessor, and share this interface with boundary-scan board-circuit testing. The JTAG interface follows the IEEE 1149.1 boundary-scan (JTAG/Test Interface) specification.

The JTAG interface consists of a set of five signals, three JTAG registers, and a test access port (TAP) controller. The TAP controller is typically embedded in the target microprocessor or device. The information related signals are TDI (Test Data In)

and TDO (Test Data Out). The boundary-scan register chain (data) includes registers controlling the direction of the input/output drivers, as well as registers reflecting the signal value received or driven. The expectation and details of particular CPU chains are encoded directly into the emulator firmware.

Each device sharing the JTAG interface employs a serial stream of relative data. The data streams for all devices can be chained together. An associated process can scan the combined chain to extract any particular device's information.

For additional information about JTAG operations, refer to the IEEE 1149.1 specification at <http://standards.ieee.org>.

6.2 Board Descriptor Files

In most cases you do not need to concern yourself with the JTAG board file. However, when performing multi-core debugging, or when debugging a target that has other devices besides the processor on the scan chain, your Wind River ICE SX requires a board descriptor file to correctly set up the JTAG scan chain for your target.

The board file provides a description of each of the devices that are included in the scan chain, and provides information about each device.

All Wind River target boards are shipped with a board descriptor file that works for that target board. If you are using a Wind River target board, you can specify the default board descriptor file for that target in the **New Connection Wizard** in Wind River Workbench, as described in the *Wind River ICE SX Hardware Reference: Establishing Communications*.



NOTE: If you choose to modify a board descriptor file that was shipped with Wind River Workbench, save your modified file with a different name to prevent overwriting the default file.

Board descriptor files are written in extensible Markup Language (XML). However, it is easiest to create or modify board files using Workbench. The software allows you to create and catalog scan chain devices such as processors, complex programmable logic devices (CPLDs), field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs), and from that catalog create a board file that properly describes the scan chain on your target.



CAUTION: Your board file must list the devices included on your scan chain in the same order as they are physically laid out on the target. If the board file and the physical scan chain do not match, the board file for your target will not work.

6.3 Creating a New Board Descriptor File

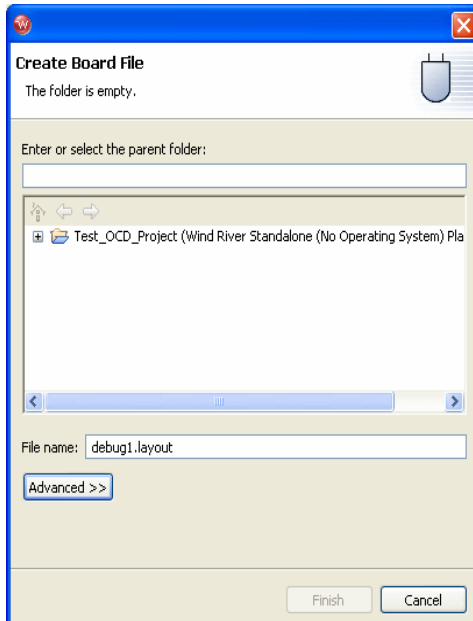
Workbench uses JTAG Editor to create and modify board files. To use the JTAG Editor view, you must first have an active project running. For information on creating projects, see the *Wind River Workbench User's Guide*.

To create a new board file:

1. Open your project in Workbench.
2. Select **File > New > JTAG Board Layout**.

The **Create Board File** dialog appears, as shown in [Figure 6-1](#).

Figure 6-1 **Create Board File Dialog**



Wind River Workbench automatically populates the **Parent Folder** field with your active project. In the **File Name** field, type a name for your board file. This creates a **.layout** file, which JTAG Editor will use to create a **.brd** file in the next step.

The example shown in [Figure 6-1](#) creates a file called **debug1.layout** for the project **debug1**.

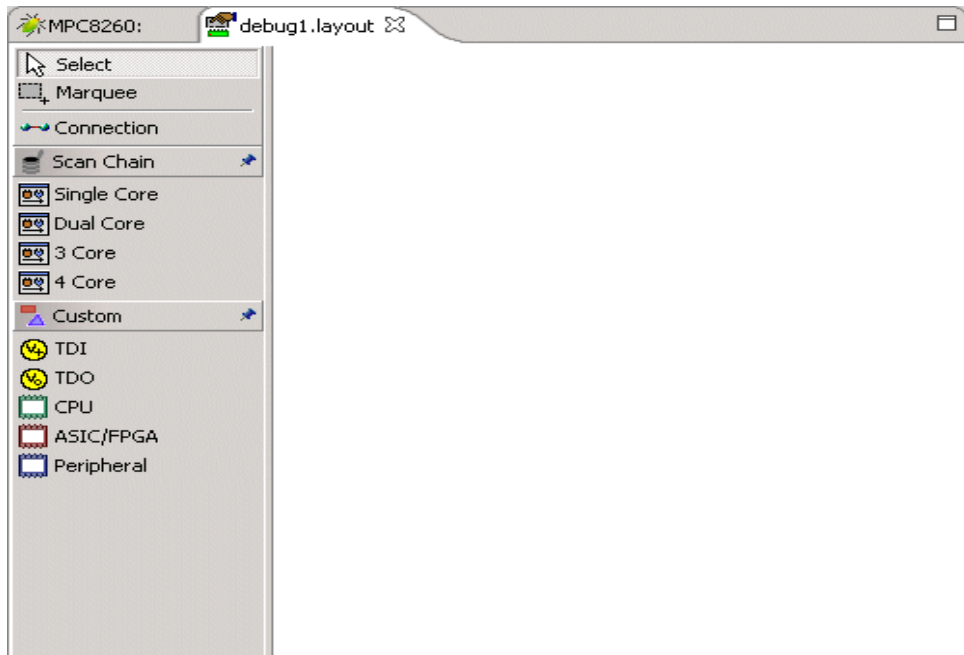
3. Click **Finish**.

This opens the JTAG Editor view, as shown in [Figure 6-2](#).



NOTE: JTAG Editor edits a **.layout** file, which is a graphic representation of the board layout. A **.brd** file cannot be created until you have created a JTAG layout, such as the one shown in [Figure 6-4](#).

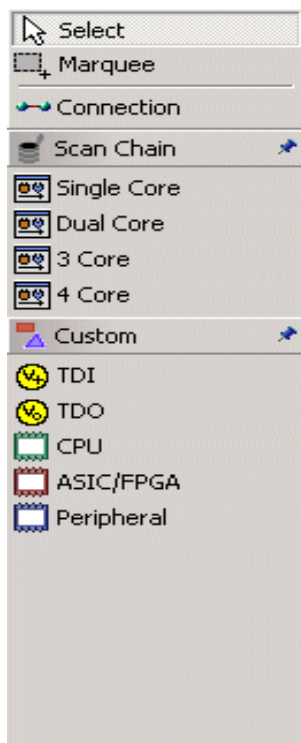
Figure 6-2 **JTAG Editor**



Using the Predefined Layouts in JTAG Editor

JTAG Editor includes predefined graphic layouts for one, two, three, and four cores, which are displayed in the Editor toolbar to the left of the editing field, as shown in [Figure 6-3](#).

Figure 6-3 JTAG Editor Toolbar



In the rare case where you need to debug more than four cores at the same time, the JTAG Editor also includes a **Custom** option. See [Using the Custom Option in the JTAG Editor View](#), p.95, for more information.

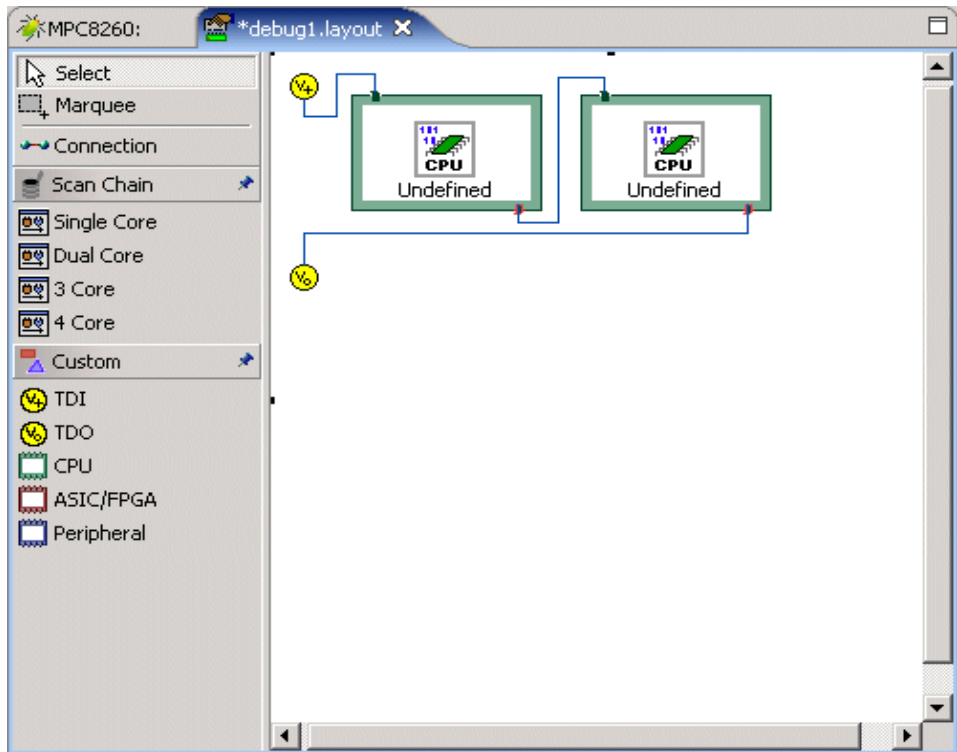
4. In the JTAG toolbar, click **Select**.
5. Under **Scan Chain**, pick the number of cores you need to debug.

For example, if the `debug1` project has two cores, click on **Dual Core** under the **Scan Chain** heading and drag it into the editing field, as shown in [Figure 6-4](#).



NOTE: The core icon must be clicked and dragged into the editing field. Just clicking on it will not do anything.

Figure 6-4 **Dual Core Layout**



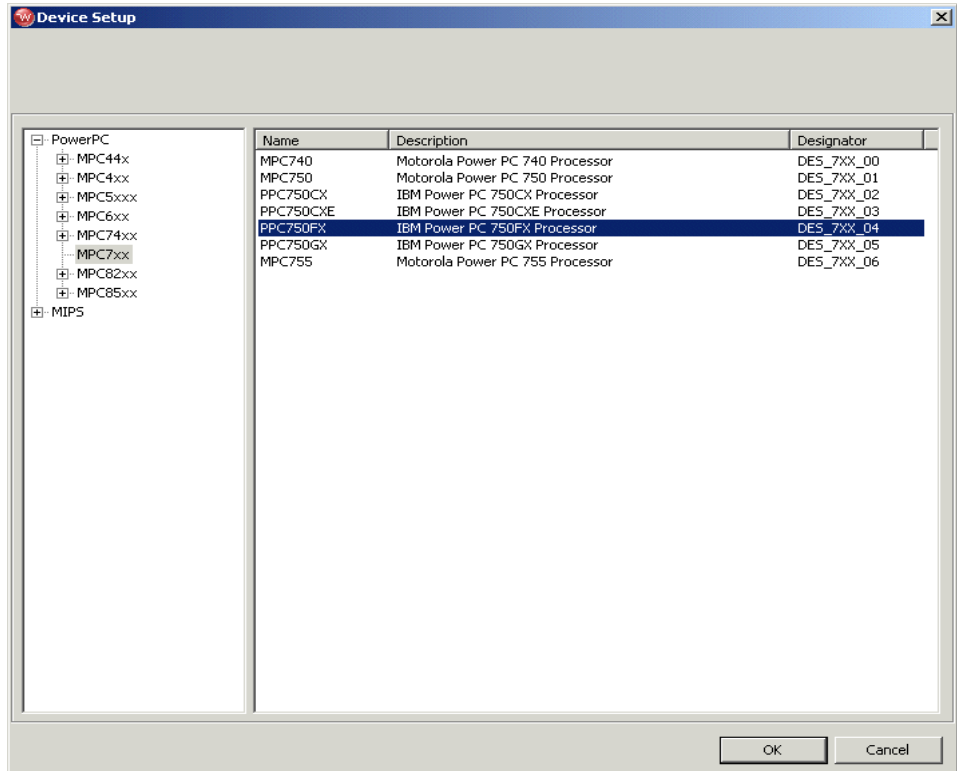
NOTE: You can only drag one predefined layout into the editing field at a time. If you drag in a second layout, it will overlay the first, causing confusion.

The editing field now shows a graphic representation of the scan chain. Notice that the two cores are labelled **Undefined**. They have no properties until you assign them in the next step.

6. Double-click on the first core.

The **Device Setup** dialog appears, as shown in [Figure 6-5](#).

Figure 6-5 Device Setup

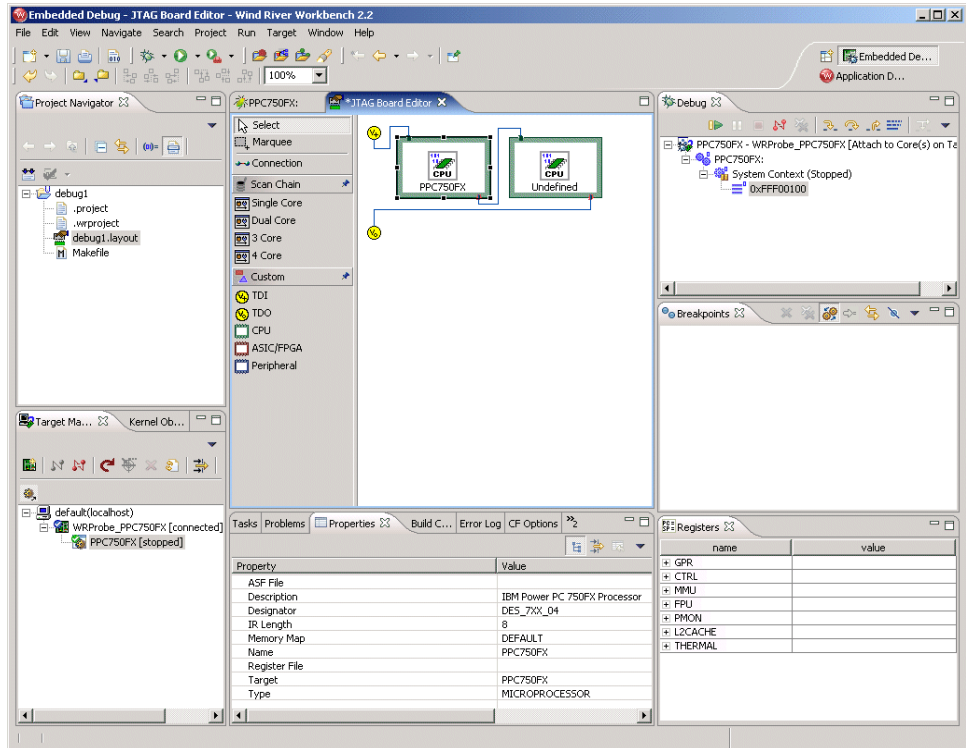


Use the dialog to select your processor type. The example in [Figure 6-5](#) shows a PPC750FX processor.

7. Click **OK**.

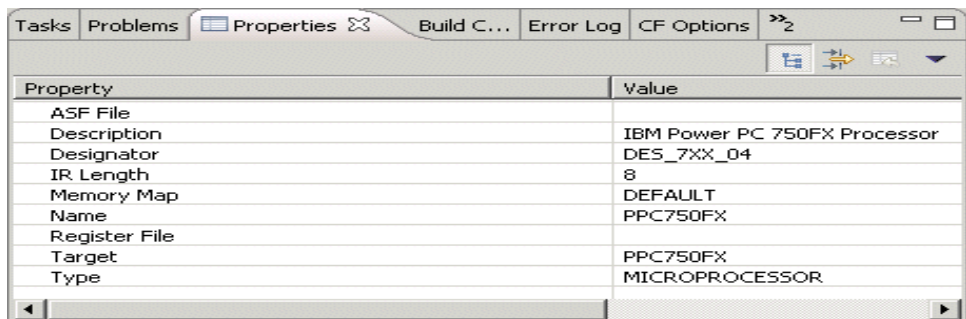
You are returned to the **Device Debug** Perspective. The first core is now defined as a PPC750FX, and the **Properties** view is displayed.

Figure 6-6 Defining the Core



You can use the **Properties** view to finish defining the first core.

Figure 6-7 Properties View



8. Click on any property to modify it.

Clicking on the **Register File** property will open a browser window; use the browser to navigate to the register file you want to use.

Your first core is now defined. To define your second core, double-click on it and repeat Steps 6 through 8.

If both cores use the same processor type, make sure you edit the **Designator** value in the **Properties** view. Workbench does not allow two cores to have the same unique designator. For example, in [Figure 6-7](#) the first core's designator is **DES_7XX_04**. If your second core is the same processor type as the first, the same designator will appear in the **Properties** window. Click on the **Designator** value to change it to (for instance) **DES_7XX_05**.

Once you have defined all your cores, you can create your board file.

9. Right-click on the editing area. In the dialog that appears, choose **Export Board File**.
A browser window appears. Choose the folder you want to save your board file in.
10. In the **File Name** field, type the name you wish to assign to your board file.
In the example, the board file name is **debug1.brd**.
11. Click **Save**.

Using the Custom Option in the JTAG Editor View

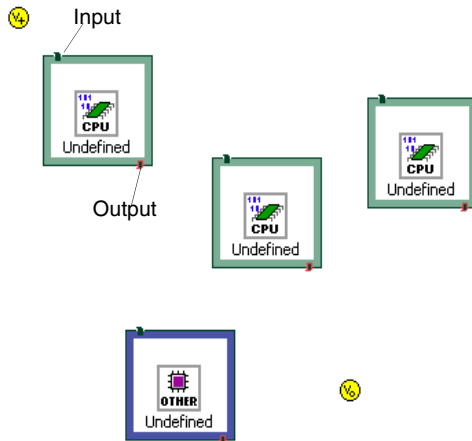
In the rare case where you need to debug more than four cores at the same time, JTAG Editor uses a **Custom** option to create a new board file piece by piece.

1. In the JTAG toolbar ([Figure 6-3](#)), click **Custom**.
2. Construct your layout using the elements under the **Custom** heading.

The elements available are an input node (TDI) and a termination node (TDO), as well as CPUs, ASICs, FPGAs, and peripherals. To add an element, click on its icon and drag it into the editing field.

[Figure 6-8](#) shows a partially completed layout with an input, a terminator, three CPUs, and a peripheral device.

Figure 6-8 Partial Custom Layout



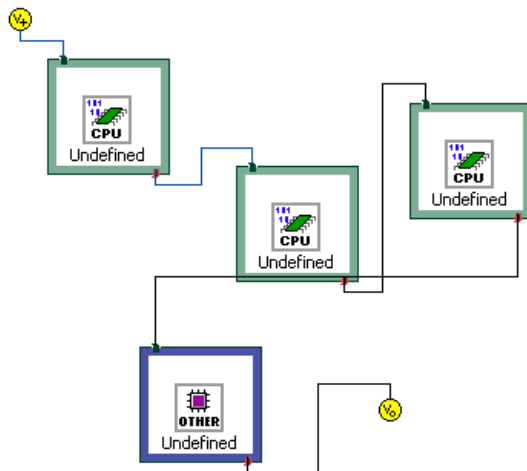
Once you have your terminating nodes and devices laid out, you need to connect them.

3. In the JTAG toolbar, click **Connections**.

When you move the cursor back into the editing field, it now looks like a power cord.

4. Click on the input node.
5. Move the cursor to your first processor and click again.
A connection line joins the input node and the processor.
6. Click on the first processor, move the cursor to the second processor, and click on it.
A connection line joins the two processors.
7. Continue this process until you complete the circuit by clicking on the terminator node.

Figure 6-9 Completed Custom Layout



8. When you have connected all devices and nodes, click **Connections** again. The cursor returns to normal.

Your custom board is now laid out. Define its properties and generate your **.brd** file by following Steps 6-11 in *Using the Predefined Layouts in JTAG Editor*, p.91.

Editing Your Board Layout

To remove a device, node, or connection from your layout, use the **Select** button or the **Marquee** button in the JTAG toolbar.

To use the **Select** button, click **Select** in the toolbar. Then click on any device, node, or connection to highlight it and press **Delete**.

To use the **Marquee** button, click **Marquee** in the toolbar. You will see that the cursor now appears as a crosshair in the editing field. Hold the mouse button down and drag the cursor to create a box around the device you wish to highlight, then press **Delete**.

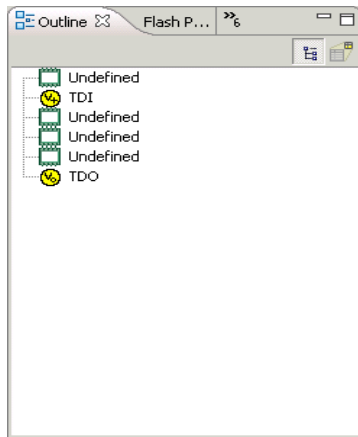


NOTE: The **Marquee** button can only highlight devices, not nodes or connections.

You can also edit your layout using the **Outline** view in Workbench. In the Workbench toolbar, click on **Window**. Select **Show View > Outline**.

The **Outline** view appears as shown in [Figure 6-10](#).

Figure 6-10 **Outline View**



The **Outline** view displays the elements of your layout in the order they were added. Click on any element to highlight it and press **Delete**.

Using the **Outline** view in this way is handy if you have accidentally overlaid one layout on top of another, or if you want to back up and start again. Use the list in the **Outline** window to delete any or all of the contents of the JTAG editing field.

6.4 XML Board Files

Board descriptor files are created in extensible Markup Language (XML). You can view the XML version of your board file by opening your **.brd** file in a text editor, or by selecting **File > Open** in Workbench and navigating to the **.brd** file in the

browser window that appears. The XML text will appear in the Workbench Editor. An example board descriptor file is shown below.

Figure 6-11 Board File XML version

```

<DEVICE_TABLE>
  <TABLE_MODE>SLOW</TABLE_MODE>
  <TABLE_CLOCK>16Mhz</TABLE_CLOCK>
  <TABLE_MULTI>ENABLE</TABLE_MULTI>
  <TABLE_TIED_RESET>OFF</TABLE_TIED_RESET>
  <DEVICE>
    <NAME>PPC750FX</NAME>
    <DESCRIPTION>IBM Power PC 750FX Processor</DESCRIPTION>
    <TYPE>MICROPROCESSOR</TYPE>
    <TARGET>PPC750FX</TARGET>
    <SELREG_FILE></SELREG_FILE>
    <DESIGNATOR>DES_7XX_04</DESIGNATOR>
    <IR_LEN>8</IR_LEN>
    <ASF_FILE></ASF_FILE>
    <REG_FILES>
    </REG_FILES>
    <MEMORY_MAP>
      <MEMORY_MODE>DEFAULT</MEMORY_MODE>
    </MEMORY_MAP>
  </DEVICE>
  <DEVICE>
    <NAME>PPC750FX</NAME>
    <DESCRIPTION>IBM Power PC 750FX Processor</DESCRIPTION>
    <TYPE>MICROPROCESSOR</TYPE>
    <TARGET>PPC750FX</TARGET>
    <SELREG_FILE></SELREG_FILE>
    <DESIGNATOR>DES_7XX_05</DESIGNATOR>
    <IR_LEN>8</IR_LEN>
    <ASF_FILE></ASF_FILE>
    <REG_FILES>
    </REG_FILES>
    <MEMORY_MAP>
      <MEMORY_MODE>DEFAULT</MEMORY_MODE>
    </MEMORY_MAP>
  </DEVICE>
</DEVICE_TABLE>

```

This is the `debug1.brd` board file created in *Using the Predefined Layouts in JTAG Editor*, p.91. The first block of code contains comments that describe what the target reference design is set for; the next blocks of code define the devices included in the file.

For information on board file fields, see *6.4.1 XML Board File Fields*, p.100.



NOTE: If you choose to modify a board descriptor file shipped with your system, it is best to save your modified file with a different name to prevent overwriting the default file.

6.4.1 XML Board File Fields

The board descriptor file contains comments, <DEVICE_TABLE >fields, and one or more <DEVICE> field-sets. A <DEVICE_TABLE> specifies common and rudimentary scan-chain (signal) operational functions and provides a list of <DEVICE> descriptions for each device sharing the JTAG interface.

<DEVICE_TABLE> Fields

<TABLE_MODE>

This field designates the scan-chain characteristics applicable to the devices on the chain. It can be set to FAST or SLOW. This also relates to the optimization implementation on the emulator. When in doubt, set it to SLOW.

<TABLE_CLOCK>

This field specifies the JTAG strobe rate, in MHz, for the information signals Test Data In (TDI) and Test Data Out (TDO). This is analogous to the emulator configuration option **CF CLK** *clock_rate*. They are not always automatically synchronized, so check your emulator to make sure you have the **CF CLK** option set to the same clock rate specified in the board file. The fastest JTAG clock rate is 16 MHz.

<TABLE_MULTI>

Set this field to ENABLE if you are debugging multiple targets on the same JTAG interface. Otherwise set it to DISABLE.

<TABLE_TIED_RESET>

Set this field to ON only if your target board's **RESET** and **TRST** signals on the JTAG interface are physically connected (tied together.)

<DEVICE> Fields

<NAME>

A reference name for the target device.

<DESCRIPTION>

A reference description of the target device.

<TYPE>

The valid types are MICROPROCESSOR, CPLD, FPGA, INTERFACE, and OTHER.

<TARGET>

The CPU type. The run-time processes on Wind River emulators require this information in order to match the exact JTAG scan chain and JTAG-specific characteristics.

<DESIGNATOR>

A mandatory field that Workbench uses to distinguish between devices. Typically this is set to U0, U1, U2....

Make sure you use a unique <DESIGNATOR> tag for each target device. Workbench does not allow two devices to use the same designator.

<IR_LENGTH>

Use this field to specify the length, in bits, of the target device's JTAG Instruction Register. To find this information, consult the manufacturer's specification for the target device.

6.5 Manually Creating XML Board Files

If you need a custom board file, it is usually easiest to take one of the generic board files from *installDir/workbench-version/dfw/build/host/boardfiles* and modify it to suit your needs. Remember to save it with a different name if you want to preserve the original file.

To create a board file that properly describes the scan chain on your target:

1. Open a text editor.
2. Begin the board file with the tag <DEVICE_TABLE>.
3. Lay out the header block.

The first block of XML defines mode, clock speed, and status of multi-core debugging. An example would look like:

```
<TABLE_MODE>SLOW</TABLE_MODE>
<TABLE_CLOCK>16Mhz</TABLE_CLOCK>
<TABLE_MULTI>ENABLE</TABLE_MULTI>
<TABLE_TIED_RESET>ON</TABLE_TIED_RESET>
```

This example is set for slow mode, with a clock speed of 16 MHz; it is enabled for multi-core debugging, and it is set to issue **RST** reset commands (which affect all cores) rather than **IN** reset commands (which affect only one core.)

The next blocks of XML define the devices included in the file. Workbench needs this information so that it can position the devices in the correct location in the 25-bit data stream. The physical location of each device can also be determined by its position in the board descriptor file.

4. Lay out the block for the first device.

A device block begins with the tag `<DEVICE>`. An example would look like:

```
<DEVICE>
  <NAME>MPC8260</NAME>
  <DESCRIPTION>Motorola Power PC 8260 Processor</DESCRIPTION>
  <TYPE>MICROPROCESSOR</TYPE>
  <TARGET>MPC8260</TARGET>
  <DESIGNATOR>U0</DESIGNATOR>
  <IR_LEN>8</IR_LEN>
</DEVICE>
```

This example describes a PowerPC 8260 target.

5. Repeat Step 4 for every device on the JTAG scan chain.

Your board file must list the devices included on your scan chain in the same order as they are physically laid out on the target. If the board file and the physical scan chain do not match, the board file for your target will not work.

When you are finished, your board file should look something like this:

```
<DEVICE_TABLE>
  <TABLE_MODE>SLOW</TABLE_MODE>
  <TABLE_CLOCK>16Mhz</TABLE_CLOCK>
```

```
<TABLE_MULTI>ENABLE</TABLE_MULTI>
<TABLE_TIED_RESET>OFF</TABLE_TIED_RESET>
<DEVICE>
  <NAME>MPC8260</NAME>
  <DESCRIPTION>Motorola Power PC 8260 Processor</DESCRIPTION>
  <TYPE>MICROPROCESSOR</TYPE>
  <TARGET>MPC8260</TARGET>
  <DESIGNATOR>U0</DESIGNATOR>
  <IR_LEN>8</IR_LEN>
</DEVICE>
<DEVICE>
  <NAME>PPC750FX</NAME>
  <DESCRIPTION>IBM Power PC 750FX Processor</DESCRIPTION>
  <TYPE>MICROPROCESSOR</TYPE>
  <TARGET>PPC750FX</TARGET>
  <DESIGNATOR>U1</DESIGNATOR>
  <IR_LEN>8</IR_LEN>
</DEVICE>
</DEVICE_TABLE>
```

This example describes two targets, but you can add as many <DEVICE> blocks as you need to describe your JTAG scan chain.

6. When you are finished, save the file with the extension **.brd**.

7

Debugging Multi-Core Targets

- 7.1 Introduction 105
- 7.2 JTAG Server 106
- 7.3 Multi-Core Debugging 107
- 7.4 Initializing the Targets 119
- 7.5 Creating a Project 125
- 7.6 Configuring Options for Multi-Core Debugging 131
- 7.7 Commands for Multi-Core Debugging 134

7.1 Introduction



NOTE: This chapter applies only to the Wind River ICE SX. Multi-core debugging is not supported for the Wind River Probe.

The Wind River ICE SX emulator allows you to control and manipulate multiple devices on a single scan chain ring. The devices included on the JTAG scan chain can be CPUs, EPLDs, CPLDs, FPGAs, and ASICs, as well as various other devices. Wind River ICE SX manages all of the devices on a scan chain through the use of JTAG Server, which resides on the ICE unit and works via the OCD link on the target.

Using the Wind River Workbench software and a Wind River ICE SX, users can have multiple debug sessions active at the same time, allowing developers to debug multiple devices at once.

JTAG Server is the software layer that allows ICE to handle multi-core debugging. JTAG Server is set up by a board descriptor file, which identifies the devices that are included in the scan chain on your target. Board descriptor files are highly specific to your target since they clearly describe the scan chain, so if you do not have a board file already available for your specific target, you must write one. For information on creating board files for your target, see [6. Using Board Descriptor Files](#).

7.2 JTAG Server

JTAG Server is a multi-core debugging solution that uses the On-Chip Debugging (OCD) link on your target reference design to connect to one or more CPUs, with or without other devices in the scan chain. JTAG Server is what allows ICE to connect to multiple devices on a single JTAG scan chain at once.

Wind River ICE SX is networked, and JTAG Server resides on the ICE unit. This means that in addition to being able to have multiple debug sessions with the target from a single host computer, you can also have multiple debug sessions running on multiple host computers, all accessing ICE over the network.

JTAG Server is a software layer that resides between the low-level JTAG drivers and the high-level user interfaces on Wind River ICE SX. This layer provides all of the control that is needed to position data correctly on the scan chain (this is a requirement for compliance with the IEEE 1149.1 specification for multiple devices on a single scan chain ring).

There are hardware optimizations included in JTAG Server that allow the utilization of the entire available JTAG bandwidth. This is a key element of high performance multi-core debugging; the peak clock speed is not as important as the aggregate data transfer, which is maximized in JTAG Server.

JTAG Server requires a board descriptor file that clearly describes your target scan chain layout. This file is what tells JTAG Server how to correctly position devices on the scan chain. Information on board descriptor files, including information about creating your own is available in [6. Using Board Descriptor Files](#).

7.3 Multi-Core Debugging

This section describes how to work with multiple devices on your scan chain at once. Before beginning, please make sure that you have obtained or created a board descriptor file for your target that accurately reflects your target's scan chain.

7.3.1 Establishing Communications with Multiple Devices

The following steps describe how to connect to multiple devices at once using the Wind River Workbench software.

1. First, open Workbench according to the method for your host computer.

Linux/Solaris Hosts

From your installation directory, issue the command

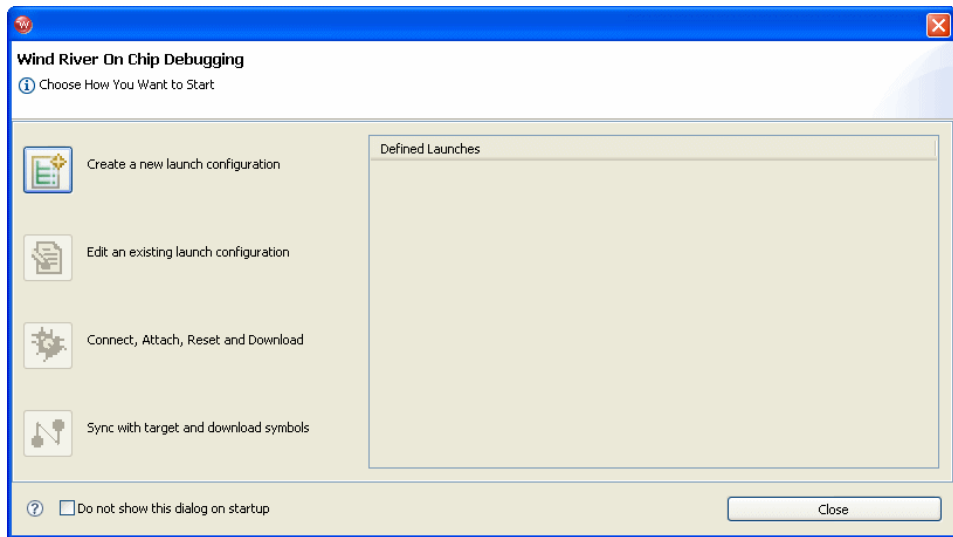
```
$ ./startWorkbench.sh
```

Windows Hosts

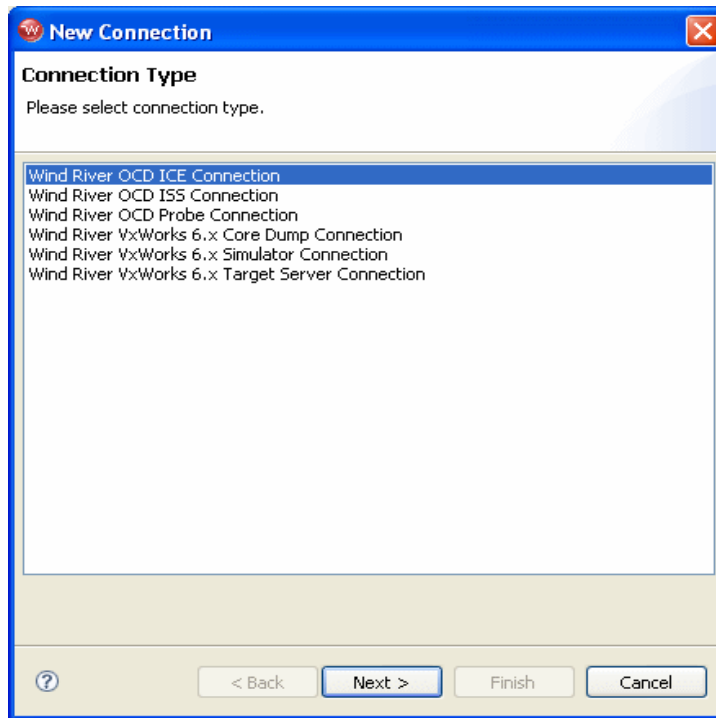
Select **Start > All Programs > Wind River > Wind River Workbench *version***.

On Windows hosts, Workbench prompts you to specify a workspace location. Linux/Solaris hosts use the default location *installDir*/**workspace**.

When Workbench opens, the **Quick Target Launch** dialog appears.

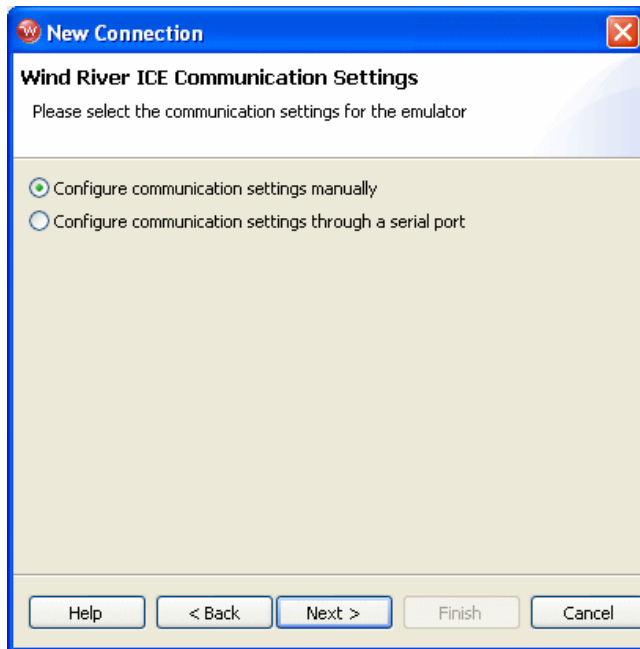


2. Select **Create a new launch configuration**.
The **Connection Type** dialog appears.



3. Choose **Wind River OCD ICE Connection** from the list of options and click **Next**.

The **Communication Settings** dialog appears.



Configuring Communication Settings Manually



NOTE: To use this option you will need to know either the network name of the emulator or its IP address. For information on assigning these values, see the *Wind River ICE SX for Wind River Workbench Hardware Reference*.

4. Select **Configure communication settings manually** and click **Next**.

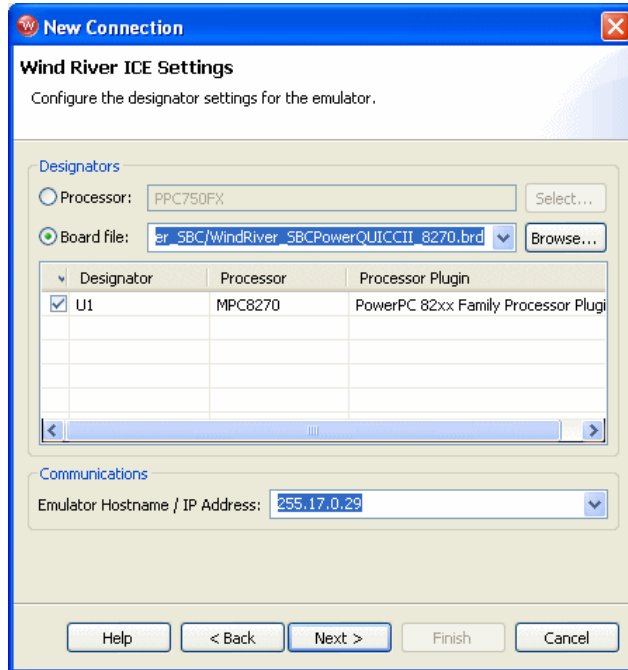
The **Emulator Settings** dialog appears.

5. In the **Designators** area, select **Board File** and click **Browse** to navigate to the board file that describes your multi-core setup. For information on board files, see [6. Using Board Descriptor Files](#).

The field below the **Board File** field will populate with a summary description of your board.

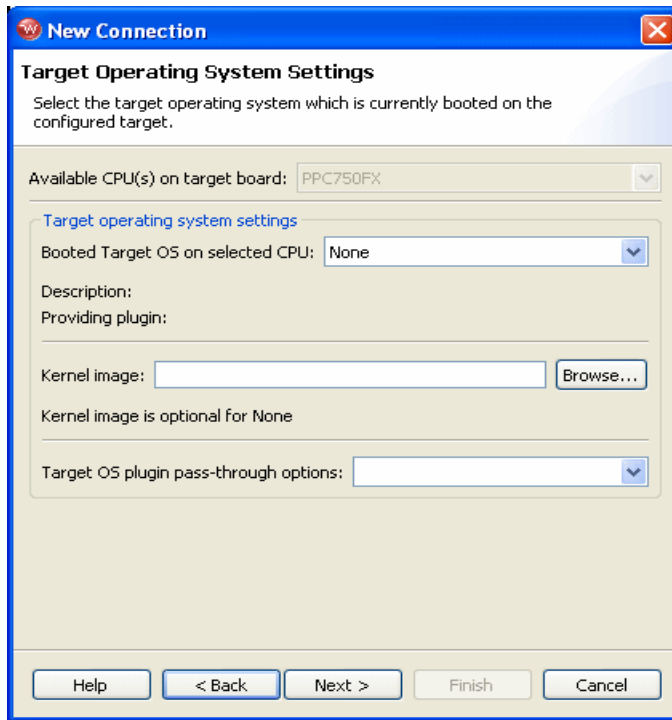
6. In the **Communications** area, fill in the **IP Address** field with the IP address you have assigned to your ICE unit.

This example shows the emulator settings dialog box with the board file for a Wind River SBC PowerQUICC II 8270 selected.



7. When you have entered the correct processor or board file and IP address, click **Next**.

The **Target Operating System Settings** dialog appears.



8. In the **Booted Target OS on selected CPU** field, select the operating system that is running on your target processor. The default is **None**.
9. Next to the **Kernel Image** field, click **Browse** to navigate to the kernel image you wish to specify. If you selected **None** in the previous step, you do not need to specify a kernel image.
10. If you are using a Linux plug-in specify the pass-through options in the **Target OS Pass-Through Options** field. If you are not using a Linux plug-in, skip this step.

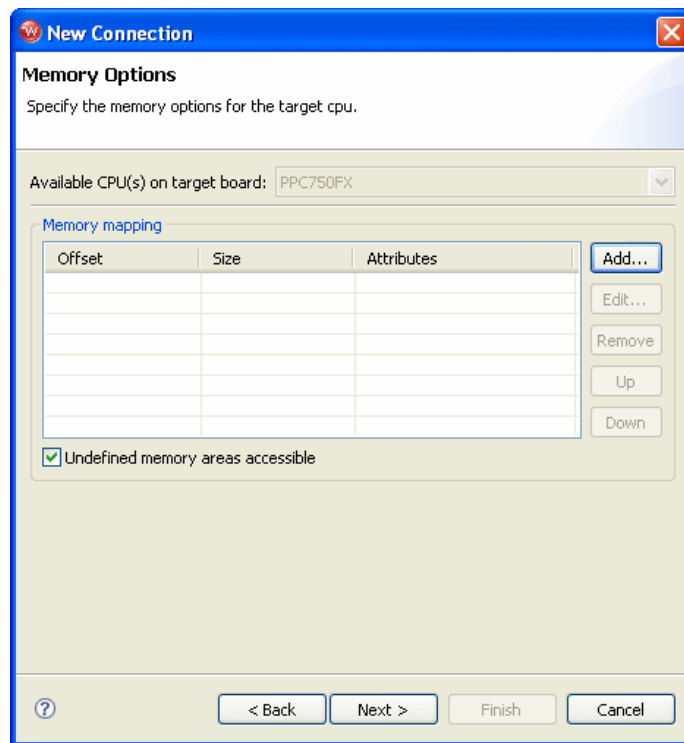
Options are passed as pairs in the format *name=value*. Separate options with a comma. The following options are available:

- **notasklist=1** : Never fetch process list.
- **noautomodules=1** : Do not plant internal breakpoints to do automatic kernel module load/unload detection. When this option is specified, you must manually refresh to see an updated module list.

- **noloadcheck=1** : Do not issue gophers until the hardware breakpoint is used to detect kernel load triggers. This option is for “sensitive” boards that don’t accept access until the kernel loads and sets up memory mapping.
- **loaddetectloc=symbol or address**: Set the hardware breakpoint used to detect kernel load at *symbol* (for example, **loaddetectloc=start_kernel**) or *address* (for example, **loaddetectloc=0x1000**). If you do not specify a symbol or address, Workbench uses a default. For most architectures the default is **start_kernel**; for PowerPC targets, the default is **0x0**.

11. Click Next.

The **Memory Options** dialog appears.

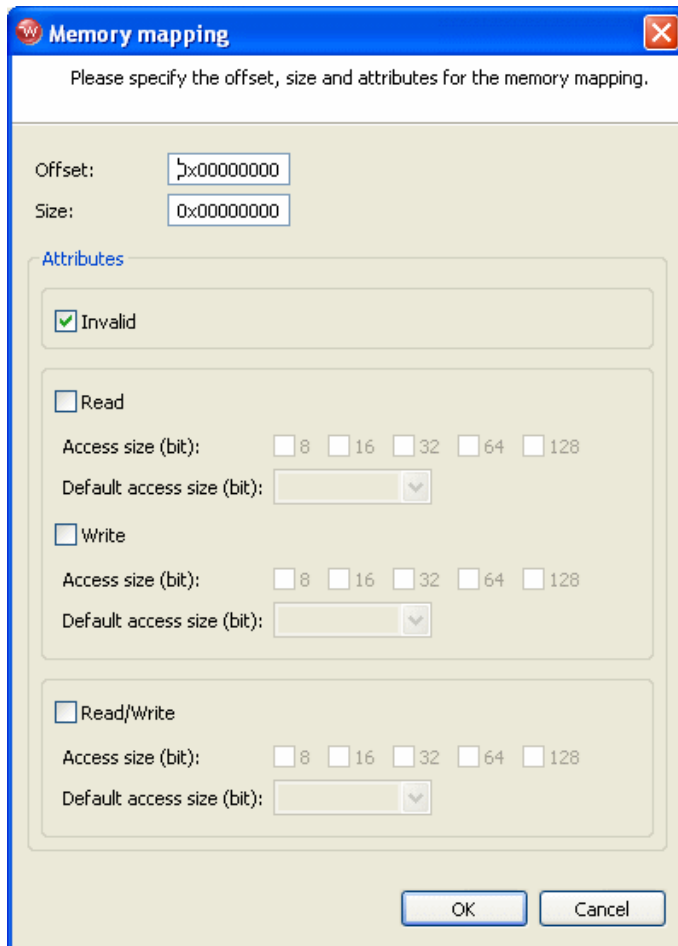


Use the **Memory Options** dialog to specify how memory on the target is partitioned, and what the attributes of the particular memory regions are.

➔ **NOTE:** The **Memory Options** dialog is only necessary for Linux or other non-VxWorks target operating systems.

To specify an area of memory, click **Add**.

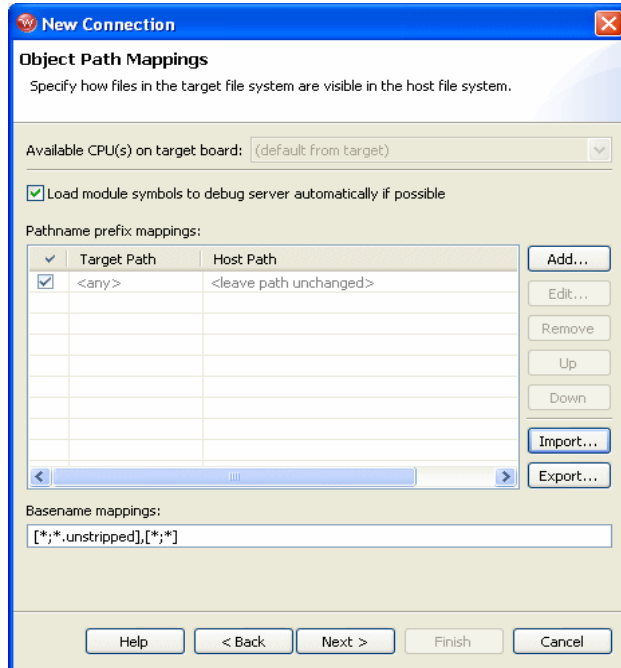
The **Set Memory Map** dialog appears.



Use the **Set Memory Map** dialog to specify which memory areas are read-only, read-write, or write-only, and to specify the access width Workbench should use to read the data from those regions.

12. Click **Next**.

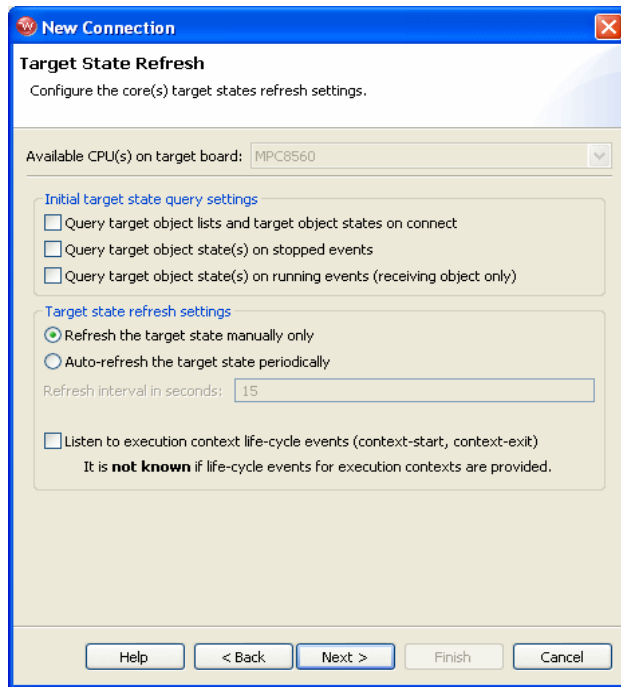
The **Object Path Mappings** dialog appears.



Use the **Object Path Mappings** dialog to specify how files in the target file system are visible in the host file system.

13. To add a host or target path, click **Add** and type the path in the dialog that appears.
14. Click **Next**.

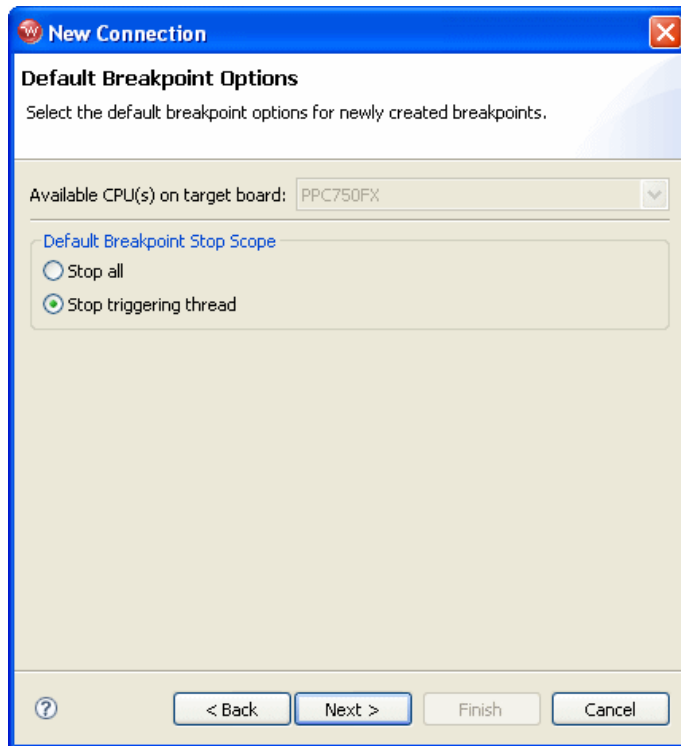
The **Target State Refresh** dialog appears.



Use the **Target State Refresh** dialog to configure the target state query and target state refresh settings on your target processor.

15. Click **Next**.

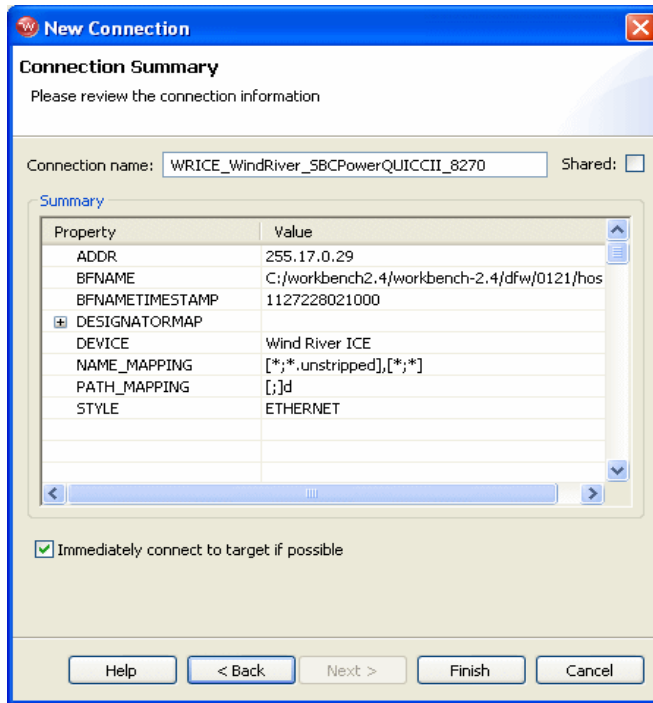
The **Default Breakpoints** dialog appears.



Use this dialog to set default breakpoint options for newly created breakpoints.

16. Click **Next**.

The **Connection Summary** dialog appears.



17. Verify that the displayed values are correct.

If you want to connect to your target now, select **Immediately connect to target if possible**. If you do not wish to connect to your target now, leave the **Immediately connect to target if possible** box unchecked. You can connect at any time by clicking the **Connect** button in the **Launch Configuration** dialog.

18. If you want to share your target connection, select **Shared**.

This option serves a dual purpose:

- When you define a target connection configuration, this connection is normally only visible for your user-id. If you define it as **Shared**, other users can also see the configuration in your registry, provided that they connect to your registry by adding it as a remote registry on their computer.
- Normally, when you disconnect a target connection, the target server (and simulator) are killed because they are no longer needed. In a connection that is flagged as **Shared**, however, they are left running so that other users can

connect to them. In other words, you can flag a connection as shared if you want to keep the target server (and simulator) running after you disconnect or exit Workbench.

19. Click **Finish**.

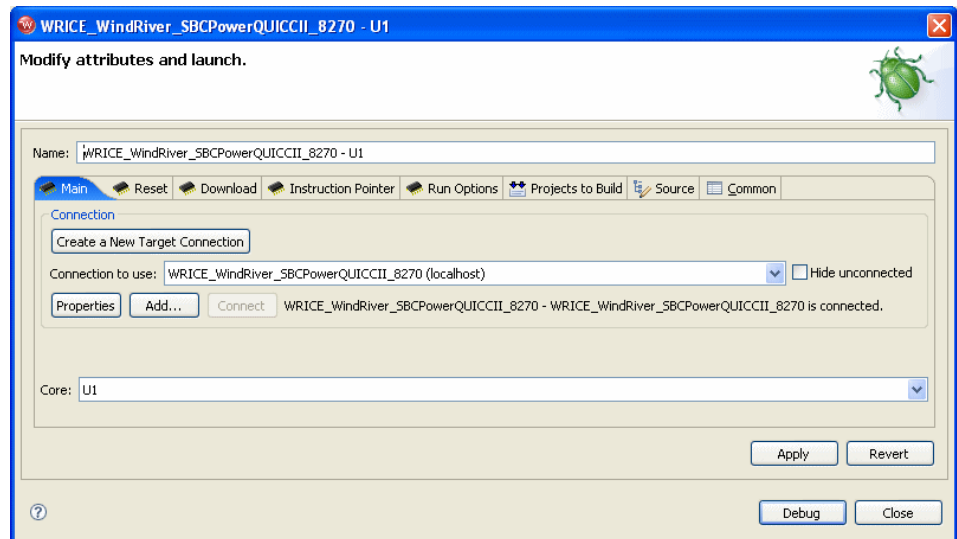
Workbench creates the connection name in the **Target Manager** view.

If you decide you want to change your emulator settings, you can return to the **Emulator Settings** dialog box by right-clicking on the connection name in the **Target Manager** and clicking **Properties**.

Your Wind River ICE SX is now connected to the host computer and your target.

7.4 Initializing the Targets

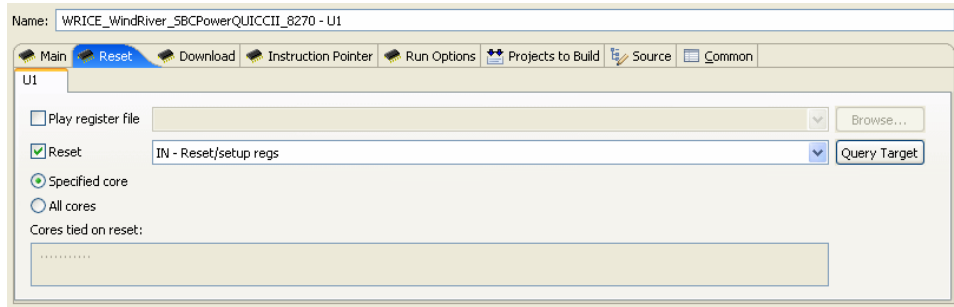
After Workbench connects to the Wind River ICE SX, the **Reset and Download** view opens.



20. Choose how you want to proceed:

- a. If you want to create a project in which to run and debug your code, skip the rest of this section and proceed to [7.5 Creating a Project](#), p.125.
- b. If you want to run and debug your code without creating a project, continue with this section.

21. In the **Reset and Download** view, select the **Reset** tab.



22. If you want to configure the target register values with a register file, select **Play Register File** and browse for the file you want to use.

Register files for many Wind River-supported targets are located in *installDir/workbench-2.x/dfw/build/host/registers*.

If you do not want to reconfigure your target registers, leave this box unchecked.

23. Choose the type of reset initialization you want to perform.

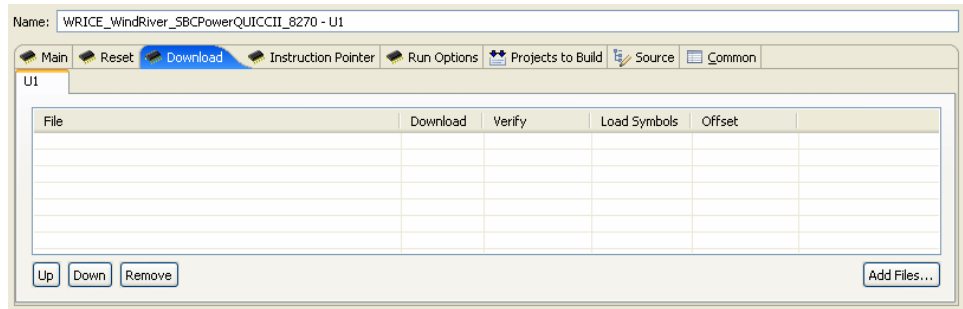
You can use the **IN** or **INN** initialization commands. For a full discussion of these two commands, see the *Wind River Workbench for On-Chip Debugging Command Reference*.

You can also choose not to perform an initialization by clearing the **Reset** box.



CAUTION: If you are manually changing registers on your target, be aware that issuing an **IN** or **INN** initialization command will overwrite your changes.

24. By default, the reset and download affects only the first core. If you want the reset and download to affect all connected cores, select the **All Cores** radio button and click **Apply**.
25. Select the **Download** tab.



26. Click **Add Files**.

In the browser window that appears, navigate to the executable file you want to run.

The file you select appears in the **Filename** field. Repeat this process as many times as necessary.

The file at the top of the list will download to the target first, followed by the others from the top down. You can edit the order of the list by clicking on any filename to highlight it and using the **Up**, **Down**, and **Delete** buttons.

27. Use the other fields to configure the download.

Download

The **Download** field is checked by default. If you clear it, the file will remain on the list but will not download data to the target. This is useful if, for example, you only want to download symbol information and not data.

Verify

The **Verify** field configures the extent to which the file you are downloading will be compared to a file that may already be on the target. There are three options: **Full**, **Compare**, and **None**.

When this field is set to **Full**, a write/read verify will occur for every download. Workbench will write to the target and then verify that the write to the target and the read from the target are identical. This is slower than a normal download, but it is a useful security option.

When the field is set to **Compare**, Workbench will verify that the image has been downloaded correctly (that is, that the image on the host is the same as the image on the target.) This is useful for programming flash.



NOTE: You should only set the **Verify** field to **Compare** if an image already exists on the target. If you set the field to **Compare** when there is no image on the target, Workbench will look for a file to compare and not find one, and the reset and download operation will fail.

When the field is set to **None**, Workbench will perform no verification.

The **Verify** field is set to **None** by default.

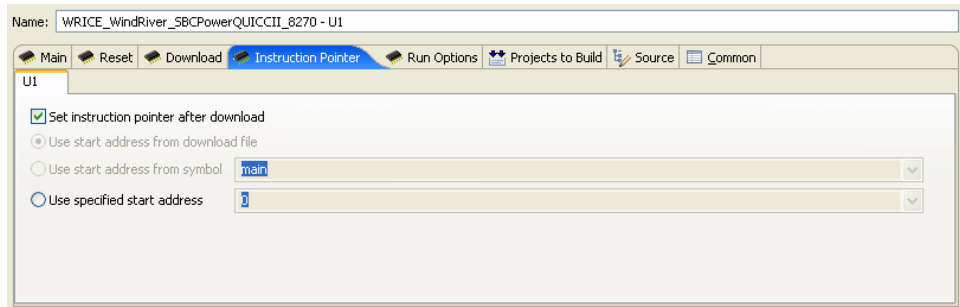
Load Symbol

The **Load Symbol** field, which is checked by default, determines whether the file's symbol information is downloaded to the target.

Offset

In the **Offset** field, you can enter a value in hex to set a memory offset bias for your application file. If you do not enter a value, Workbench uses the default value **0x00000000**.

28. Select the **Instruction Pointer** tab.



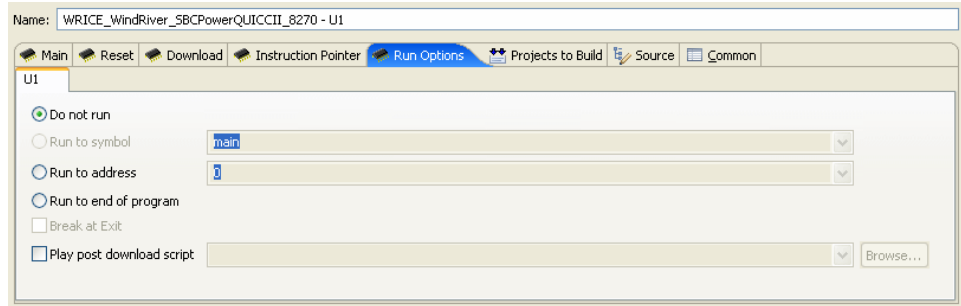
29. Set the starting point for your file.

By default, the instruction pointer is set to use the starting address from the download file.

You can set the instruction pointer to start the file from the first occurrence of a particular symbol (for example, **main**) or you can just specify a starting address by entering the address value in hex in the **Use Specified Start Address** field.

If you do not want to set a starting point, clear the **Set Instruction Pointer After Download** box.

30. Select the **Run Options** tab.



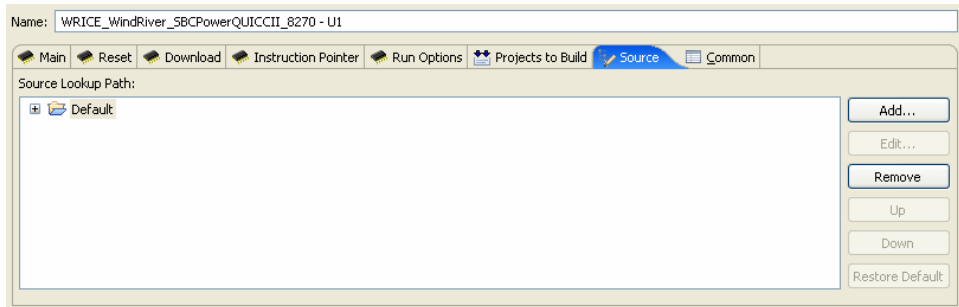
31. Determine how you want your file to run.

By default, the **Reset and Download** view is set not to run the file after downloading. If you want the file to run, you have several options to determine where it should break:

- You can set it to break at the first occurrence of a symbol (for example, **main**) by selecting **Run to Symbol** and entering the symbol in that field.
- You can set it to break at the end of your program by selecting **Run to end of program**.
- You can set it to break at a given memory address by selecting the **Run to Address** box and entering the address in hex in that field.
- You can set it to break at an **_exit** routine by selecting the **Break at Exit** box.

If you need to perform a post-initialization, you can define it here. Select the **Play post download script** box and click **Browse**. In the browser window that appears, navigate to your initialization file.

32. Select the **Source** tab.

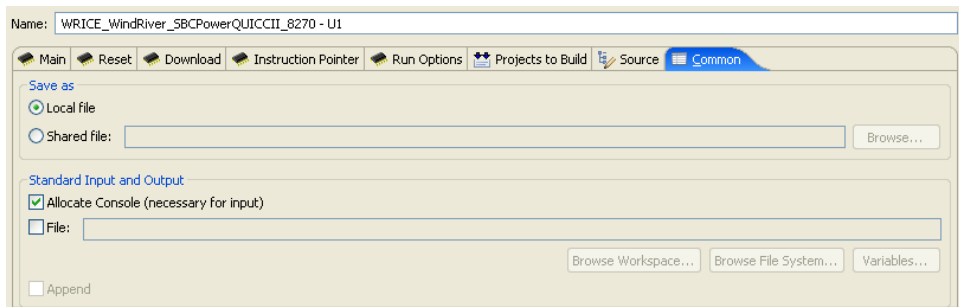


33. Use the **Source** tab to configure the source path of your file.

Workbench uses the input path of the local file system by default. Unless you need to use a different path, you do not need to do anything in the **Source** tab.

If you need to use a different path, click **Add...** and use the **Add Source** dialog to configure the appropriate search path for your project.

34. Select the **Common** tab.



35. Specify whether your launch configuration is local or shared.

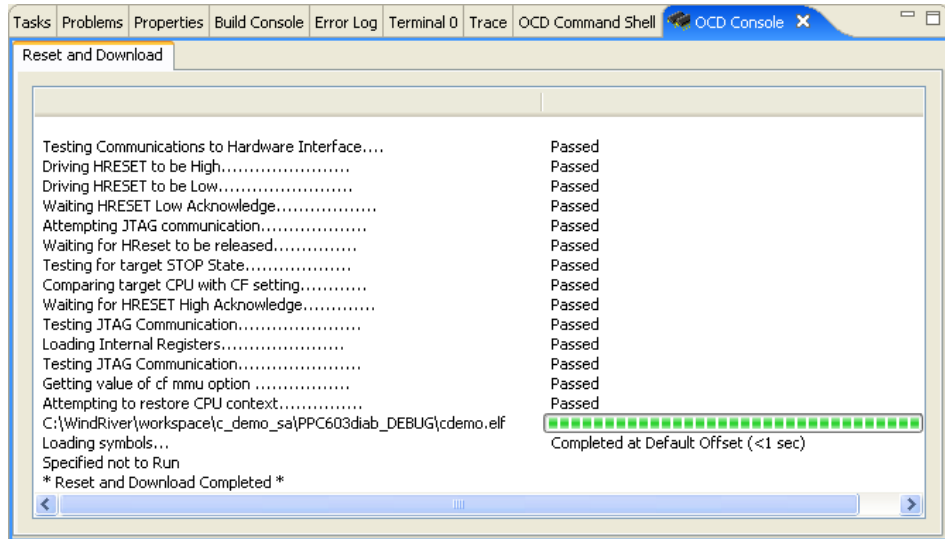
The configuration is local by default. To make it shared, click **Shared file:** and browse to the shared directory where you want the configuration to be located.

You have now fully defined your reset and download operation.

36. Click **Debug**.

Workbench initializes the target board, then downloads the file, then runs the file.

The **OCD Console** view opens to show the progress of the reset and download operation.



At this point, all of your devices should be in background mode and you are ready to begin debugging all of the devices.



NOTE: If you cannot get into background mode with any of your devices, first make sure your board file is correct for your target. Then see the *Wind River ICE SX for Wind River Workbench Hardware Reference* for troubleshooting tips.

Proceed to [7.6 Configuring Options for Multi-Core Debugging](#), p.131.

7.5 Creating a Project

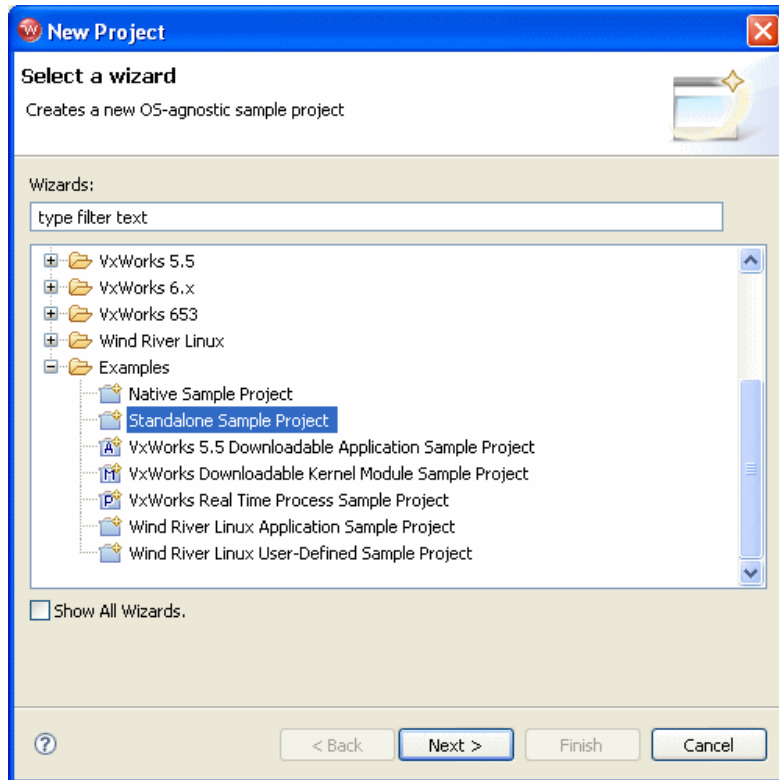
If you do not plan to build or edit your source files within Workbench, skip this section and proceed to [7.6 Configuring Options for Multi-Core Debugging](#), p.131.

Click **Close** in the **Reset and Download** view.

This tutorial uses the C Demonstration Program, which is included in your Workbench installation.

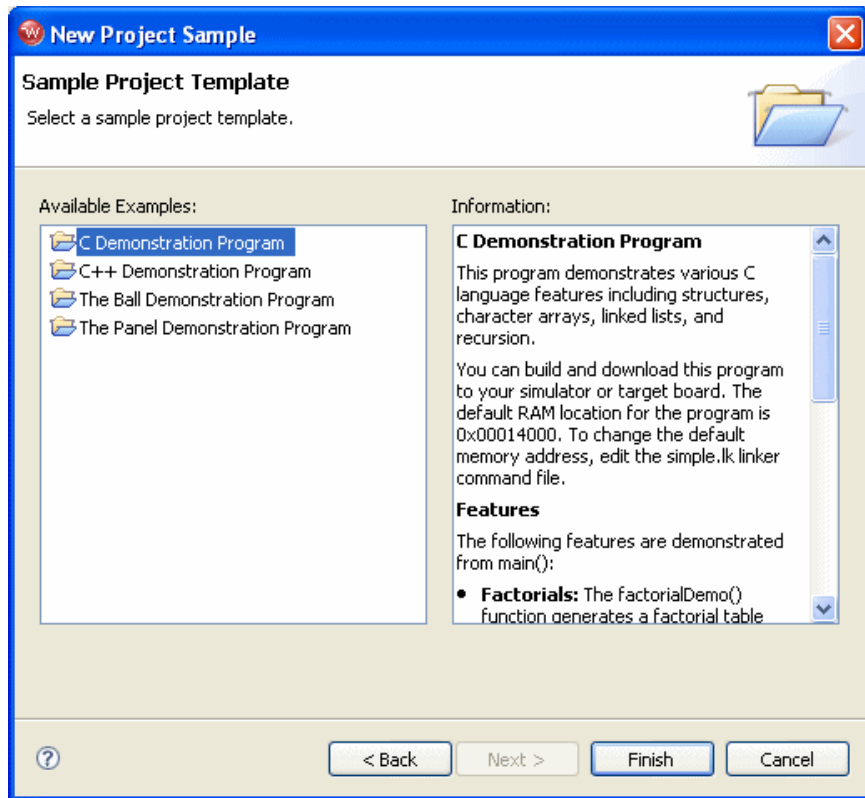
1. In the Workbench toolbar, select **File > New > Project**.

The **New Project** wizard appears.



2. Expand the **Examples** folder and select **Standalone Sample Project**.
3. Click **Next**.

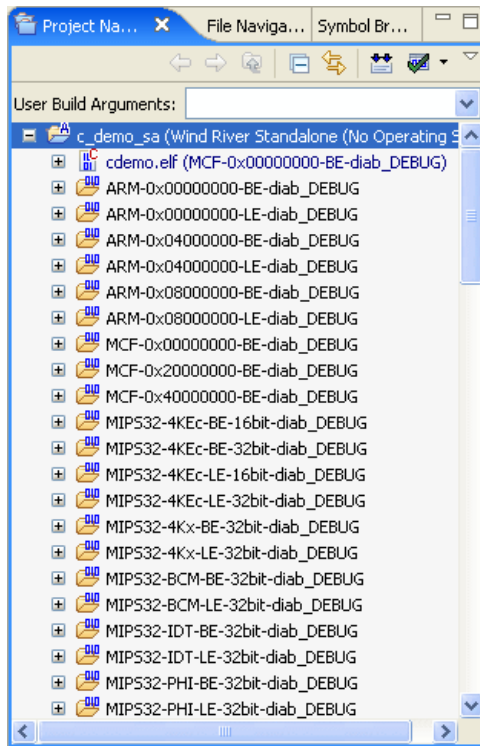
A sample project template appears.



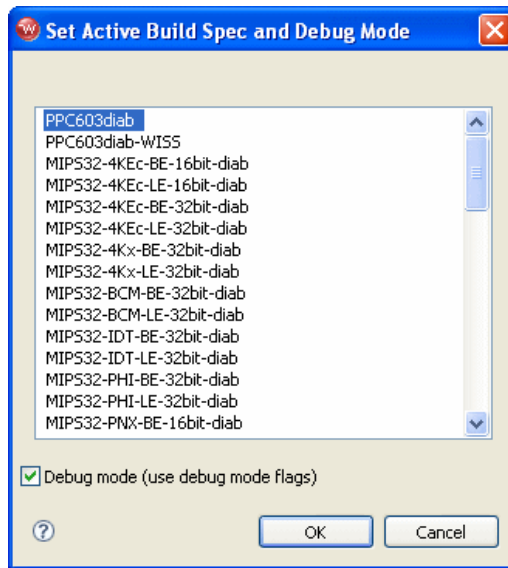
4. Select **C Demonstration Program** and click **Finish**.

Workbench creates the sample project in the default workspace folder and opens the **Application Development** perspective.

5. In the **Project Navigator** view, expand the **c_demo_sa** project.

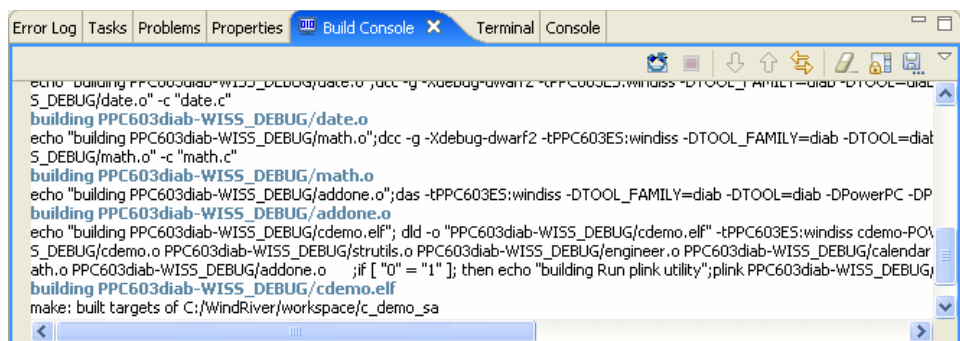


6. To build the sample project for use with PowerPC targets, right-click on the `c_demo_sa` top-level folder and select **Build Options > Set Active Build Spec**.
The **Set Active Build Spec and Debug Mode** dialog appears.



7. Scroll to the top and highlight **PPC603diab**.
8. Select **Debug mode (use debug mode flags)** so Workbench will generate symbolic debug information.
9. Click **OK**.
10. Right-click on the project name and select **Rebuild Project**.

Workbench builds the sample project. The results of the project build appear in the **Build Console** view.

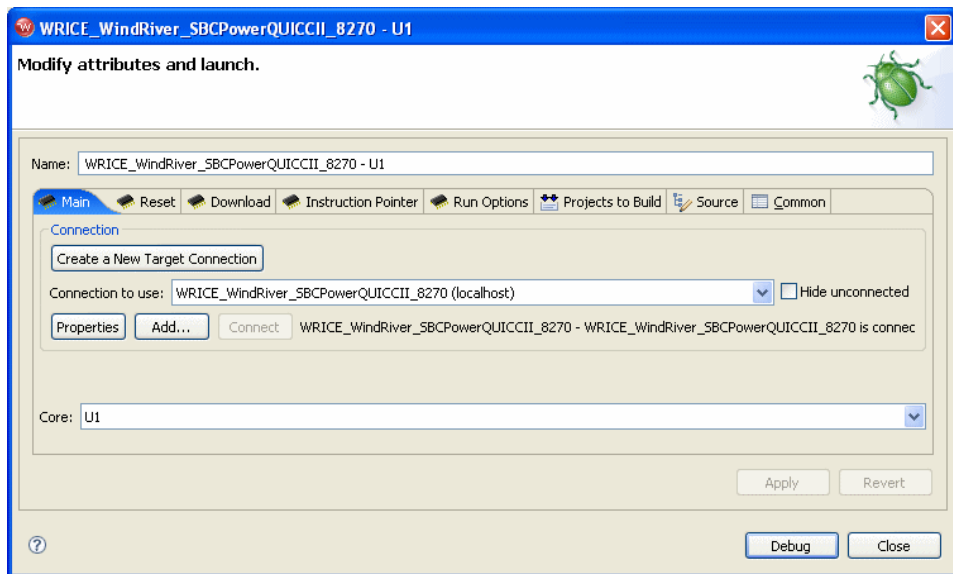


7.5.1 Downloading the Sample Code

To run the sample code, use the following steps:

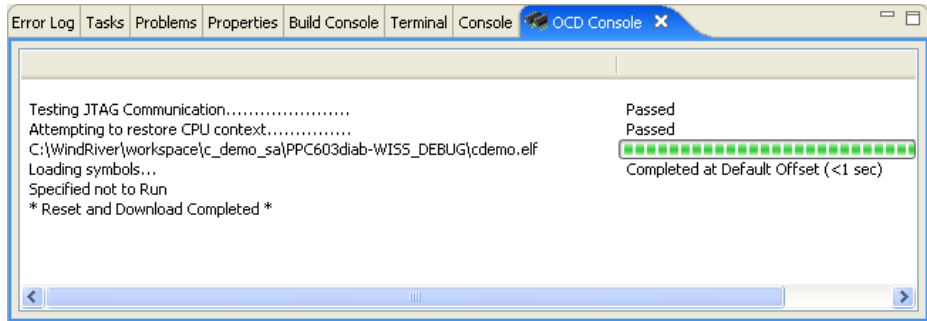
1. In the **Target Manager**, highlight the target connection name.
2. In the **Project Navigator** view, right-click on **cdemo.elf** and select **Reset and Download**.

The **Reset and Download** view appears.



3. Leave all settings at their defaults and click **Debug**.

The **OCD Console** view opens.



The **OCD Console** view shows the progress of the download operation, as Workbench downloads the sample code to the Wind River Instruction Set Simulator.

7.6 Configuring Options for Multi-Core Debugging

For multi-core debugging, the **Reset** and **Download** options will detect if the `<TABLE_TIED_RESET>` tag in the board file is set to **ON**; if it is, the **Reset** and **Download** options will issue **RST** commands (which affect all cores) rather than **IN** commands (which affect only one core.) You can use the JTAG Editor to set this option in your XML board file.

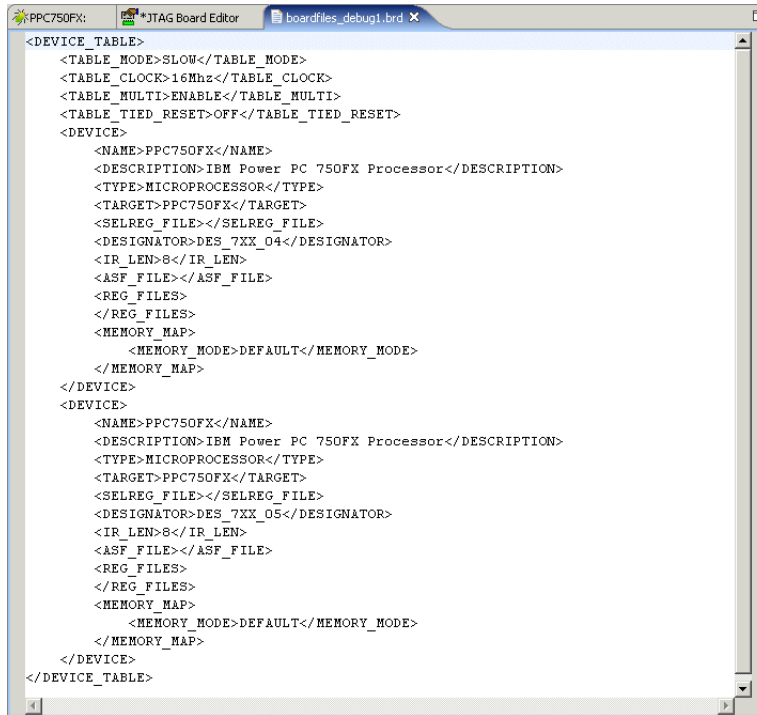
In conjunction, there are two configuration options that you may want to change if you are debugging multiple devices on your target: **HRESET** and **CMDRST**. These must be set correctly for each core you are debugging. For explanations of these options, see [CF HRESET](#), p.133, and [CF CMDRST](#), p.133.

To configure for multi-core debugging:

1. In the Workbench toolbar, select **File > Open > yourBoardFile**.

[Figure 7-1](#) shows the example board file **debug1.brd**. Notice that the `<TABLE_TIED_RESET>` tag is set to **OFF**.

Figure 7-1 debug1.brd



```
<DEVICE_TABLE>
<TABLE_MODE>SLOW</TABLE_MODE>
<TABLE_CLOCK>16Mhz</TABLE_CLOCK>
<TABLE_MULTI>ENABLE</TABLE_MULTI>
<TABLE_TIED_RESET>OFF</TABLE_TIED_RESET>
<DEVICE>
  <NAME>PPC750FX</NAME>
  <DESCRIPTION>IBM Power PC 750FX Processor</DESCRIPTION>
  <TYPE>MICROPROCESSOR</TYPE>
  <TARGET>PPC750FX</TARGET>
  <SELREG_FILE></SELREG_FILE>
  <DESIGNATOR>DES_7XX_04</DESIGNATOR>
  <IR_LEN>8</IR_LEN>
  <ASF_FILE></ASF_FILE>
  <REG_FILES>
  </REG_FILES>
  <MEMORY_MAP>
    <MEMORY_MODE>DEFAULT</MEMORY_MODE>
  </MEMORY_MAP>
</DEVICE>
<DEVICE>
  <NAME>PPC750FX</NAME>
  <DESCRIPTION>IBM Power PC 750FX Processor</DESCRIPTION>
  <TYPE>MICROPROCESSOR</TYPE>
  <TARGET>PPC750FX</TARGET>
  <SELREG_FILE></SELREG_FILE>
  <DESIGNATOR>DES_7XX_05</DESIGNATOR>
  <IR_LEN>8</IR_LEN>
  <ASF_FILE></ASF_FILE>
  <REG_FILES>
  </REG_FILES>
  <MEMORY_MAP>
    <MEMORY_MODE>DEFAULT</MEMORY_MODE>
  </MEMORY_MAP>
</DEVICE>
</DEVICE_TABLE>
```

2. Edit your board file to set <TABLE_TIED_RESET> to ON.
3. Right-click on your board file and select **Save**.
4. In the Workbench toolbar, select **Window > Show View > CF Options**.
5. Under the **Command Name** heading in the **CF Options** view, scroll down to **HRESET**.
6. Double-click on the value under the **Current Setting** heading to bring up a list of parameters. Set the option to **ENABLE** or **DISABLE**, depending on your application (see [CF HRESET](#), p.133.)
7. Under the **Command Name** heading in the **CF Options** view, scroll down to **CMDRST**.

8. Double-click on the value under the **Current Setting** heading to bring up a list of parameters. Set the option to **ENABLE** or **DISABLE**, depending on your application (see [CF CMDRST](#), p. 133.)
9. Repeat Steps 5 through 8 for each core you are debugging.

CF HRESET

If **HRESET** is set to **ENABLE**, then every time an **IN** command is issued the **HRESET** line will be pulled. Because the **HRESET** line is a physical line that is accessed through a single OCD connector on your target, all of the devices on your scan chain will be reset when an **IN** command is issued.

In the example described previously, in which there is a target with two processors (A and B), consider the case where Processor A is running correctly and you want to initialize Processor B. If the **CF HRESET** option is set to **ENABLE** and you issue an **IN** command in the view of Wind River Workbench pertaining to Processor B, both processors are reset, even though Processor A was already running correctly, because the **HRESET** line on your target runs through both processors.

If you set the **CF HRESET** option to **DISABLE**, the **HRESET** line will not be pulled when an **IN** command is issued. In the case described above, where Processor A is running correctly, if the **CF HRESET** option is disabled and you issue an **IN** command on Processor B, the **HRESET** line will not be pulled, and only Processor B is reset. Processor A remains running correctly without interruption.

CF CMDRST

This option controls the **HRESET** line into the target, and it affects the **RST** command. The **RST** command simply issues an **IN** command to all devices in your scan chain simultaneously. If this option is set to **ENABLE** then the **HRESET** line is pulled when an **RST** command is issued. If you **DISABLE** this option, then all of the devices on the scan chain are synchronized, and the **HRESET** line will not be pulled. With the **RST** command, all of the devices are reset anyway, so choosing to enable or disable the **CMDRST** depends on the hardware on your target.

Be aware of these two CF options, and make sure you have set them correctly for your system before you begin debugging. Failure to do so could cause you to accidentally reset a device that was already running correctly.



NOTE: Make sure you set the CF options correctly in each instance of Wind River Workbench that you have running. In the example used above, if you had set the options correctly for Processor A but had not set them in Processor B, working in the instance of Wind River Workbench pertaining to Processor B could cause you to reset a device unintentionally.

7.7 Commands for Multi-Core Debugging

When you are working with multiple devices in your scan chain simultaneously, any one device can be controlled and manipulated by going to the view of Wind River Workbench pertaining to that device and working with it as you would a single core target.

There are a few commands available that allow you to control all of the devices on your scan chain simultaneously. For a guide to all low-level commands, see the *Wind River Workbench On-Chip Debugging Command Reference*.

RST and RSTINN

The **RST** command is a way to issue an **IN** command to every device on your scan chain simultaneously. It behaves in an identical fashion to the **IN** command, in that it attempts to initialize Background Mode communications for each device and it transfers the chip-select table and any stored register settings to your target via the OCD link.

Similarly, the **RSTINN** command is a way to issue an **INN** command to every device on your scan chain simultaneously. As with the **INN** command, the **RSTINN** command merely places every device on your scan chain into background mode without affecting your target's register or chip select settings.

When you issue an **RST** or **RSTINN** command in one view of Wind River Workbench, a message appears in the other views that a synchronous reset is in progress.

GO ALL and HALT ALL

These are two additional commands that can be used to affect all of the devices on your scan chain simultaneously. To use these commands, first download code to your target in each of the perspectives of Wind River Workbench. The demo

programs provided may be used for this purpose; for information on downloading code to your target please see the *Wind River Workbench User's Guide*.

Once code is successfully downloaded to each device, the **GO ALL** command can be issued from any of the instances of Wind River Workbench that you have open. All of the code is started running on each of your devices simultaneously. Similarly, the **HALT ALL** command stops the code running on all your devices at once.



NOTE: If you start all the devices running simultaneously, and then use the standard **HA** command to stop one of the devices, the other devices will continue to run.

8

Configuring Target Registers

- 8.1 [Introduction](#) 137
- 8.2 [Downloading a Register File](#) 138
- 8.3 [Saving Register Settings from a Target](#) 139
- 8.4 [Enabling and Disabling Register Groups](#) 141
- 8.5 [Configuring Registers Manually](#) 143
- 8.6 [Working With Custom Register Groups](#) 147
- 8.7 [System Configuration \(SC\) Commands](#) 155

8.1 Introduction

Regardless of how you plan to initialize and configure your system, you must program and configure the internal registers on your target at least to the point where you are able to download any boot and application code.

This is done in two steps: first, configure register settings in the emulator's non-volatile RAM (NVRAM); second, copy the register settings from the emulator to the target.

Your emulator includes an area of NVRAM where you can store register settings for a target. Once you store register settings in NVRAM, you can load the register settings to and from the target.

Once the register values are present in NVRAM, the emulator automatically loads them to the target after each cold start, warm start, or target IN initialization command. To select which register values are written to the target, enable or disable the appropriate register groups.

Wind River emulators use low-level commands to configure register settings. These low-level commands are stored in a script called a *register file*, a text file with the extension ***.reg**.

If your target already has register values set and configured, you can upload these values to the emulator NVRAM. From there, you can save the register settings to a register file and store it on your host computer for use on other targets.

You can work with registers for your target using a Wind River Probe or a Wind River ICE SX with Wind River Workbench, or by using low-level commands in the **OCD Command Shell** in Workbench.

8.2 Downloading a Register File

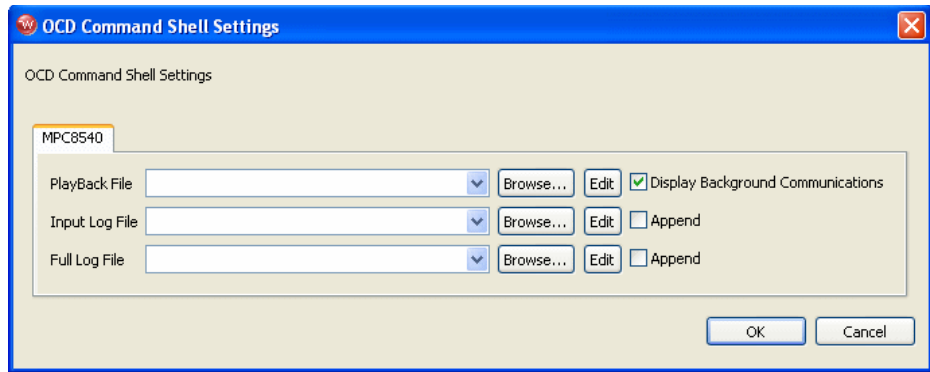
Wind River supplies register files for Wind River evaluation boards, as well as for many third-party target boards.

All Wind River hardware reference designs are shipped with a register file that you can use to initialize the target registers, so that code can be downloaded to the board. To use a register file to initialize your target registers, you must first download the file to the emulator NVRAM, then from the NVRAM to the target.

To download a register file, use the following steps.

1. In the Workbench toolbar, select **Window > Show View > OCD Command Shell**.
2. In the **OCD Command Shell**, click **Playback File**.

The **OCD Command Shell Settings** dialog appears.



3. Next to the **PlayBack File** field, click **Browse**.
4. Navigate to the register file you wish to use and click **Open**.
Register files for Wind River hardware reference designs are located in *installDir/workbench-2.x/dfw/build/host/registers*.
5. Click **OK**.
6. In the **OCD Command Shell**, click the **Playback File** icon again.
Workbench downloads the register values from the register file you selected to your emulator NVRAM.
7. In the **OCD Command Shell**, enter the command **IN**.
The emulator initializes the target and copies the register values from its NVRAM down to the target.

8.3 Saving Register Settings from a Target

If you are working with a target that already has its registers correctly initialized, you can upload those register settings to a file and save them on your host computer.

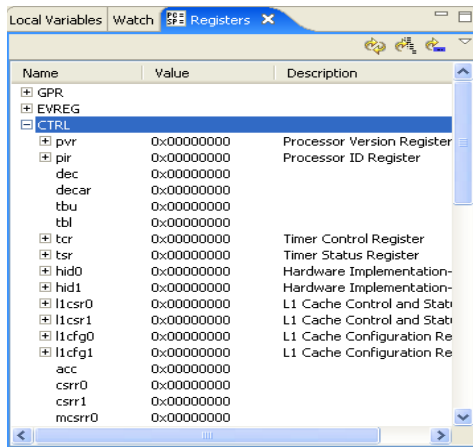
It is useful to save register settings to a file if you have received a target with registers already initialized, or if you have manually programmed registers on

your target one by one and wish to use those settings again on another target. After you save the settings to a register file, you can download the settings to a new target.

To save the register settings from a target, complete the following steps:

1. Select **Window > Show View > Registers**.

The **Registers** view appears.



2. Right-click in the **Registers** view and select **Save Target Register Values to File**.

The **Save Target Registers** dialog appears.

3. Click **Save**.

Use the browser window that appears to specify a file. Workbench saves the file in the location you specify with the extension ***.reg**.

Saving Register Settings Using Low-level Commands

Alternatively, you can save the register values using low-level commands in the **OCD Command Shell**.

First, copy the register settings to the emulator configuration file from the target using the low level command **SCT COPY**.

➔ **NOTE:** Only registers in enabled register groups are copied up from the target during an **SCT COPY** operation.

Next, copy the register settings from the emulator to a register file.

➔ **NOTE:** Although you could just copy the register settings to the emulator configuration file and not create a register file, Wind River recommends that you create one anyway. If you change targets, or if the emulator configuration file were to get corrupted or overwritten, a register file is the easiest way to fix it.

Upload register settings from the emulator to a file using the **PJ UPLOAD** command as described in the *Wind River Workbench On-Chip Debugging Command Reference*.

8

8.4 Enabling and Disabling Register Groups

Workbench stores registers in logical register groups. You can enable or disable any register group using the **Registers** view.

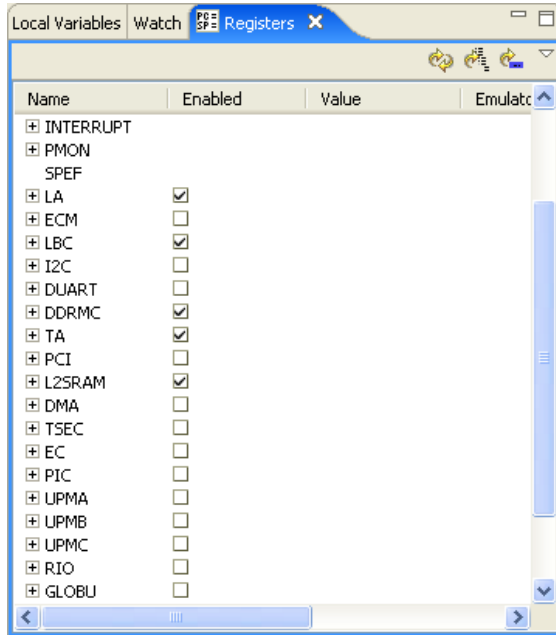
When you initialize your target using the **IN** command, the register values that are stored by the emulator are copied down to the target. However, the emulator only copies the register settings for the register groups that are enabled. Register groups that are disabled on your target do not have register data transferred. Disabling a register group enables you to view the target register value, but prevents it from being overwritten during target initialization.

➔ **NOTE:** If you change a register value directly on the target of a register group that is disabled, that register does not get overwritten by the emulator during an initialization. Note, however, that the processor may still reset that register value to the processor default during a target initialization.

The following steps describe how to enable or disable a register group on your target.

1. In the Workbench toolbar, select **Window > Show View > Registers**.
2. Right-click in the **Registers** view and select **Show Emulator Settings**.

The **Registers** view now shows a check box next to each register group, under the heading **Enabled**.



3. Select or clear the box next to any register group to enable or disable that group.

Enabling and Disabling Register Groups with Low-Level Commands

You can also enable and disable register groups with the command **CF GRP** in the **OCD Command Shell** in Workbench.

To use the **CF GRP** command, use the following steps.

1. In the Workbench toolbar, select **Window > Show View > OCD Command Shell**.
2. At the **>BKM>** prompt, type the command **CF GRP**.

The first register group appears, as shown below:

```
>BKM>cf grp
Group          (CF GRP (M/S)   Name = ENABLED/DISABLED
```

```
CUSTOM (0=Disable 1=Enable) Enabled >
```

The name of the register group is displayed, along with its current status (either **ENABLED** or **DISABLED**).

3. Type **0** to disable the group or **1** to enable it.
4. To leave the setting as it is and advance to the next register group, press the **ENTER** key without typing **0** or **1**.
5. Continue through the list of register groups enabling and disabling them as required.
6. When you have enabled or disabled all groups, type **CF UPLOAD GROUP** at the **>BKM>** prompt.

This displays a list of all of the register groups on your target with their current settings.

```
>BKM>cf upload group
CF GRP          GT64260_CPU ENABLED      ; GROUP
CF GRP          GT64260_SDRAM ENABLED    ; GROUP
CF GRP          GT64260_DEVICE ENABLED    ; GROUP
CF GRP          GT64260_GPP ENABLED       ; GROUP
CF GRP          GT64260_MPP ENABLED       ; GROUP

>BKM>
```

8.5 Configuring Registers Manually

If you are using a target for which Wind River does not supply a register file, you may have to modify an existing one (see [Modifying an Existing Register File](#), p.154); or, if your target has default register settings, you may modify them manually.

Remember that the register file sets the register values in the emulator NVRAM, not on the target. The emulator copies the values you set in its NVRAM down to the target when you initialize the target with an **IN** command. Without a register file, the NVRAM contains default register values, typically made for a Wind River evaluation board, which most likely are not suitable for your target. So the **IN** command will not set the target registers properly.

Some target processors, for instance most PowerPC targets, come with default register settings. If your target has default register settings, you can modify the

registers directly on your on your target manually, at least to the point where you can download your boot ROM application code.

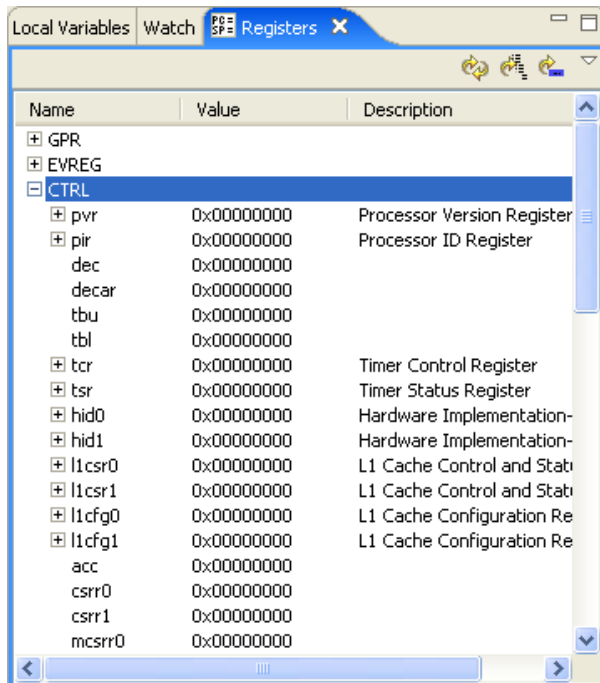
Remember that if you modify your registers manually, an IN command or target reset will overwrite your changes.

To modify registers manually, use the **Registers** view in Workbench. The **Registers** view lets you view the bit-level detail for each register. The following sections describe the **Registers** view and the bit-level detail provided.

The Registers View

When the **Registers** view is open in Workbench, all of the register groups for your target are displayed with + signs beside them. Clicking on a + sign expands the register group, showing all of the registers that are included in that register group along with the value that they are currently set to. An example of an expanded register group is shown in [Figure 8-1](#).

Figure 8-1 Expanded Register Group



➔ **NOTE:** [Figure 8-1](#) is only an example of an expanded register group. The groups and the register values vary widely depending on your target architecture.

Bit-Level Detail

You can view the bit-level detail for any register by clicking on the + sign beside the register in the register group.

➔ **NOTE:** Before you can make any changes to your register settings, you need to enable the register group that contains the register you want to modify, so that the values download to the target when you initialize your system. If you do not enable the register group, you can still modify the settings in the emulator but not on the target. For more information, see [8.4 Enabling and Disabling Register Groups](#), p. 141.

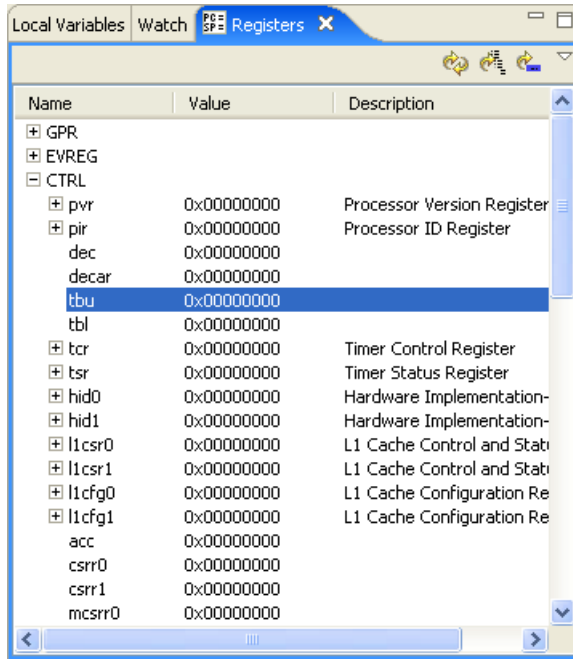
You can make changes to any of the register settings by modifying each of the bit-level settings for any register.

To modify bit-level values for your target, complete the following steps:

1. In the **Registers** view, double-click on the name of the register you wish to edit.

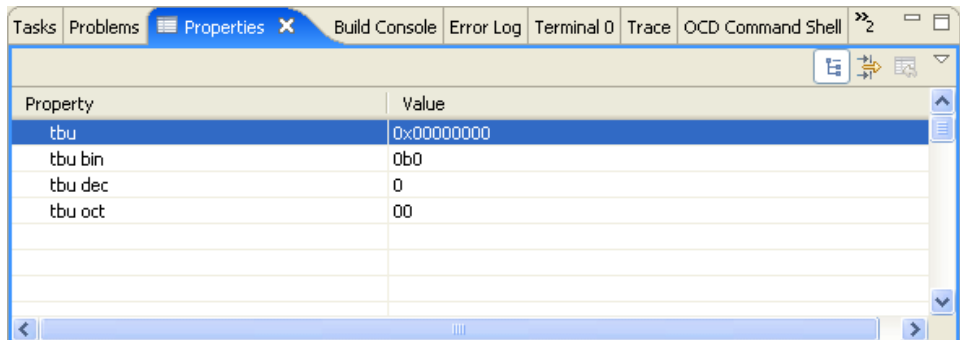
[Figure 8-2](#) shows the **Registers** view with the TBU register selected.

Figure 8-2 **Selecting a Register**



This opens the **Properties** view, which shows the name of the register you have selected under the **Property** heading and its current setting under the **Value** heading, as shown in [Figure 8-3](#).

Figure 8-3 **Properties View**



2. Select the value under the **Value** heading and edit it as necessary.
3. In the **Registers** view, click the **Refresh Values** button. The register information reappears with your changes.



NOTE: Some registers are write-protected and cannot be edited.

8.6 Working With Custom Register Groups

8

You can create custom register groups to perform various tasks, such as initializing peripheral memory.

This section presents two scenarios to describe how to add custom registers or register groups to your target, depending on what you want to do:

- Creating a new set of registers for your target
- Modifying an existing register file

Creating a New Set of Registers

To create new registers using the **Registers** view, complete the following steps:

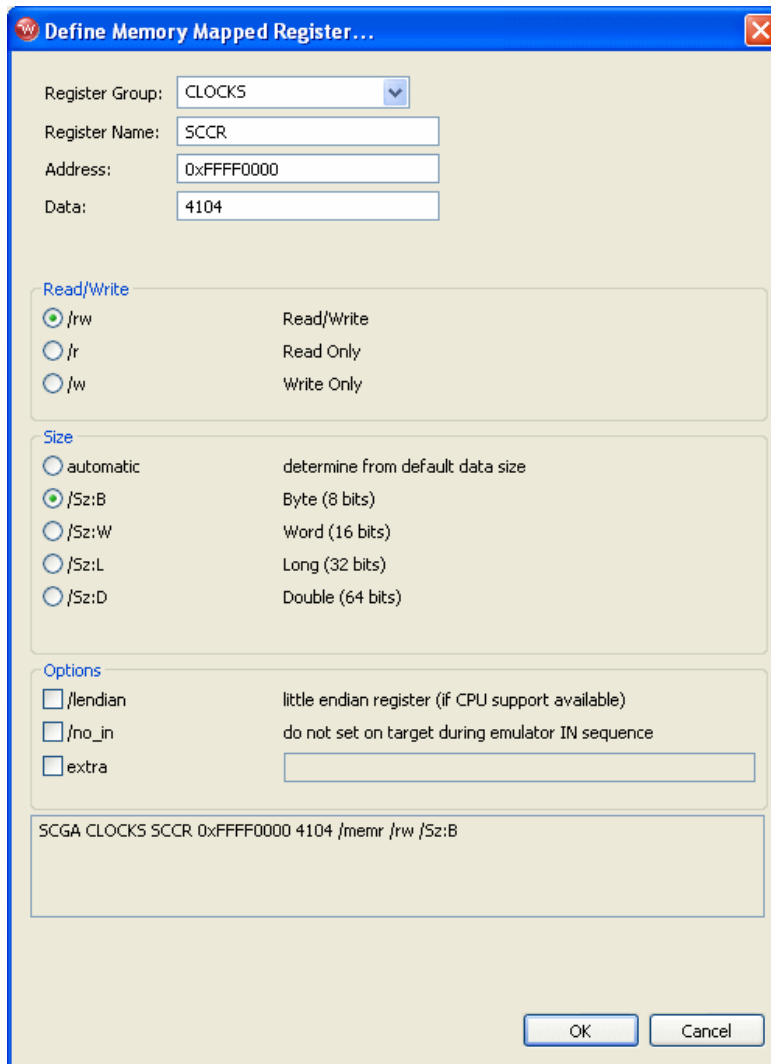
1. Right-click in the **Registers** view and select **Define Memory Mapped Register...**

The **Define Register** dialog appears, as shown in [Figure 8-4](#).



NOTE: Only memory mapped registers can be created with this dialog. To create a non-memory mapped register, see [Creating New Registers With Low-Level Commands](#), p.149.

Figure 8-4 Define Register Dialog



2. In the dialog text fields, specify the details of your new register as follows:
 - In the **Register Group** field, specify the register group to which the new register will be added.
 - In the **Register Name** field, specify the name of the new register.

- In the **Address** field, specify the address at which the new register will be located.
- In the **Data** field, specify the data to be stored in the new register.

The example in [Figure 8-4](#) shows a new register called **SCCR** that will be placed in the register group **CLOCKS**, located at address **0xFFFF0000**, and has the value **4104**.

3. Set the following options:
 - Set the register to be read/write, read-only, or write-only.
 - Set the register to store data as **Byte, Word, Long, or Double**.
 - Set the register as little-endian. (This only applies if your target CPU is able to switch between big-endian and little-endian modes.)
 - Specify whether the register will be set on the target during an initialization sequence.

The example in [Figure 8-4](#) shows the register set to read/write and size B (byte).

There are advanced options not available from the **Define Register** dialog; these options must be set using low-level commands in the **OCD Command Shell**. For information about these options, see [SCGA Options](#), p. 151.

4. Click **OK** to create the register.

Creating New Registers With Low-Level Commands

To create a new set of custom registers for your target, create an ASCII text file with the extension ***.reg**. Then load that file to your emulator and the target, thus setting up register groups as required for your system.

Prior to creating your own register file, look at one of the default register groups that was included with your Wind River Workbench software to learn more about how register files are laid out. The following is an example of a register file:

```
SC GRP ERASE

SCGA GEN   SCR           FFB00000  4006A300  /r
SCGA GEN   SSR           FFB00004  0A80042F
SCGA GEN   PLLCR        FFB00008  06000000
SCGA GEN   SOFTSR       FFB0000C  00000000  /no_in

SCGA MPC107_PCI  VENDOR  00000000  1057 /r(nwf) /w(nwf) /r /ua:1 /lendian
SCGA MPC107_PCI  ADDR_00 00000000  80000000 /wo /hide /w /ua:0 /lendian
```

```
SCGA MPC107_PCI ID 00000002 0004 /r(nwf) /w(nwf) /r /ua:1 /lendian
SCGA MPC107_PCI ADDR_02 00000000 80000002 /wo /hide /w /ua:0 /lendian
SCGA MPC107_PCI PCICMD 00000000 0004 /r(nwf) /w(nwf) /ua:1 /lendian
SCGA MPC107_PCI ADDR_04 00000000 80000004 /wo /hide /w /ua:0 /lendian
```

The register file you create for your target depends entirely on the information that you need to set up for your system, so your file may look similar to the default register groups or it may look completely different. The list below describes some of the items that you must include in your file.

- **Include the line SC GRP ERASE**

Include the line **SC GRP ERASE** in your register file to erase any existing register groups and settings that might be set up by default on your target or in the emulator file. Since you are creating an entirely new register set for your target, you need to make sure that any residual information on the target or in the emulator is removed prior to adding new information.

- **Include any configuration options**

Use **CF** commands to include any configuration options required for your target. Always include a **CF TAR** *target_processor* configuration line in this section to make sure that your register file matches the target that you are working with. *target_processor* must match the processor you selected when you made your connection to the emulator.

Including this line also makes it clear to anyone else who plans to use your register file which target your file is intended for. Include this line, and any other configuration options that are required for your target, in your register file. For more information on the available configuration options for your architecture, and for syntax information, see the *Wind River Workbench for On-Chip Debugging Configuration Options Reference*.

- **Set up register groups for your target**

New registers and register groups are created using **SCGA** commands. The syntax for adding a new register in a register group is:

```
SCGA GroupName RegisterName Address Data Options
```

This command adds the register *RegisterName* to the register group *GroupName*.

GroupName — This is the name of the existing register group that the new register is added to.

RegisterName — This is the name of the register that you are creating.

Address — This is the address where the new register is located.

Data — This is the data that is stored in the register you are creating.

Options — There are many options associated with the **SCGA** command. These are described in [SCGA Options](#), p.151.

Continue adding all of the new registers for your target. When you finish adding the required registers, save the file with the extension*.reg.

You are now finished creating a register file for your target. You can download the file to the emulator and your target as described in [8.2 Downloading a Register File](#), p.138. When you have downloaded the file and issued an **IN** command, all of the new register settings become visible in the **Registers** view in Workbench.

SCGA Options

/cpur

This option specifies that the register you are creating is a CPU core register (that is, SPR, or other non-memory-mapped register.)

Example:

```
SCGA SIM_MMU SIM_IBATOL 4014 00000004 /cpur
```

This example creates a new register group called **SIM_MMU**, with a core register called **SIM_IBATOL** embedded in it.



NOTE: Newly created groups are disabled by default. To enable the register group, see [8.4 Enabling and Disabling Register Groups](#), p.141.

/hide

This option specifies that the register will not be visible when an **SC** or **DR** command is issued. It will only be visible when an **SC UPLOAD** or **SCG UPLOAD** command is issued.

/lendian

This option specifies that the register you are creating is little-endian. This option only applies if your target CPU can switch between little-endian and big-endian modes.

/memr

This option specifies that the register you are creating is a memory-mapped register. This is the default for all self-defined registers and all registers created in the **Define Register** dialog.

/no_in

This option specifies that the register you are creating will not be set on the target during an **IN** initialization sequence.

/r, /w, /rw

These options are read-only, write-only, and read/write flags. Registers are set to **/rw** by default.

/Sz:B, /Sz:W, /Sz:L, /Sz:D

These options force the size of the register to either Byte (8 bits), Word (16 bits), Long (32 bits), or Double (64 bits). The default register size is determined by the amount of characters used to specify the default value. (In the **Define Register** dialog this is the value you enter in the **Data** field.)

/va_dr

This option is used on anchor registers to make them available on a **DR** command. (For PowerPC, on an **IMMR** command.)

/wo

This option defines a fixed-value register. A register created with this option will not be affected by an **SCT COPY** command.

/w(nwf)

This option specifies a write cycle (next write first.) It indicates that in order to write a value to this register, you first need to write the following register value to the target.

Example:

```
SCGA MPC_PCI PCICMD 80000CFC 0600 /w(nwf) /r(nwf)
SCGA MPC_PCI ADDR_04 80000CF8 04000080 /wo /hide
```

In this example, you create a register called **PCICMD** in the register group **MPC_PCI**. The option specifies that the register **PCICMD** cannot be written to

unless a write to the register `ADDR_04`, in the same register group, is performed first.

/r(nwf)

Similar to the ***/w(nwf)*** option, this option specifies a read cycle (next write first.) It indicates that in order to read a register, you first need to write the following register value to the target.

/w(nwa)

This option specifies a write cycle (next write after.) It indicates that if you write a value to this register, you need to write the next register value to the target afterwards.

For more information on the **SCGA** and **SCGD** commands and their options, see the *Wind River Workbench for On-Chip Debugging Command Reference*.

Using Your New Register File

The register file you created is designed to erase all of your register groups and recreate them every time you play back the file. Since all of the settings are stored in the emulator file, it is likely that you will not have to play back the register file very frequently, and therefore having the file configured to erase all of the settings each time is acceptable in most cases.

If you prefer that your register settings are not erased and recreated each time you play back the file, complete the following steps:

1. Create your register file as described in the steps above using **SCGA** commands.
2. Play back that file once, to make sure all of the required register groups are created in the emulator and on your target.
3. Next, remove the **SC GRP ERASE** line from your file, which prevents all of the groups from being erased each time the file is loaded.
4. Change all of the **SCGA** commands to **SC** commands.

For example, a line such as:

```
SCGA SYSMGR SYSCFG 03FF0000 87FFFA0
```

would be changed to:

```
SC SYSCFG 87FFFA0
```

Note that the syntax for the **SC** commands is different than for the **SCGA** commands. See the *Wind River Workbench for On-Chip Debugging Command Reference* for more information. The syntax is different because **SCGA** commands are used to create new registers, whereas **SC** commands are used to change the values of existing registers. Using **SCGA** commands for registers that already exist results in a syntax error. Using **SC** commands for registers that do not exist also results in a syntax error.

After you replace all of the **SCGA** commands with **SC** commands, playing back the register file only updates the registers that already exist on your target and does not erase anything.

Modifying an Existing Register File

If you have a register file that you are satisfied with, but you want to add some additional groups of registers to it, it is easiest to modify your existing register file rather than create an entirely new one.

You can either add registers to an existing group or add a new group of registers to an existing file.

Adding Registers to an Existing Register Group

To add registers to an existing register group, open the register file you want to modify in a text editor and erase the existing group using the **SC GRP ERASE** *GroupName* command. Then add back all of the registers that were previously included in that group, as well as any new ones you want to add, using the syntax shown below:

```
SCGA GroupName RegisterName Address Data Options . . .
```

GroupName — The name of the existing register group that the new register is added to.

RegisterName — The name of the register that you are creating.

Address — The address where the new register is located.

Data — The data that is stored in the register you are creating.

Options — There are many options associated with the **SCGA** command. A full description of all of the available options is available in the *Wind River Workbench for On-Chip Debugging Command Reference*.

Adding a New Group of Registers to an Existing File

To add a new group of registers to an existing file, open the register file you want to modify in a text editor and include the command **SC GRP ERASE** *GroupName*. Then add each of the registers to be included in the group using the following syntax:

SCGA GroupName RegisterName Address Data Options...

GroupName — The name of the existing register group that the new register is added to.

RegisterName — The name of the register that you are creating.

Address — The address where the new register is located.

Data — The data that is stored in the register you are creating.

Options — There are many options associated with the **SCGA** command. A full description of all of the available options is available in the *Wind River Workbench On-Chip Debugging Command Reference*.

As described in [Creating a New Set of Registers](#), p.147, you can change your register file after you play it back the first time so that the register groups that you have added or made changes to are not erased each time you play back the register file. To do this, first make sure that you play back the register file once to create the register groups in the emulator and on the target. Then open your register file and remove the **SC GRP ERASE** *GroupName* line from the file and everywhere that you have included **SCGA** commands, replace them with **SC** commands using the syntax described in the *Wind River Workbench for On-Chip Debugging Command Reference*. Doing this updates the registers every time you play back the register file instead of deleting and recreating them.

8.7 System Configuration (SC) Commands

The **SC** commands allow you to edit any of the internal peripheral registers for your target processor. They allow you to modify non-volatile values and store them in your host computer. The values are loaded into the emulator file and downloaded to your target any time it is initialized. The **SC** commands also let you view and edit any of the current target values.

For information on all of the available SC commands, see the *Wind River Workbench for On-Chip Debugging Command Reference*.

9

Programming Flash Memory

- 9.1 Introduction 157
- 9.2 Connecting to a Target 158
- 9.3 Testing Flash Workspace 163
- 9.4 Configuring Registers 164
- 9.5 Using the Flash Programmer View 165
- 9.6 Flash Configuration Tab 166
- 9.7 Flash Programming Tab 168
- 9.8 Flash Memory/Diagnostics Tab 174

9.1 Introduction

The **Flash Programmer** view provides the ability to flash images into flash chips present on your target board.

To program flash correctly you need to know the physical characteristics of your flash bank. For instance, your board may have one flash device connected to a 64-bit bus. Or it may have a bank of several flash devices, for example two flash devices, each wired at 16 bits, connected along a 32-bit bus.

If you are using a Wind River-supported board, this information can be found in the file *installDir/vxworks-6.x/target/config/yourTargetBoard/target.ref*.

If you are not using a Wind River-supported target, consult your target's documentation. The design primitives of your target board should be included in its board specification and schematics.

To program target flash, you must create an active target connection and configure your target registers.

9.2 Connecting to a Target

This tutorial uses a Wind River Probe emulator connected to a Wind River PPMC750FX target.

To connect to your target, use the following steps:

1. Launch Wind River Workbench according to the method for your host.

Linux/Solaris Hosts

From your installation directory, issue the command

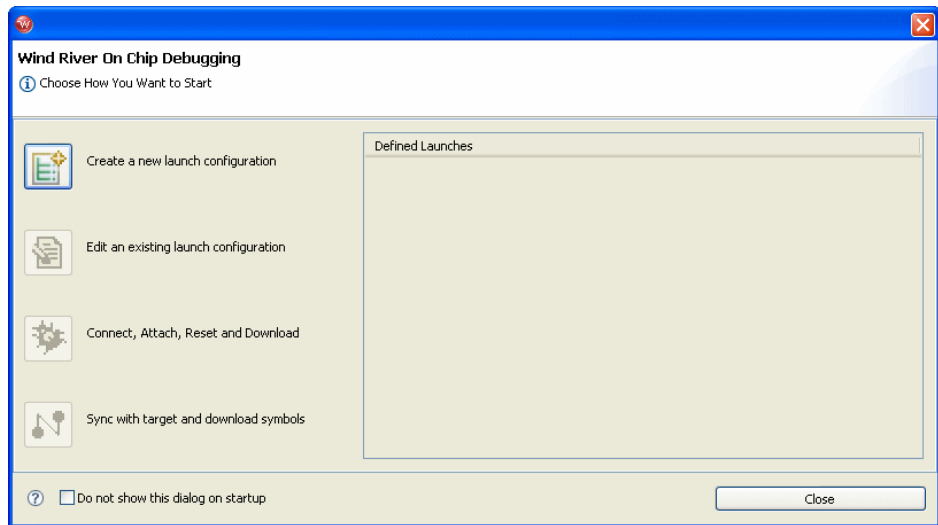
```
$ ./startWorkbench.sh
```

Windows Hosts

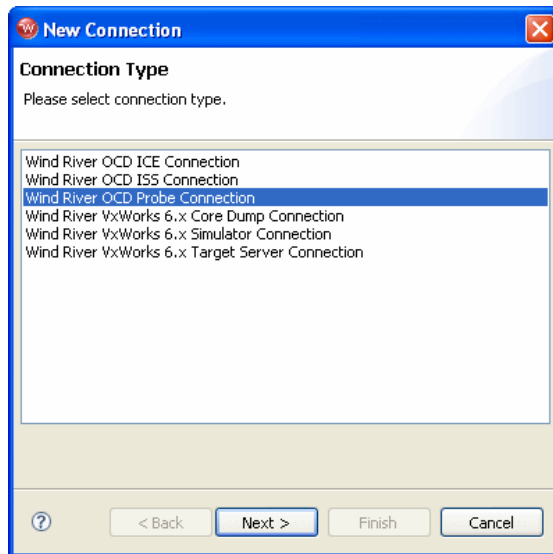
Select **Start > All Programs > Wind River > Wind River Workbench** *version*.

On Windows hosts, Workbench prompts you to specify a workspace location. Linux/Solaris hosts use the default location *installDir/workspace*.

When Workbench opens, the **Quick Target Launch** dialog appears.

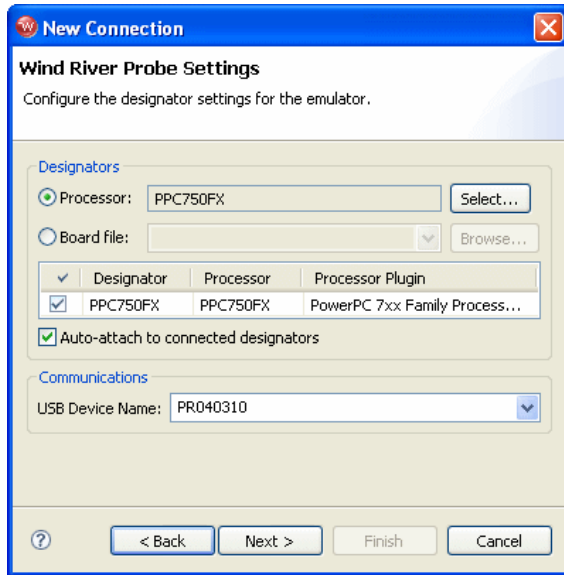


2. Select **Create a new launch Configuration**.
The **Connection Type** dialog appears.

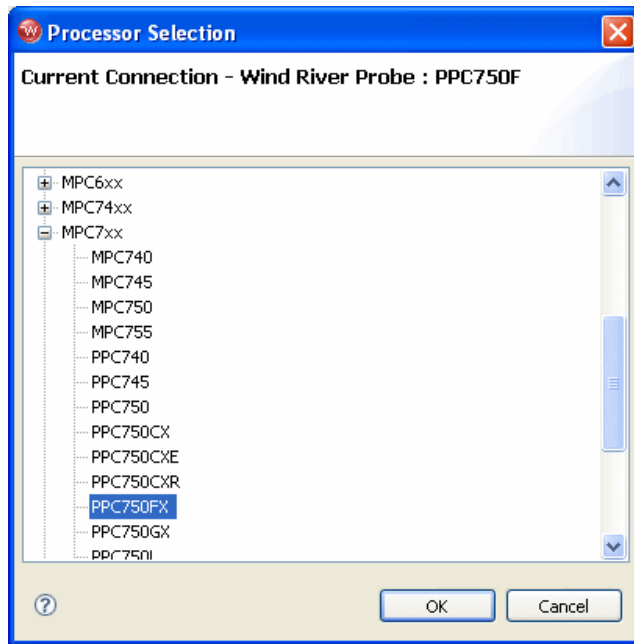


3. Select **Wind River OCD Probe Connection** and click **Next**.

The **Processor Selection** dialog appears.



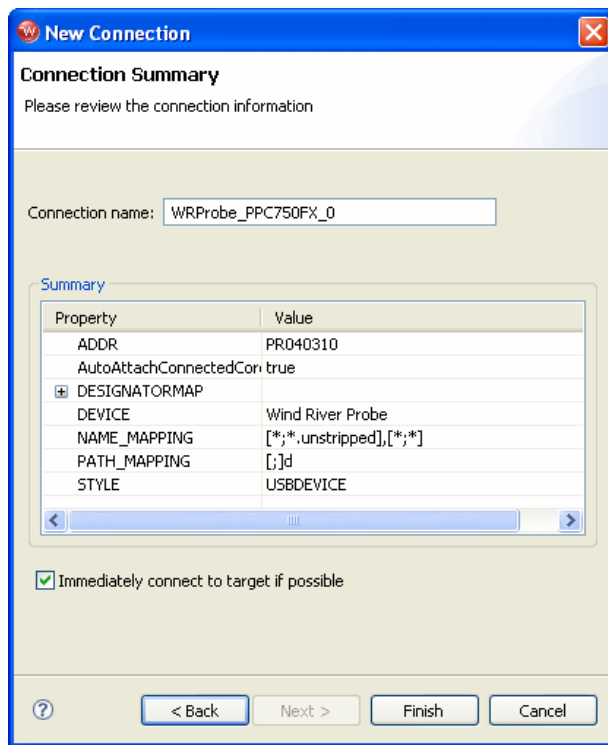
4. Click **Select**. From the list that appears, expand **MPC7xx** and select **PPC750FX**.



5. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

6. Click **Next**.
7. The connection wizard passes through a number of screens that you do not need to configure for this tutorial. Leave all settings at their defaults and click **Next** until you come to the **Connection Summary**.



8. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

Workbench creates a target connection called **WRProbe_PPC750FX** in the **Target Manager** view and opens the **Reset and Download** view.

You do not need to download code to program flash memory, so click **Close** to close the **Reset and Download** view.

9.3 Testing Flash Workspace

The flash programming algorithm needs to run on the target. This requires a RAM workspace, to which the algorithm will download, and breakpoints, which are used to stop an erase and program operation at completion.

Reading and Writing Memory

Once you have established communications with the target, use the following procedure to make sure you can write to and read from the target. In this example we assume that the RAM workspace is 0x00F00200.



NOTE: A RAM workspace address of 0x00F00200 is not appropriate for all targets. For Wind River-supported targets, you can find the necessary RAM workspace in your target's **target.ref** file, located in *installDir/vxworks-6.x/target/config/yourTarget/target.ref*.

Wherever the RAM workspace is located on your target, you must make sure that memory is writable there.

At the **>BKM>** prompt, enter **dm 00F00200** and press **ENTER**. Doing so displays the memory on your target at address 0.

Next, enter **sm 00F00200 1234** and press **ENTER** to set the memory at address 0 to the value 1234. Enter **dm 00F00200** to display the memory at that address again.

If you are communicating properly with your target, output is similar to that shown below:

```
>BKM>dm 00f00200
00F00200: FF7C EFFE FEFF E3FE 0D01 0FBE F0FD BFB6      .|.....
>BKM>sm 00f00200 1234
>BKM>dm 00f00200
00F00200: 1234 EFFE FEFF E3FE 0D01 0FBE F0FD BFB6 .4.....
>BKM>
```

Occasionally, you may have difficulty programming flash memory on your target if software breakpoints are not being hit properly. Test this functionality before you continue.

To use the test, enter the following commands at the **>BKM>** prompt in the **OCD Command Shell**:

```
>BKM>df e 0
>BKM>di 0 6
```

```
$00000000 : 0x60000000 :ppc nop
$00000004 : 0x60000000 :ppc nop
$00000008 : 0x60000000 :ppc nop
$0000000C : 0x60000000 :ppc nop
$00000010 : 0x7C0004AC :ppc sync
$00000014 : 0x4BFFFFFF0 :ppc b          0x4
>BKM>go 0
>RUN>dr pc
PC = 00000004
>RUN>dr pc
PC = 00000010
>RUN>sb 8

>RUN>

!BREAK! - [msg12000] Software breakpoint; PC = 0x00000008 [EVENT Taken]
>BKM>
>BKM>rb

>BKM>
```

9.4 Configuring Registers

Before you can program target flash, you must configure your target registers. This is done in two steps: first, configure register settings in the emulator's non-volatile RAM (NVRAM); second, copy the register settings from the emulator to the target by issuing an **IN** initialization command.

Wind River emulators use low-level **SCGA** commands to configure register settings. These low-level commands are stored in a script called a *register file*, a text file with the extension ***.reg**. Register files for Wind River hardware reference designs are located in *installDir/workbench-2.x/dfw/build/host/registers*.

To configure target registers for a Wind River PPMC750FX, use the following steps:

1. In the Workbench toolbar, select **Window > Show View > OCD Command Shell**.
2. In the **OCD Command Shell**, select **Settings**.
The **OCD Command Shell Settings** dialog appears.
3. Next to the **PlayBack File** field, click **Browse**.
4. Navigate to the file you wish to use and click **Open**.

The register file for the Wind River PPMC750FX is **ppmc750fx.reg**, located in the folder **WindRiver_PPMC** in the directory *installDir/workbench-2.x/dfw/build/host/registers/PowerPC/7xx*.

5. Click **OK**.

You are returned to the **OCD Command Shell**.

6. Click **Playback File**.

Workbench plays the register file and configures the emulator NVRAM.

7. At the **>BKM>** prompt in the **OCD Command Shell**, enter the command **IN**.

Workbench initializes the target and configures the target registers with the values from the emulator NVRAM.

You have now configured the target registers. For more information on registers, see [8. Configuring Target Registers](#).

9.5 Using the Flash Programmer View

Once you have connected to Wind River Workbench, and configured your target registers, you are ready to begin programming flash.

In the Workbench toolbar, select **Window > Show View > Flash Programmer**.

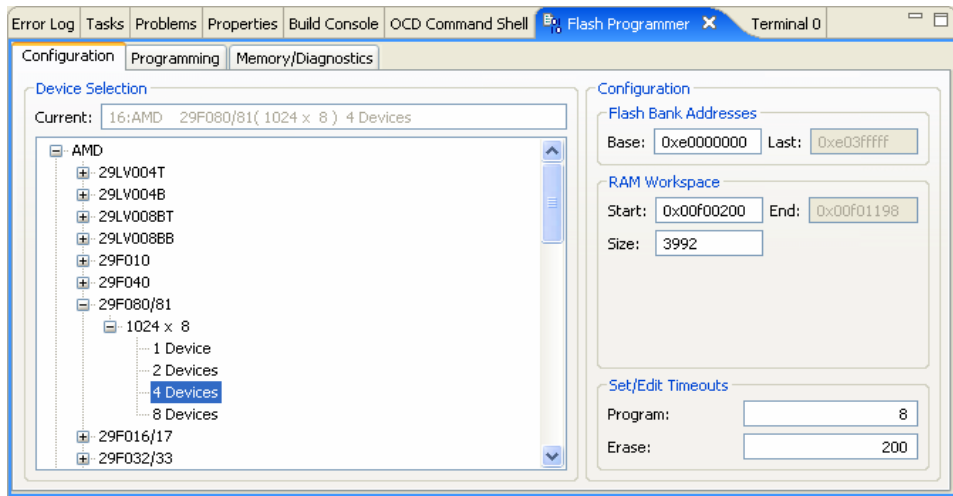
The **Flash Programmer** view appears.

The **Flash Programmer** view has three tabs: **Configuration**, **Programming**, and **Memory/Diagnostics**. Use these tabs to configure your flash address and RAM workspace, choose files for download, execute erase and program operations, and check the results of your operations.

9.6 Flash Configuration Tab

Use the **Configuration** tab to configure the base address and workspace address for flash memory erase operations. You can also enter the physical description of your flash devices.

Figure 9-1 **Configuration Tab**



9.6.1 Selecting a Flash Driver

In the **Device Selection** field, browse to a description of your flash bank. [Figure 9-1](#) shows an example of a flash bank consisting of four 8-bit AMD 29F0808 devices.



NOTE: For AMD flash devices, “F” and “LV” devices are interchangeable in Workbench.

If you attempt to move on to the **Programming** tab without selecting a flash bank description in the **Configuration** tab, Workbench displays an **Invalid Flash Bank** error and returns you to the **Configuration** tab.

9.6.2 Configuring Flash Memory Bounds

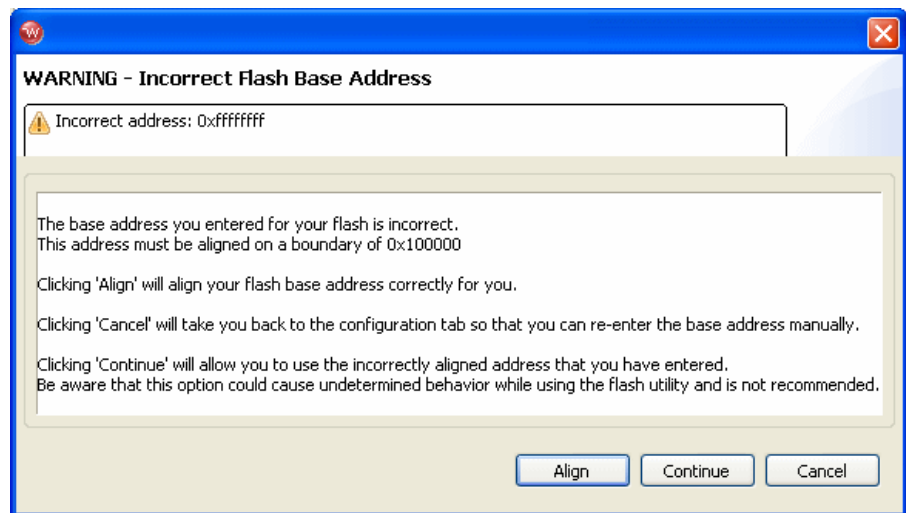
In the **Configuration** field, enter the **Base** value for the area of flash memory you wish to erase. In [Figure 9-1](#) the address used is 0xe0000000. The **Last** field populates automatically.



NOTE: Workbench erases flash memory sector by sector. That means that no matter where the address you enter in the **Base** field is located within the flash sector, Workbench will still erase the entire sector.

If Workbench detects that the address you entered in the **Base** field is not correctly aligned with the flash sector boundary, it displays the following warning message:

Figure 9-2 **Incorrect Flash Base Address**



- To have Workbench align your base address, click **Align**. Workbench aligns the base address with the nearest preceding sector boundary.
- To go back to the **Configuration** tab and re-enter the address manually, click **Cancel**.
- To use the base address as you entered it, without aligning it with the flash boundary, click **Continue**.



CAUTION: Choosing **Continue** may cause unpredictable results in your flash programming operations. Wind River recommends that you align the base address with the flash sector boundary.

9.6.3 Configuring RAM Workspace

The flash programming algorithm needs to run on the target. This requires a RAM workspace, to which the algorithm will download.

In the **RAM Workspace** field, enter the **Start** value for the area of RAM you wish to use as the workspace. In the **Size** field, enter the desired size of the workspace in bytes. In [Figure 9-1](#) the starting address used is 0x00F00200 and the workspace size is 3992. The **End** field populates automatically.



NOTE: A RAM workspace address of 0x00F00200 is not appropriate for all targets. For Wind River-supported targets, you can find the necessary RAM workspace in your processor's **target.ref** file, located in *installDir/vxworks-6.x/target/config/yourTargetBoard/target.ref*, or **target.ref.linux** file, located at <http://www.windriver.com/support>.

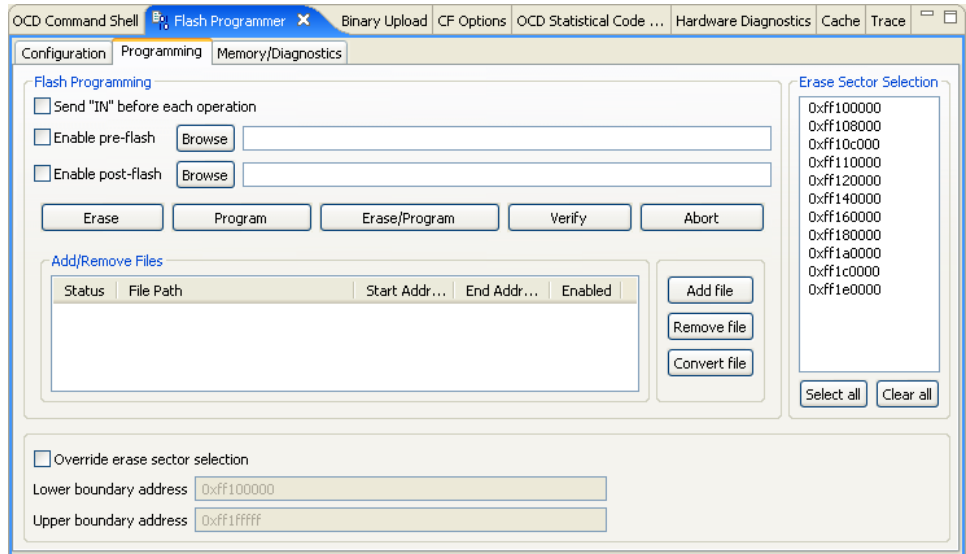
9.6.4 Setting Timeouts

To set a program or erase timeout, use the **Program** or **Erase** fields in the **Set/Edit Timeouts** area. Enter a timeout value in seconds. If you enter an invalid number, Workbench resets the timeout to its default setting.

9.7 Flash Programming Tab

Use the **Programming** tab to execute erase and program operations in flash and to specify files for download.

Figure 9-3 Programming Tab



9

9.7.1 Erasing and Programming Flash

To issue an IN initialization command before erase or program operations, select the **Send "IN" before each operation** checkbox.

Click **Erase** to erase the contents of the flash memory sectors you selected in the **Configuration** tab.

Click **Program** to program the flash memory with the files you selected in the **Add/Remove Files** area of the **Programming** tab.

Click **Erase/Program** to perform both operations. Workbench will erase all selected flash sectors before programming.

Click **Abort** to stop the erase or program operation.

9.7.2 Verifying Flash Contents

Click **Verify** to execute a byte-by-byte comparison between the file you just downloaded and the file already in memory. If there is a discrepancy, Workbench will break at that address and deliver an error message.

9.7.3 Running a Pre- or Post-Flash Script

You can specify a script to run before or after an erase or program operation. Select the **Enable pre-flash** or **Enable post-flash** checkboxes (you can select either or both for any operation). Next to the checkbox, click **Browse** and navigate to the script you wish to run.

9.7.4 Selecting Flash Sectors for Erasure

The **Sectors** field automatically populates with the starting addresses of sectors of flash memory, depending on which flash device you specified in the **Configuration** tab. Click on a sector to select it. You can select all sectors by clicking **Select All**. Click **Clear All** to deselect all sectors.

Before you erase all sectors, make sure you know what resides in the flash. For example, PowerPC 82xx processors read their reset configuration word from FE000000 out of the flash device, so for 82xx processors, erasing the entire device may cause problems with resetting the board.

9.7.5 Manually Configuring Flash Memory Erasure Bounds

Workbench allows greater user control by allowing manual configuration of the flash memory bounds for erase operations.

You can manually configure the flash memory bounds by checking the **Override erase sector selection** checkbox. When this box is checked, Workbench will allow you to enter any addresses in the **Lower boundary address** and **Upper boundary address** fields.



NOTE: If the values you enter result in a memory address range that is outside your target board's flash programming area, erase operations will not perform correctly.

9.7.6 Adding Files

To add a **.bin** file, click **Add File**. This opens the **Choose File for Flash Download** browser window. Workbench automatically looks for a folder labeled **firmware**, located in *installDir/workbench-2.x/dfw/version/host/firmware*, where *version* is the installed version of the debugger middleware. If your **.bin** files are stored in

another folder, use the browser to navigate to it. Select the file you want and click **Open**. The file will appear in the **File Path** field.

9.7.7 Removing Files

To remove a file from the list, highlight it and then click **Remove File**.

9.7.8 Converting Files To Wind River Flash Binary Format

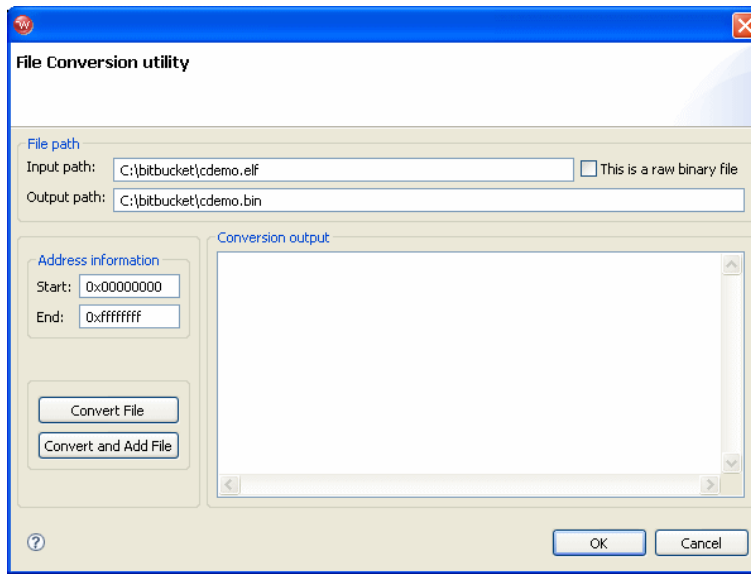
In order to use a file to program flash, you must convert it to a Wind River binary format that the **Flash Programmer** can use. Workbench can convert any of the following file types to Wind River binary format:

- **elf** files
- **hex** files
- **srec** files
- any headerless flat binary (RAWBIN) file

To convert a file to Wind River binary format, use the following steps:

1. In the **Programming** tab, select **Convert File**.
2. In the browser window that opens, navigate to the file you want to convert and click **Open**.

The **Convert** utility appears.

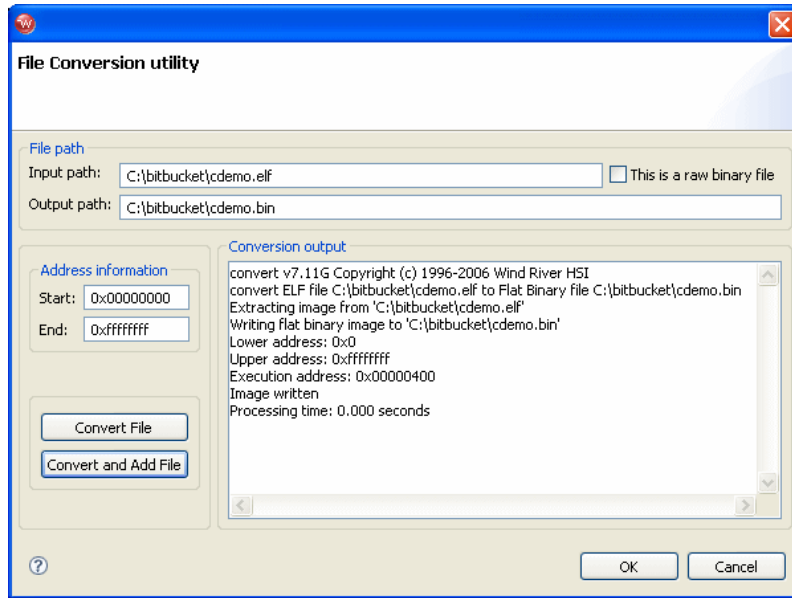


Converting the file to Wind River binary format does not delete the original file.

By default, Workbench stores the new binary file in the same location as the original file. If you want the new binary file stored somewhere else, enter the path to the desired location in the **Output path** field.

3. Select **Convert and Add File**.

Workbench converts the selected file to Wind River binary format and adds it to the file list in the **Programming** tab.



NOTE: To convert the selected file to Wind River binary format without adding it to the file list in the **Programming** tab, select **Convert File**.

4. Click **OK**.

You are returned to the **Programming** tab. The file you just converted now appears in the **File Path** field.

9.7.9 Setting The Download Offset Of A File

In some cases, before you program the file into flash, you may need to set a memory offset bias to divert the data to other areas of the flash bank.

Each file is built with a start address. This start address may or may not be the address where you want the image to reside on the board. If you subtract the start address of the image from the address where you want the image to reside on the board, then you end up with the proper bias address.

For example, if the image was built with a start address of 0x00 and you wanted the image to reside at the reset vector 0xFFFF00100, then the offset bias would be FFF00100.

You can use the **Add/Remove Files** area to edit the starting address of a **.bin** file to offset the file into flash. Click on the value under the **Start Address** heading to highlight it. Edit the value as needed.

9.7.10 Enabling A File For Download

Enable a file by clicking on the checkbox under the **Enabled** heading. If the file address is outside your specified address range, an error message appears:

```
Cannot enable for download.  
Part of this file falls outside your flash address range.
```

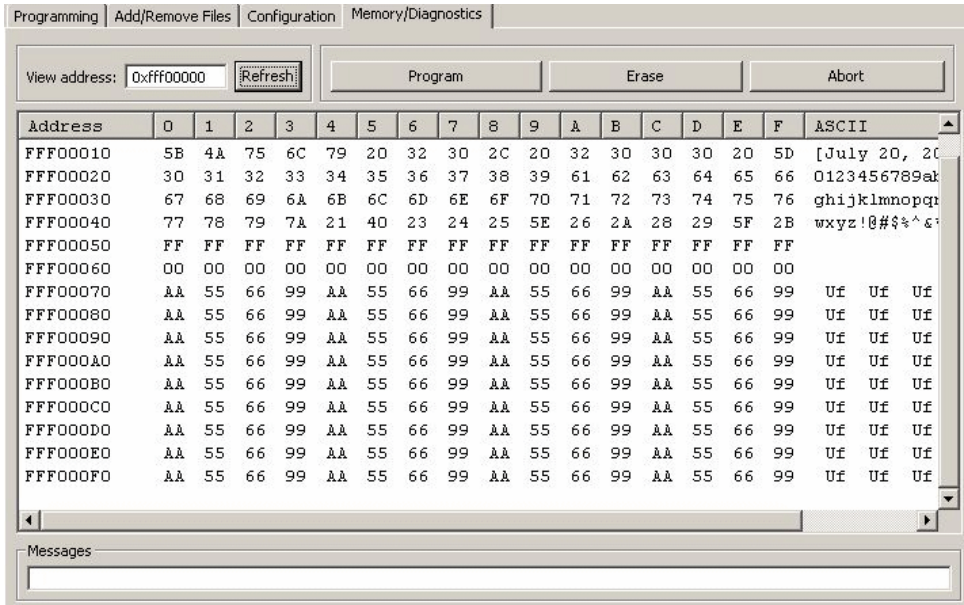
To correct this error, you must either change the start address of your file or use the **Configuration** tab to change your flash address range.

9.8 Flash Memory/Diagnostics Tab

Use the **Memory/Diagnostics** tab to view the contents of flash memory and to run diagnostic tests to verify your ability to write and erase flash.

You must set up the **Configuration** tab before using the **Memory/Diagnostics** tab.

Figure 9-4 Memory/Diagnostics Tab



9.8.1 Viewing Memory

Enter the address you wish to view in the **View Address** field. The area below displays the bit-level detail. To change the view, edit the address in the **View Address** field and click **Refresh**. You can also use the scrollbar on the right to scroll up and down from the starting address to the end address.

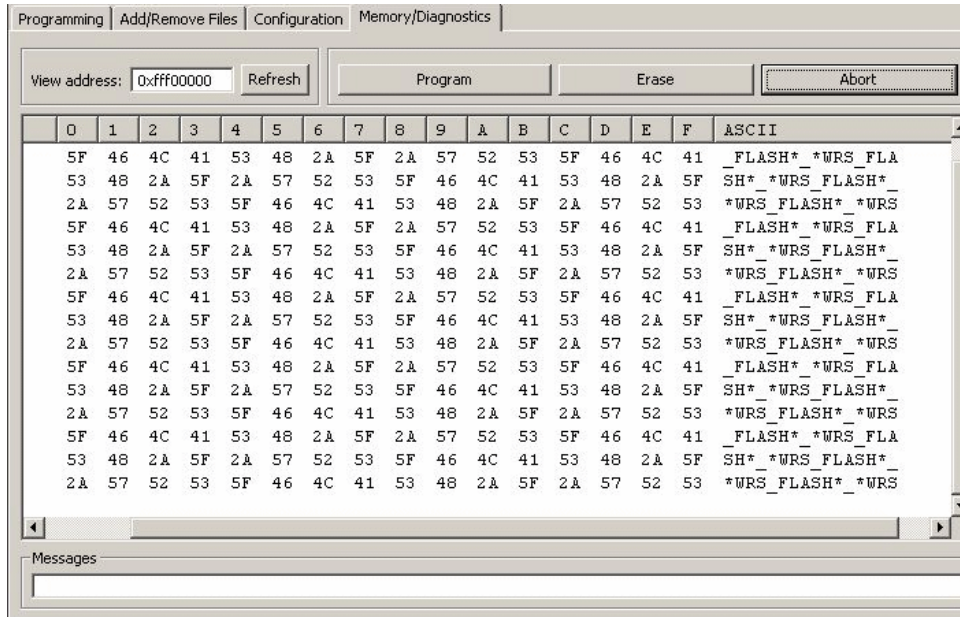
9.8.2 Running Diagnostic Tests

To test your ability to write to flash memory, click the **Start Program Diagnostic** button. This writes a bit pattern to flash.

You may see a **Target Exception** message. This requires no action.

If the write operation is successful, you should see the pattern ***WRS_FLASH*** repeated under the **ASCII** heading in the **Memory/Diagnostics** tab, as shown in [Figure 9-5](#).

Figure 9-5 **Successful Program Diagnostic**



If the write operation is unsuccessful, the diagnostic will never complete. You will need to click the **Abort Diagnostic** button to stop the write operation. Check to make sure that you have the right flash device selected in the **Device Selection** area in the **Configuration** tab, and that you are using the correct base address.

To test your ability to erase flash memory, click the **Start Erase Diagnostic** button. This will erase the selected flash sectors.

You may see a **Target Exception** message. This requires no action.

If the erase operation is successful, the selected sectors will be erased and the space under the **ASCII** heading in the **Memory/Diagnostics** view will be empty.

If the erase operation is unsuccessful, the diagnostic will never complete. You will need to click the **Abort Diagnostic** button to stop the erase operation. Check to make sure that you have the right flash device selected in the **Device Selection** area in the **Configuration** tab, and that you are using the correct base address.

10

On-Chip Debugging for Linux

- 10.1 Introduction 177
- 10.2 Linux Virtual Memory Management Architecture 178
- 10.3 Connection Parameters 179
- 10.4 Emulator Configuration 183
- 10.5 MMUL Settings 183
- 10.6 Booting a Linux System with OCD 185
- 10.7 Boot Line Commands 192
- 10.8 Reverse-Engineering the Boot Line Parameters 195
- 10.9 Debugging the Linux Kernel 196
- 10.10 Kernel Configuration 198
- 10.11 Debugging User Space Applications with OCD 198
- 10.12 Linux Troubleshooting 201

10.1 Introduction

This chapter describes the basic concepts required to use Wind River On-Chip Debugging (OCD) tools to debug a Linux system.

Wind River OCD tools for kernel and kernel module debugging allow you to develop Linux board-support code, such as LSP, drivers, stacks, I/O, and kernel priorities.



NOTE: Linux memory management is not supported for all processor families.

For general Workbench information, see the *Wind River Workbench for Linux User's Guide*.

10.2 Linux Virtual Memory Management Architecture

Since Wind River OCD tools typically access the entire physical address space (memory, devices, and peripherals), it's important to know how Linux manages the virtual memory space and translates to an OCD memory access.

The Linux memory architecture consists of three identified spaces:

- Kernel Space - Virtual and partially linear (generally fixed and not paged.)
OCD Kernel Mode supports *static translation* and *paged translation*.
 - Static translation covers all areas statically linked to the kernel.
 - Paged translation covers all dynamically allocated areas, such as kernel modules, **insmod** and **kmalloc**.
- Exception Space - Fixed and linear (component of kernel space, architecture dependent.) Used for Real Mode.

Real Mode is the mode of debugging of an application that is executing in the target when the Linux memory management unit (MMU) is disabled. In this mode the CPU is not translating addresses. It usually covers the startup of the kernel until the MMU is initialized and the exception vector.

- User Space - Virtual but not linear (demand paged). User space is the memory area where all User Mode application work can be swapped out when necessary.

Pages currently swapped out of these spaces to a filesystem cannot be accessed.

10.3 Connection Parameters

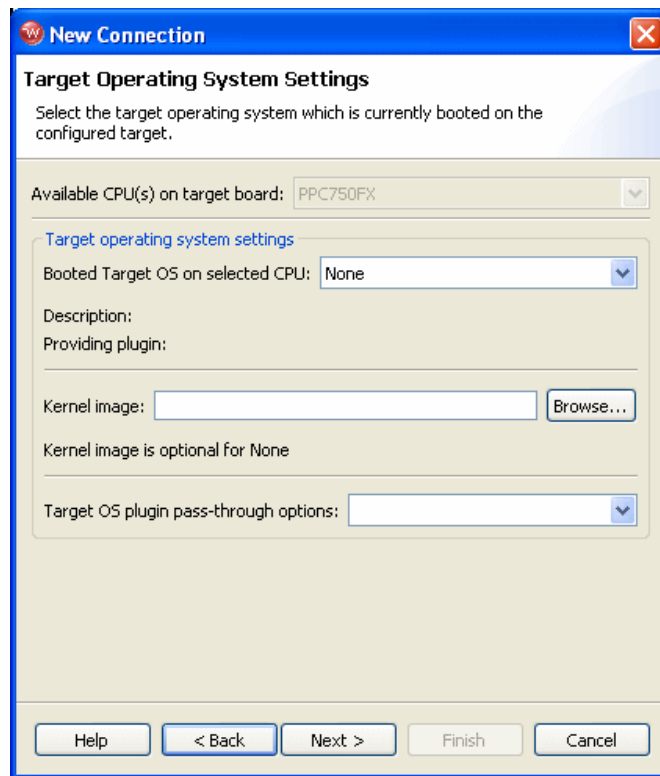
To use the Target Operating System (TOS) awareness for Linux, some configuration is necessary in the connection definition you create in the **Target Manager** view in Workbench.

1. Having selected your emulator type and target processor, click **Next**.

The **Target Operating System Settings** dialog appears, as shown in [Figure 10-1](#).

2. In the **Booted Target OS on selected CPU:** field, point to the kernel image that you are using. Workbench uses this kernel image only to load the symbols automatically, and not to download them physically to the target.

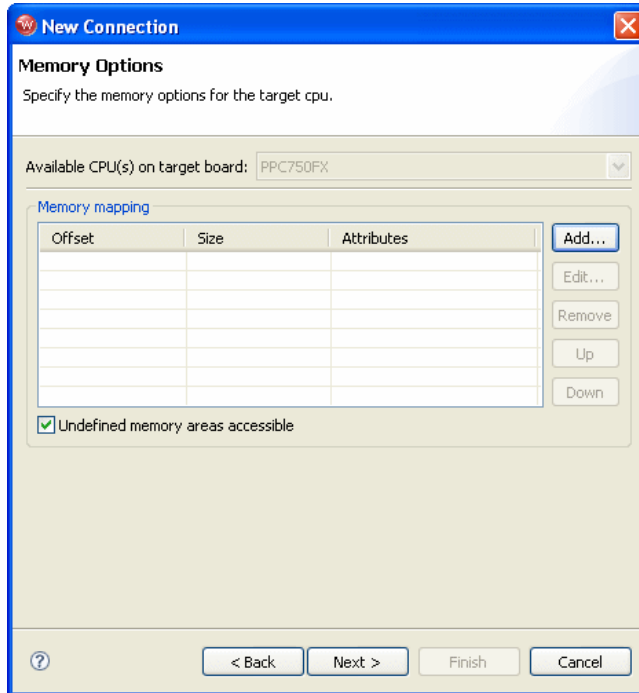
Figure 10-1 **Target Operating System Settings**



3. Click **Next**.

The **Memory Options** dialog appears, as shown in [Figure 10-2](#).

Figure 10-2 **Memory Options**



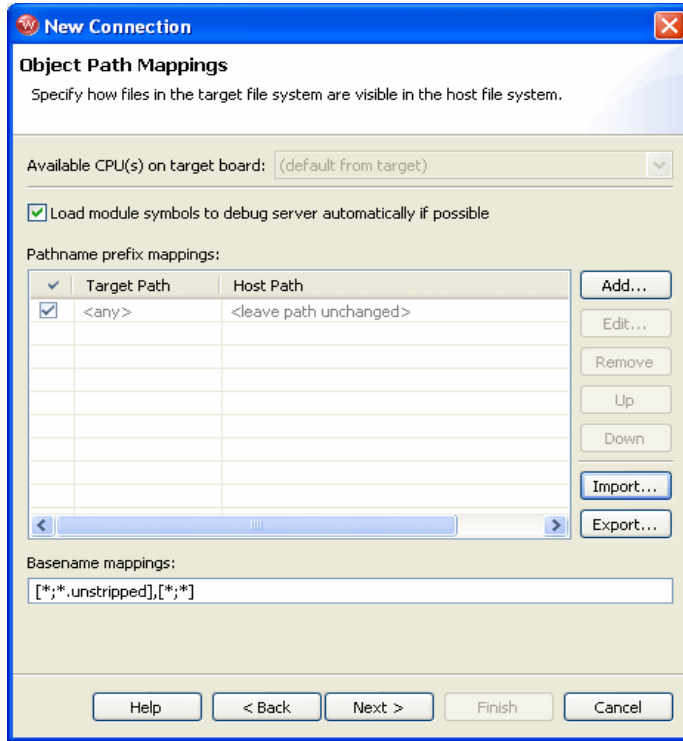
Use this dialog to specify the base address and size of the physical RAM that is allocated to the Linux system.

4. Click **Next**.

The **Object Path Mappings** dialog appears, as shown in [Figure 10-3](#).

5. Select **Load module symbols to debug server automatically if possible**.
6. Define the path substitution required between your target filesystem and host filesystem. If your system loads a large number of modules automatically, you may want to load the symbols manually for only the subset wanted.

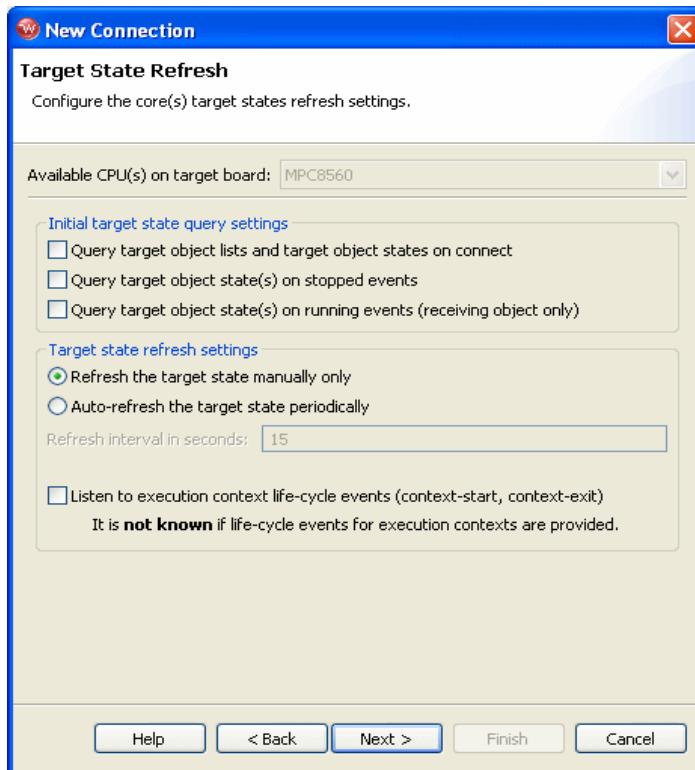
Figure 10-3 Object Path Mappings



7. Click **Next**.

The **Target State Refresh** dialog appears, as shown in [Figure 10-4](#).

Figure 10-4 Target State Refresh



8. Select your desired refresh method.

The default setting is **Refresh the target state manually only**. Retrieving target memory through OCD is very time-consuming, so if you select **Auto-refresh the target state periodically**, it can slow your project significantly.

The change of state (running or stopped) is event driven, and does not require auto-refresh to update Workbench on these events.

9. Click **Next** to bring up the connection summary.
10. Click **Finish**.

10.4 Emulator Configuration

Once you have connected to your emulator, you must enable the MMU configuration.

In the **CF Options** view, set the **Memory Management Unit Mode (MMU)** option to **ENABLED**, or enter the command

```
>BKM>cf mmu enable
```

at the **>BKM>** prompt in the **OCD Command Shell**.

This will enable the translation mechanism required to debug a Linux kernel with an OCD connection.

If you plan to use OCD to transfer the boot line to the kernel (see [10.6 Booting a Linux System with OCD](#), p.185):

In the **CF Options** view, set the **Load Boot Table on IN (BL)** option to **ENABLED**, or enter the command

```
>BKM>cf bl enable
```

at the **>BKM>** prompt in the **OCD Command Shell**.

Enabling this option will cause the boot line to be written into memory upon every target reset.

10.5 MMUL Settings

Linux has address locations that are defined and translated before runtime. For instance, the Linux image may be compiled at an effective address of 0xC0000000, loaded into the target at RAM location 0, and run from location 0 until the Memory Management Unit is initialized and turned on. Other locations may be I/O, DMA, or other fixed and linearly mapped locations, such as the address locations of exceptions that run in Real Mode. The MMU commands allow you to pre-map these locations for facilitating downloads and improving performance.

MMU commands:

- **MMUL**: List the pre mapped translation(s)
- **MMUA**: add a translation

- **MMUD:** remove one or all translations

In most cases, only one MMUL translation needs to be defined: the one that maps the entire physical address space into the kernel space.

Syntax

MMUL *logical_address physical_address mask Mode Process_ID*

Example 1

For a board that has 512 MB of RAM and the kernel linked at 0xC0000000, the following MMUL is required:

Logical address: 0xC0000000

Physical Address: 0

Mask: 0xe0000000 // mask a region of 512 MB

Mode and Process ID (PID) can be set to 0, as they are reserved for future use.

At the >BKM> prompt in the **OCD Command Shell**, enter

```
>BKM>mmua c0000000 0 e0000000 0 0
```

Example 2

To map direct access to a 64 KB I/O region located at FE0000000 and 64KB:

At the >BKM> prompt in the **OCD Command Shell**, enter

```
>BKM>mmua fe000000 fe000000 ffff0000 0 0
```

Example 3

Some PowerPC architectures (such as PPC6xx, PPC7xx, and PPC82xx) disable their MMU entirely when entering an exception. These architectures require a translation to map the bottom of the memory so the tools can debug exception handler (64K in this example):

```
>BKM> mmua 0 0 ffff0000 0 0
```

These settings are persistent, and need to be entered only once.



NOTE: The Workbench Linux plug-in automatically sets the MMUA to 0xC0000000 based on the memory settings you enter in the **Memory Options** page in the **New Connection** wizard.

10.6 Booting a Linux System with OCD

There are two main methods to boot a Linux target with OCD tools:

- Rely on the existing boot loader, such as **redboot**, **uboot**, or **yamon**, to boot the system as if the OCD tools were not connected to the target.
- Use the OCD tools' boot line capabilities to download and boot a Linux kernel image without using a boot loader.

10.6.1 Standard Boot

If a boot loader is already developed for your target, this is probably the easiest way to start debugging a Linux System with OCD tools.

You need to reset the target, let the target resume from its reset vector, and let the boot loader load and boot the Linux image.

If a boot loader is resident and configured to boot Linux, you do not need a register file.

Workbench automatically installs a hardware breakpoint at the start of the Linux kernel to detect that the kernel has been loaded. This enables the TOS awareness features (such as the List of Processes), as well as all the breakpoints you have already defined.



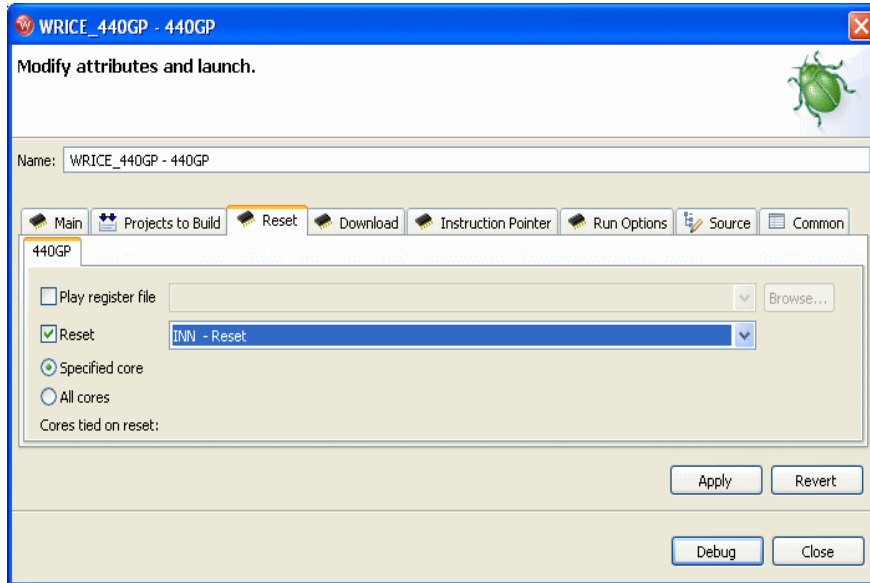
NOTE: After every reset, all user breakpoints are automatically disabled until the kernel has started.

For a standard boot, use the following steps:

1. In the **Target Manager** view, select **Reset and Download**.

The **Reset and Download** view appears, as shown in [Figure 10-5](#).

Figure 10-5 **Reset Tab**

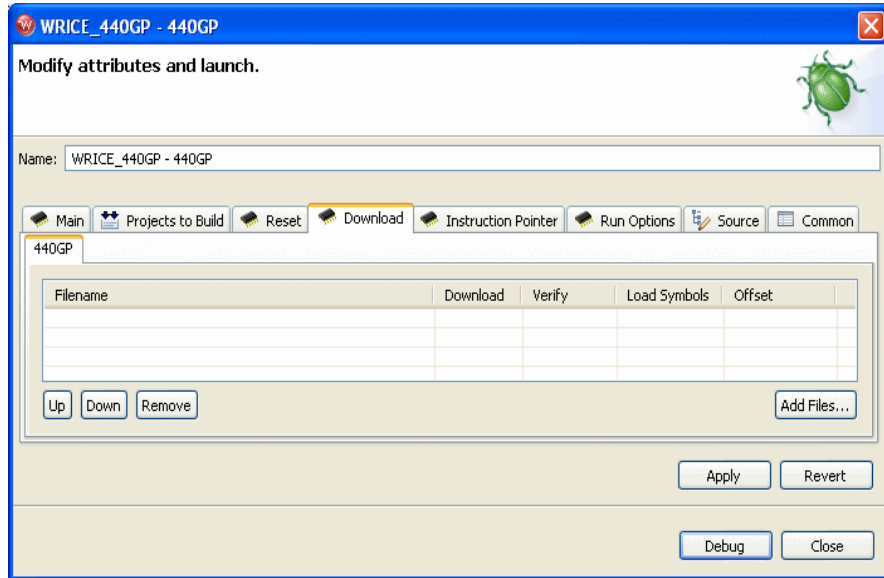


2. In the **Reset** tab, check the **Reset** box and select **INN -- Reset**.

On an INN command, the tools will reset the target without initializing any of the peripherals.

3. Select the **Download** tab.

Figure 10-6 Download Tab

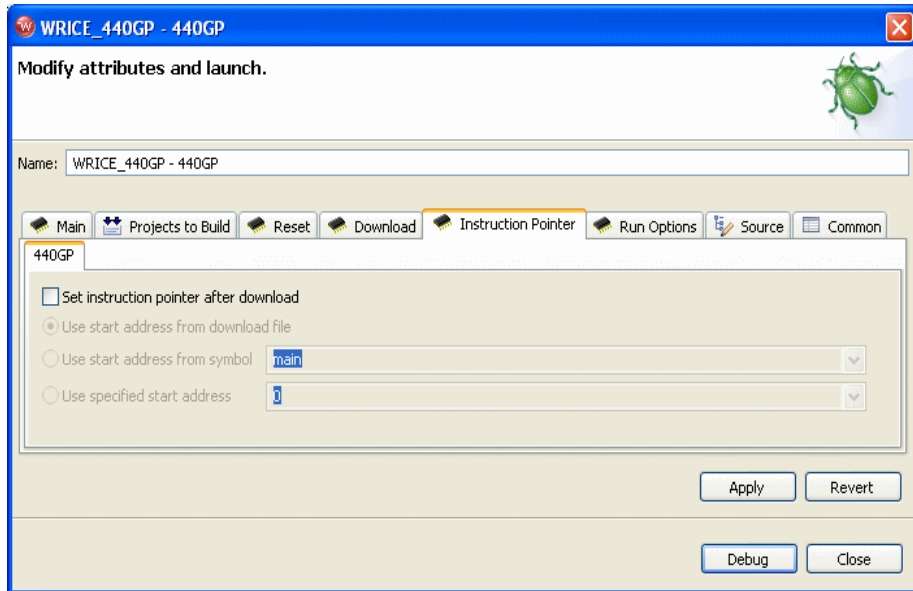


If you entered a kernel image name in the **Target Operating System Settings** dialog (see [10.3 Connection Parameters](#), p.179), you do not need to specify a download file.

If not, select your Linux image (**vmlinux**) and select the checkbox in the **Download** field. Set the **Verify** field to **None** and leave the **Load Symbols** field unchecked.

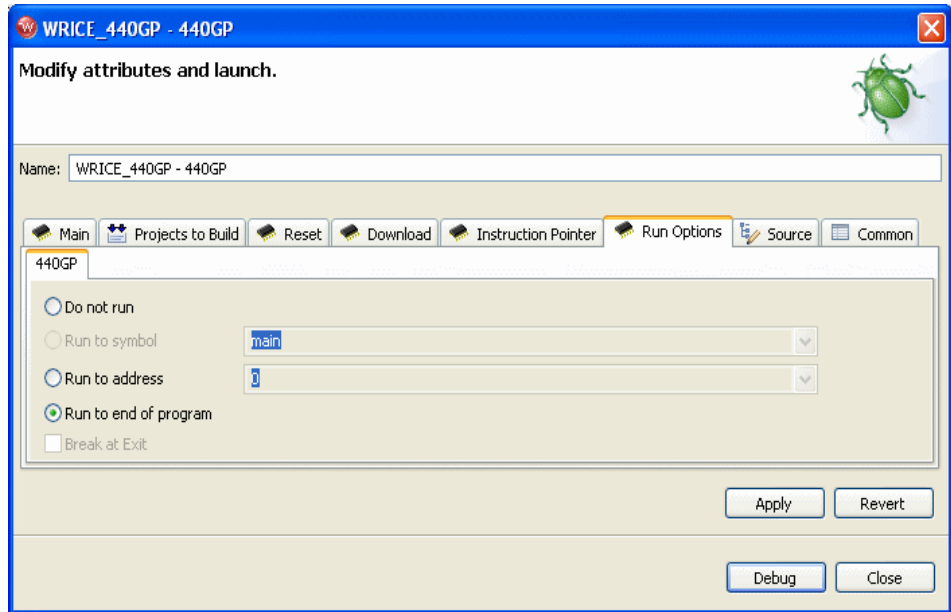
4. Select the **Instruction Pointer** tab.

Figure 10-7 **Instruction Pointer Tab**



5. Uncheck **Set instruction pointer after download**. The INN reset will leave the instruction pointer already at the reset vector.
6. Select the **Run Options** tab.

Figure 10-8 Run Options Tab



7. Select **Run to end of program**. This automatically starts the target after the reset is completed. (Optional, but recommended for targets running a Watchdog timer by default to reduce the time elapsed between INN and GO.)



NOTE: AMCC 40X and 44X processors have a debug control register (DBCR0) that controls debug event conditions, which can affect the operation of any emulator. By default, **uboot** clears this register, which disables the breakpoint mechanism. Either use a version of **uboot** that does not clear this register, or select **Play post download script** and specify a script to issue the command `DR DBCR0 81000001` to the target, thereby re-initializing the **DBCR0** register for emulator debugging.

8. Click **Debug**.

10.6.2 OCD Boot

OCD tools can download the kernel image directly into memory, allowing you to start the kernel without boot loader involvement.

If you are loading and starting the Linux kernel directly through JTAG, without a boot loader, you need a register file. The register file needs to initialize the registers for correct memory access as well as the boot line.

No MMUL or MMUOS settings should be part of the register file; these parameters are initialized upon connection to the target, when a symbol file is loaded (or reloaded.)

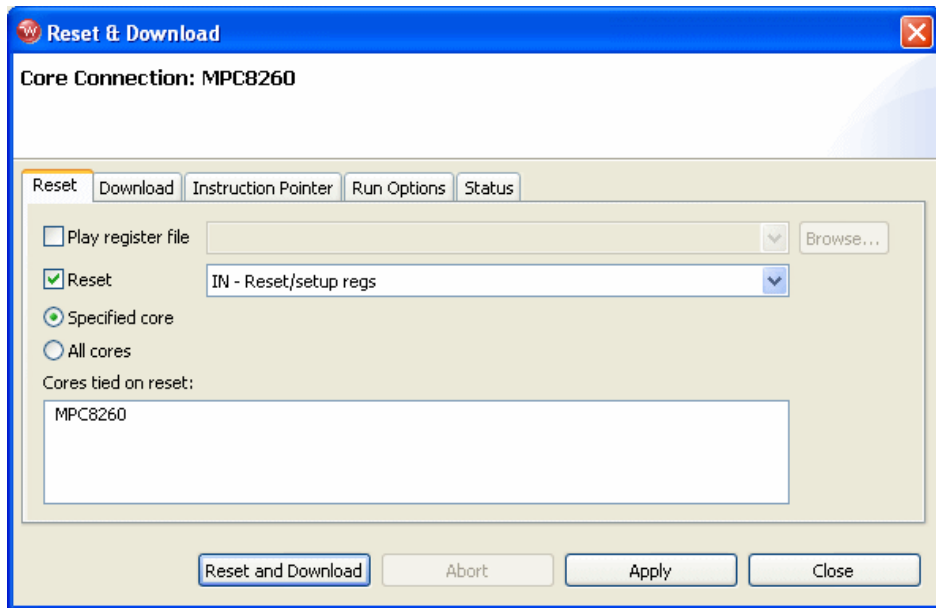
The OCD tools will reset the target, initialize the required target peripherals according to the values defined in the register file, and initialize the required boot parameters into memory as defined in the BL command.

To perform an OCD boot, use the following steps:

1. In the **Target Manager** view, select **OCD Reset and Download**.

The **Reset and Download** view appears, as shown in [Figure 10-9](#).

Figure 10-9 **Reset Tab**

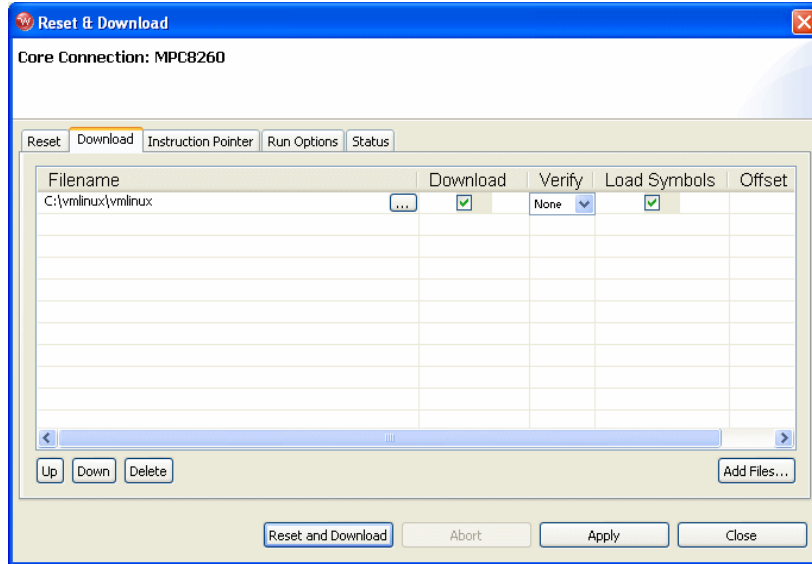


2. In the **Reset** tab, check the **Reset** box and select **IN -- Reset/setup regs**.

On an IN command, the tools will reset the target, initialize peripherals with the register file value, and initialize the boot line.

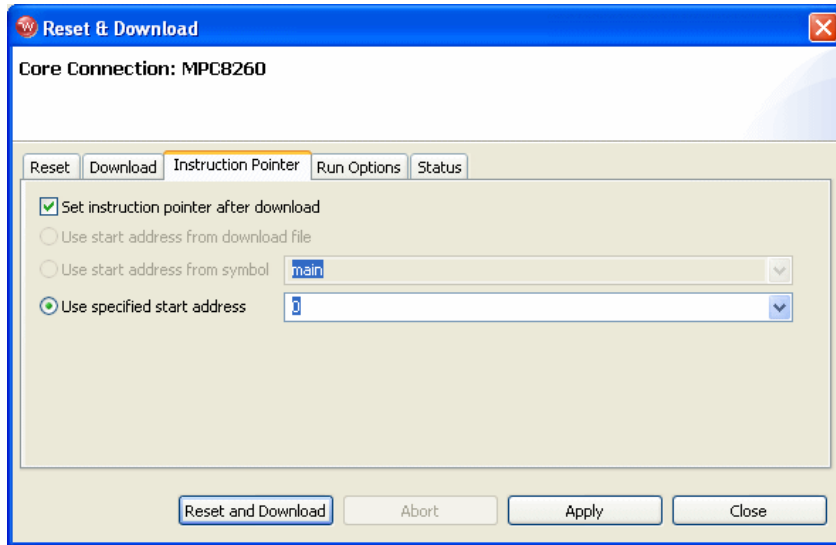
3. Select the **Download** tab.

Figure 10-10 **Download Tab**



4. Click **Add Files** to specify the kernel image to download. Select both **Download** and **Load Symbols**. Set the **Verify** field to **None**.
5. Select the **Instruction Pointer** tab.

Figure 10-11 **Instruction Pointer Tab**



6. Select **Set instruction pointer after download**.
7. Select **Use specified start address** and set the instruction pointer to the start of the kernel.



NOTE: For all architectures that start with the MMU disabled, the start address is not the one defined in the *.elf file, but the translated one to the real space. For example, 82xx **vmlinux** reports a start address at 0xC0000000, but the real start address is 0.

8. Click **Reset and Download**.

10.7 Boot Line Commands

Wind River emulators support various low-level commands and configuration options that allow you to create, define, pre-set, and manage the Linux boot line parameters and Linux kernel paging mechanism.

The following table gives an overview of the low-level commands and configuration options that are specific to Linux boot line configuration. For full descriptions, see the *Wind River Workbench On-Chip Debugging Command Reference* and the *Wind River Workbench for On-Chip Debugging Configuration Options Reference*.

Table 10-1 Low-Level Boot Line Commands

Command	Examples	Description
BL	BL ADD BL DELETE BL MODIFY BL DISPLAY BL UPLOAD	These commands allow you to customize the location and values for the Linux boot line, which are automatically passed to Linux through a register group. Available only when the configuration options MMU and BL are set to ENABLED .



NOTE: In addition, these commands may require other resources, such as target workspace (RAM). For instance, the **BL** command requires you to allocate target memory using the **CF WSPACE** configuration option.

Like other operating systems, Linux requires parametric information about its network, file system, and target board environment. In some cases, these boot line parameters may be compiled into the Linux image as default parameters; or Linux may acquire these parameters from the boot loader for more versatility. Consequently, Linux must be customized to accept the parameter format of the boot loader (for example, **uboot**). Generally, Linux retrieves pointers or values to boot line parameters by using registers. Boot line parameters include memory information, IP address information, and so on. Incorrect boot line parameters are an obstacle to the successful bring-up of Linux.

The various **BL** commands allow you to create any number of structures, which are appropriate for virtually any Linux startup. The structures may reflect the fields of the Linux architecture-specific structure **bd_info bd_t**; the Linux boot line; or even the MAC address assigned to the board.

Pointers to the structures are generally passed using a group of registers. The structures and their registers can be entered and held by the emulator using a **BL** command and passed on a **GO** command. Once both tables (Structure and Register) are configured with the **BL** commands, Linux boot loader code is no longer required to be pre-installed-embedded into the target and/or included within the downloaded application. Those tables are retained in the emulator's NVRAM, so their contents are persistent from one session to another.

The following example displays a customized set of boot line parameters for a particular distribution of Linux that have been generated by the **BL** command and stored in the emulator's NVRAM. On a **GO** command, these parameters will be loaded into the target memory and passed to Linux via a set of register pointers. This process is controlled by the emulator's run-time firmware DLL.

Example:

Dynamic Boot Table: structure configuration

Entry	Description	Value/String
00	MemStartAdd	0x00000000
01	MemSize	0x04000000
02	FlashStart	0x40000000
03	FlashSize	0x00400000
04	FlashOffset	0x00040000
05	SRAMStart	0x00000000
06	SRAMSize	0x00000000
07	IMMR_Base	0xf0000000
08	BOOTFlags	0x00000001
09	IP_ADDR	0x00000000
10	ENETADDR[6]	0x00a01ea87bcb
11	ETHSPEED	0x6c79
12	INTFREQ	0x0bcd3d80
13	vBUSFREQ	0x01f78a40
14	CPMFREQ	0x03ef1480
15	BRGFREQ	0x01f78a40
16	SCCFREQ	0x01f78a40
17	VCO	0x07de2900
18	BAUDRATE	0x00002580
19	bi_mon_fnc	0x0fffffff
20	CmdStrg	->console=ttyS0,9600 root=/<- ->dev/ram0 rw

The **Entry** field is a sequential reference for each line item.

The **Description** field is an ASCII field only used for comment.

The **Value/String** field can contain a char, byte, unsigned long, or unsigned short value.

- Unsigned long values are displayed in hex using 8 digits, as shown in Entry 01.
- Unsigned short values are displayed in hex using 4 digits, as shown in Entry 11.
- Char values are displayed as shown in Entry 20. (Char strings greater than 20 characters are displayed on several lines, using arrows.)
- Byte values are displayed as shown in Entry 10.

Examples of boot lines for various architectures and boot loaders are provided as part of the Linux register files distributed with Wind River Workbench.

10.8 Reverse-Engineering the Boot Line Parameters

The boot line parameters change with the target architecture, the Linux architecture, and the developer. You may desire the boot line parameter information without knowing the low-level details in order to set up a boot line script. In most cases, this is possible by acquiring an operational target board with startup/bootstrap firmware and doing the following:

1. Set up the target with your Wind River emulator.
2. Allow the boot ROM to run from ROM by issuing the **INN** and **GO** commands. (Assuming you have set up the **MMUL** properly.)
3. Download your image using **Tftpboot**.
4. Halt the emulator by issuing a **HA** command, and set a hardware breakpoint at the beginning of where your Linux image will start. This start location can be retrieved from the **System.map** of the **vmlinux** build.



NOTE: Some processors have debug control registers that are essential for emulator function, and which may be manipulated by the boot ROM. Halting after **Tftpboot** may allow the emulator to reacquire control of these debug control registers.

5. Uncompress and boot your image using **bootm**.
6. After hitting the hardware breakpoint, issue a **DR** command to display the register setup before Linux starts.

The registers contain pointers to the boot information structures. For **uboot**, consider **r2**, **r3**, **r6**, and **r7** (**r4** and **r5** are generally the RAMdisk pointers and should be set to zero if there is no RAMdisk).

7. At the **>BKM>** prompt, enter **DML (r3)** and press **ENTER** several times. This displays the boot information. Do not make any changes at this time. This is only to understand what your boot flash put into memory for Linux boot information.

Now you should be able to download your Linux image (as before, but without a reset or register load) and run the image as if it were booted from ROM. If the boot ROM successfully boots the Linux image, then your Linux image is compatible with your boot ROM and the image of the boot information structure can be retrieved by the emulator. If not, you must set up the boot line in a register file in accordance for Linux. To do this, you need to know what your Linux image expects for boot information (in memory).

8. The downloaded image (via the emulator) is now functioning properly and has the proper boot line and register setup from ROM. Next, repeat Steps 2 through 5 and capture/duplicate the register setup (as in Step 6) and memory storage with the boot line facility.

10.9 Debugging the Linux Kernel

Wind River Workbench OCD Edition allows you to debug every part of the kernel, starting at the very first opcode, as if it were any standalone application. It is possible to debug the early initialization routines, as well as the entire boot sequence.

Wind River recommends that the kernel be built with **-gdwarf2** to optimize the symbol reading performances in Workbench.



NOTE: Due to the required optimization level used to build the Linux kernel, some of the source level single stepping operation seems to not follow the code flow, and steps out of order.

10.9.1 Debugging Linux Kernel Modules

Linux kernel modules are dynamically allocated in the kernel space when an **insmod** is performed.

Kernel Module Detection

Workbench automatically detects that a module was installed and lists the module name in the **Target Manager** view.

Workbench transparently installs breakpoints in the `sys_init_module` and `free_module` functions to keep track of the module loading and unloading in the system.

If you defined the target `RootFileSystem` path in the connection wizard (see [10.3 Connection Parameters](#), p.179), Workbench can automatically read the symbol file and relocate the different sections according to where the module has been allocated, to allow debugging of the module. (You can also manually load the symbol file by right-clicking the module name and selecting `Load Symbol File`.)

Debugging the `init()` Function of a Module

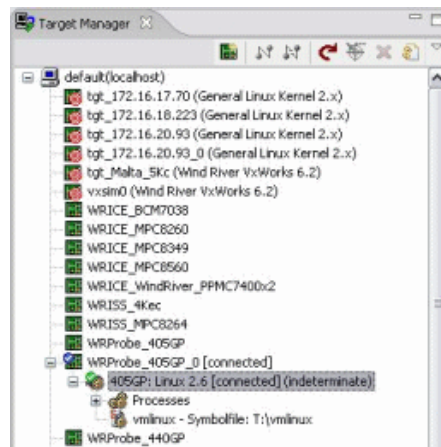
To debug the `init()` function of a module, use the following steps:

1. Place a breakpoint in the `sys_init_module` function where the `mod->init()` is called.
2. In the Linux shell, enter the command `insmod module_name`.
3. When the breakpoint is hit, step in the `init()` function.

Note that Workbench automatically detects the module when `sys_init_module` is exited.

To see the module appear in the **Target Manager** view while stopped in the middle of `sys_init_module`, select the target that needs to be refreshed in the **Target Manager** view and press the **Refresh** button in the top right of the view.

Figure 10-12 Target Manager View



10.10 Kernel Configuration

For the Linux 2.6 kernel, the kernel module section link information is not persistent in the kernel structures by default. For Workbench to properly relocate the different modules' symbol files, this information must be made persistent using one of the following methods.

- For kernel versions 2.6.12 and above, set **CONFIG_KGDB**.
- For kernel versions 2.6.11 and below, you must both set **CONFIG_KGDB** and apply a patch to the kernel. The necessary patch is named **module.patch** and is located in *installDir/wrwb_2.x_50/linux-2.x/kgdb/wrs-2.2.1/linux-2.6.10*.

If turning on KGDB is an issue, the equivalent functionality can be obtained by enabling the code conditionally compiled for **CONFIG_KGDB** in the */kernel/module.c* and */include/linux/module.h* files.

In both **module.c** and **module.h**, replace

```
#ifdef CONFIG_KGDB
```

with

```
#if defined (CONFIG_KGDB) || defined (CONFIG_WINDRIVER_OCD)
```

In **module.h**, add

```
#define CONFIG_WINDRIVER_OCD
```

(There are four instances in **module.c** and one in **module.h**.)

10.11 Debugging User Space Applications with OCD

You can use the **Target Manager** in Workbench to attach to a running process or a particular thread of a process. This creates a debug context to see the state of a particular process and debug it. The debug context thus created represents the *User Mode* context of the attached process.

Every Linux application has two contexts: User Mode and Kernel Mode. Both modes use the same process ID (PID.)

Workbench can attach only to threads that have a User Mode context. Workbench cannot currently attach to kernel threads.

The User Mode context can make system calls to the Kernel Mode context, but you can see Kernel Mode context only in the system context.

OCD allows full debug capabilities within the user space application and its shared libraries.

10.11.1 Attaching to a Process

You cannot create a process with OCD; you can only attach to one that is already existing.

To attach to a running process, use the following steps:

1. In the **Target Manager**, click the **Refresh View** button.
2. Right-click on the process you want to attach to and select **Attach to Process**.
This option is only available for User Mode processes. For processes where it is not available, the option will be greyed out.
3. Currently there is no way to auto-resolve a symbol file for the application, so you must load symbols manually. Right-click on the attached process and select **Load/Add Symbols to Debug Server**.
The **Load/Add Symbols** dialog appears.
4. Click **Add**.
5. In the browser window that opens, navigate to the symbol file you wish to use and click **Open**.
You are returned to the **Load/Add Symbols** dialog.
6. Click **OK**.

10.11.2 Debugging a Process

Use the **Debug** view to run, step, and set breakpoints in process threads.



NOTE: You cannot stop only one thread. Whenever a breakpoint is hit in any thread, the whole system stops.

10.11.3 Setting Breakpoints

Linux uses on-demand memory allocation; memory is not allocated to an application until it needs it. You cannot set software breakpoints in areas of memory that are not yet allocated. Attempting to set a software breakpoint in unallocated memory returns an error:

```
!ERROR! [msg 170000] : Unable to map virtual address
```

However, you can always set hardware breakpoints in the entire address space.

10.11.4 Thread-Qualified Breakpoints

By default, a set breakpoint goes to the parent; but Workbench allows you to set the scope of a breakpoint to a single thread by selecting the drop-down menu in the **Breakpoints** view and selecting **Breakpoint Preferences**.

However, whenever the breakpoint is hit, Workbench will stop the target, determine if the breakpoint was hit in the correct thread, and if not, restart the target. This transparent restart of the target can have an impact on the real-time aspect of the application.

This is also true of processes. If you run the same process several times, the code section of the application is not replicated; all the processes use the same area of user space. So if a breakpoint is hit in one process, all other processes will stop to check if the breakpoint was hit in the correct process and then resume.

10.11.5 Debugging the Beginning of a Process

To debug the beginning of a process, use an internal hardware breakpoint.

Set an expression hardware breakpoint from the system debug context at the start address of the application. Then start your process.

10.11.6 Limitations

- You cannot start a process using OCD; you can only attach to an already existing process, started from the shell or elsewhere.
- There is no notification when a process dies. You must detach from the process manually.

- Workbench cannot currently attach to kernel threads. You can debug kernel threads through the system context.

10.12 Linux Troubleshooting

If you are having trouble, check the following:

- Make sure the kernel symbol file has debug information.
- Make sure the **MMUL** command returns 1 translation of type **PHY-KERN** to match the Linux kernel translation.
- Make sure the **MMUOS** command returns a list of parameters consistent with the Linux kernel version you are using.

11

Using the WDB Transparent Mode Driver

11.1 Introduction 203

11.2 Connecting Through the Transparent Mode Driver 204

11.3 Using the TMD With the Wind River ICE SX 206

11.4 Configuring the Target Server 209

11.5 Moving On 218

11.1 Introduction



NOTE: The Transparent Mode Driver is not supported for Wind River Probe. This chapter applies only to the Wind River ICE SX.

In VxWorks debugging, a Wind River Debug (WDB) agent runs on the target as a kernel task to provide debugging support. You can use the WDB agent to debug kernel tasks and real-time processes on the target. The WDB agent specifies how the target server on the host communicates with the target agent on the board. Typically, Workbench communicates with the WDB agent using an Ethernet or serial connection.

In some cases an Ethernet or serial connection may not be available for use; for example, if your target does not have an Ethernet or a serial port on it, or if you are

using those ports for some other purpose. In this case you can use the Wind River Transparent Mode Driver.

The Transparent Mode Driver provides an alternate communications channel for allowing Workbench to talk to the WDB agent. The Transparent Mode Driver works through the Wind River ICE SX, implementing communication over the BDM/JTAG/EJTAG connection. The connection to the target operates entirely through the standard BDM or JTAG debug link.

The Wind River Transparent Mode Driver supports all of the debug capabilities of the WDB agent, including system mode, task mode, and virtual I/O. When connected through the Transparent Mode Driver, the Wind River ICE SX also functions as a second on-chip debug channel. This allows you to use the ICE to download VxWorks images, set breakpoints in the kernel, debug device drivers, and so on, in addition to using the WDB agent for task and process debugging.

In order to use the Transparent Mode Driver, you must set your Wind River ICE SX to TMD mode, as described in [TMD Mode](#), p.207.

You must also incorporate the Transparent Mode Driver into your Workbench build. For information doing this, and for general information on the WDB agent, please see the *Wind River Workbench User's Guide: Setting Up Hardware*.

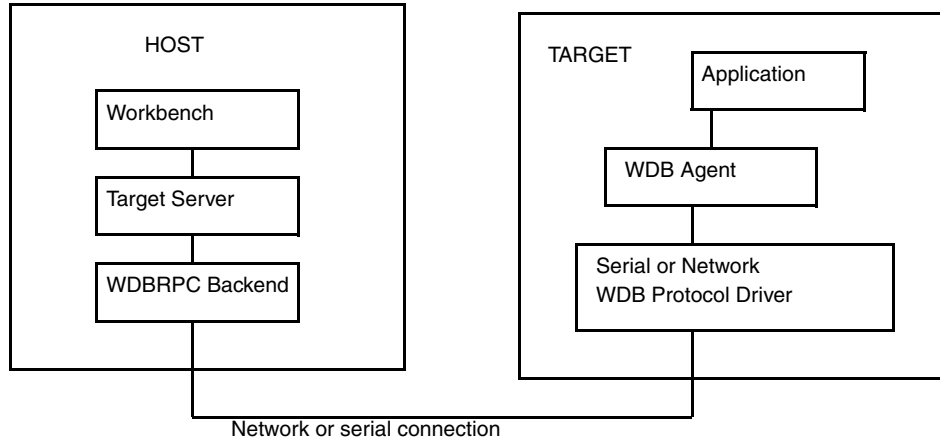
11.2 Connecting Through the Transparent Mode Driver

The most common use of the Transparent Mode Driver is to act as a WDBRPC connection mechanism to the WDB target agent.

This section briefly explains some of the technical details regarding how the Transparent Mode Driver works when it is being used as the link connecting the WDB agent on the target and the target server on the host.

When the Transparent Mode Driver is not being used, the host to target agent connection consists of either a network or a serial connection, as shown in [Figure 11-1](#).

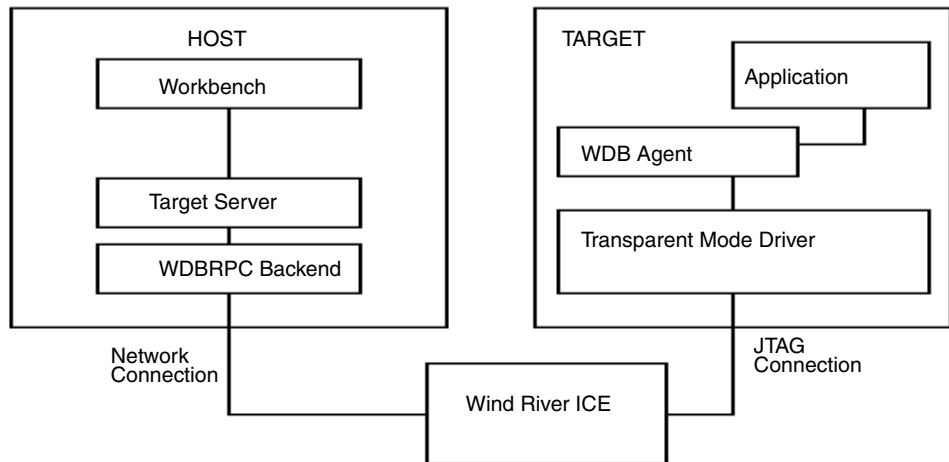
Figure 11-1 Typical Host to Target Agent Connection



For this connection scheme to work, you need a fully functional Board Support Package (BSP), along with known good hardware, as well as network and/or serial drivers.

When you use the Transparent Mode Driver, the connection scheme is simplified, as shown in [Figure 11-2](#).

Figure 11-2 Connection Using the Transparent Mode Driver



In this scenario, the Wind River ICE SX provides the link between the backend and the target agent. From the perspective of the host tools, the Transparent Mode Driver appears to be identical to the standard supplied connection (either an Ethernet or a serial connection).

When you are using the Transparent Mode Driver, the only difference when setting up the target server is that the IP address of the Wind River ICE SX is listed in the target server configuration dialog box instead of the IP address of the target.

Information on configuring a target server is available in [11.4 Configuring the Target Server](#), p.209.

11.3 Using the TMD With the Wind River ICE SX

You may need to configure your Wind River ICE SX before you can use it with the Transparent Mode Driver.

11.3.1 Configuring Wind River ICE SX

Information on configuring Wind River ICE SX for network operation is available in the *Wind River ICE SX for Wind River Workbench Hardware Reference*. Follow the instructions provided in that document to configure your Wind River ICE SX for network operation.

There is one additional option that must be set when you are using the Transparent Mode Driver; the UDP Console Port must be set to **17185**. The following steps explain how to do this.

1. In the **ethsetup** menu, accessible from the **>NET>** prompt (as described in the *Wind River ICE SX for Wind River Workbench Hardware Reference*), select Option 5 to view the current port settings.
2. If **UDPCNSL** is already set to **17185**, no modifications are necessary, and you should exit the **ethsetup** menu.
3. If **UDPCNSL** is not set to **17185**, type **6** to allow the port values to be changed.

A list of port settings appears.

4. Type the number assigned to the UDPCNSL port, which will allow the UDPCNSL port setting to be changed, and change it to **17185**.
5. Type **0** to exit the **Change Port Settings** menu.
You are returned to the main **ethsetup** menu.
6. To save the changes in the ICE unit's NV-RAM, select Option 8 and press **ENTER**.
7. Select Option 9 to exit the **ethsetup** menu.
You are returned to a **>NET>** prompt in the **Terminal** view.
8. Power cycle the Wind River ICE SX unit so these changes take effect.
The ICE unit runs through the same series of internal tests as on initial startup, making sure that all the hardware and firmware in the unit is functioning correctly. These tests are again displayed in the **Terminal** view in Wind River Workbench, and when they conclude, the **> NET >** prompt is again visible in the view.

11.3.2 Configuration Options

For some processors, you may also need to change some of the configuration options on your Wind River ICE SX.



NOTE: For Freescale ColdFire processors, setting the CF options **TMD Mode** and **Trap Exception** is not necessary. For ColdFire processors you only need to set the CF option **Target Console Redirection**.

Setting CF Options in the CF Options View

In the Workbench toolbar, click on **Window** and select **Show View > CF Options**.

TMD Mode

1. Under the **Command Name** heading in the **CF Options** view, scroll down to **TMD**.
2. Double-click on the value under the **Current Setting** heading to bring up a list of options.
3. Select **ENABLE**.

4. Click on the **Send All CF Options to Target** icon.

Trap Exception

1. Under the **Command Name** heading in the **CF Options** view, scroll down to **TRPEXP**.
2. Double-click on the value under the **Current Setting** heading to bring up a list of options.
3. Select **BREAKPOINTONLY**.
If **BREAKPOINTONLY** is not available for your target board, set **TRPEXP** to **NO**.
4. Click on the **Send All CF Options to Target** icon.

Target Console Redirection

1. Under the **Command Name** heading in the **CF Options** view, scroll down to **TGTCONS**.
2. Double-click on the value under the **Current Setting** heading to bring up a list of options.
3. Select **BDM**.
4. Click on the **Send All CF Options to Target** icon.

Setting CF Options with Low-Level Commands

You can also set these options using low-level commands.

1. In the Workbench toolbar, click on **Window** and select **Show View > OCD Command Shell**.
2. At the **>BKM>** prompt in the **OCD Command Shell**, type **CF**.
A list of the current settings for the CF options on your Wind River ICE SX will appear.
3. At the **>BKM>** prompt, type **CF TMD ENABLE** and press **ENTER**.
4. Type **CF TRPEXP BREAKPOINTONLY** and press **ENTER**.
If **BREAKPOINTONLY** is not available for your target board, set **TRPEXP** to **NO**.
5. Type **CF TGTCONS BDM** and press **ENTER**.
6. Type **CF** again.

The CF options now show the values you entered.

7. Type the command **IN** or **INN** to reset the processor. Your changes will not take effect until you reset the processor.

11.4 Configuring the Target Server

If an image is running on your target, your host will be able to communicate with the running WDB agent. To do this, you must configure and activate a Target Server. To configure a Target Server, use the following steps.

First, open Workbench according to the method for your host computer.

Linux/Solaris Hosts

From your installation directory, issue the command

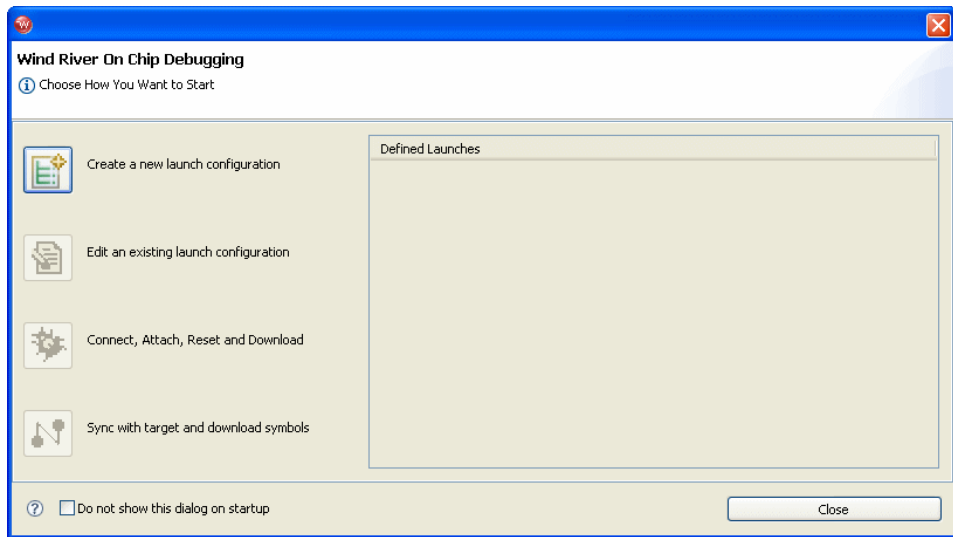
```
$ ./startWorkbench.sh
```

Windows Hosts

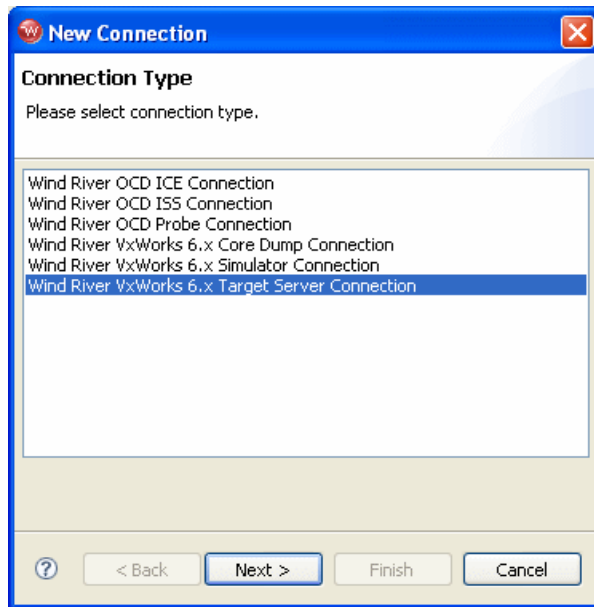
Select **Start > All Programs > Wind River > Wind River Workbench** *version*.

On Windows hosts, Workbench prompts you to specify a workspace location. Linux hosts use the default location *installDir/workspace*.

When Workbench opens, the **Quick Target Launch** dialog appears.



1. Select **Create a new launch configuration**.
The **Connection Type** dialog appears.

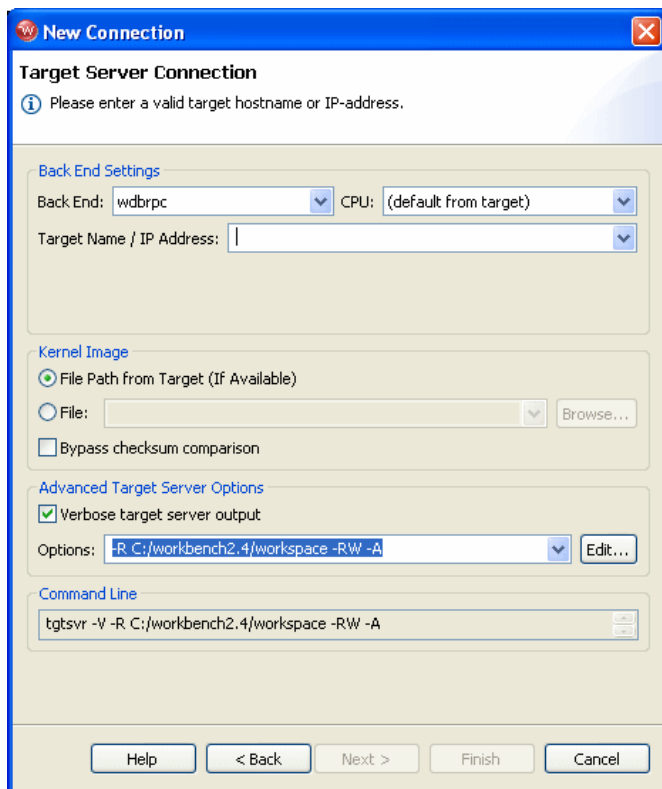


2. Choose **Wind River VxWorks 6.x Target Server Connection**.



NOTE: The Transparent Mode Driver is not currently supported for Linux.

3. Click **Next**.
The **Connection Settings** dialog appears.

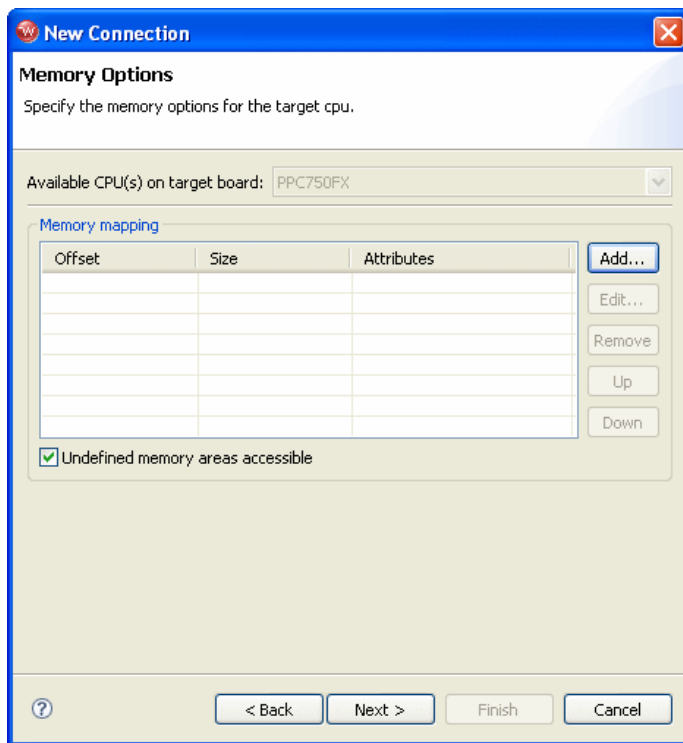


4. Set the **Back End** field to **wdbrpc**.
5. In the **Name/IP Address** field, enter the IP address of your Wind River ICE SX unit. (Make sure you use the IP address of the ICE, and not the IP address of your target.)

For information on assigning an IP address to your ICE unit, see the *Wind River ICE SX for Wind River Workbench Hardware Reference*.

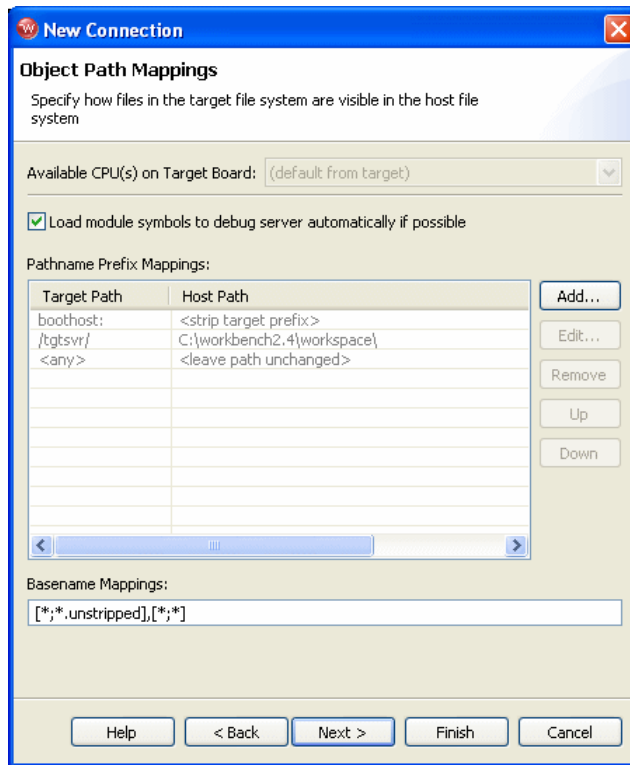
6. Click **Next**.

The **Memory Options** dialog appears.



The **Memory Options** dialog is only necessary for Linux or other non-VxWorks target operating systems, so leave the settings at their defaults and click **Next**.

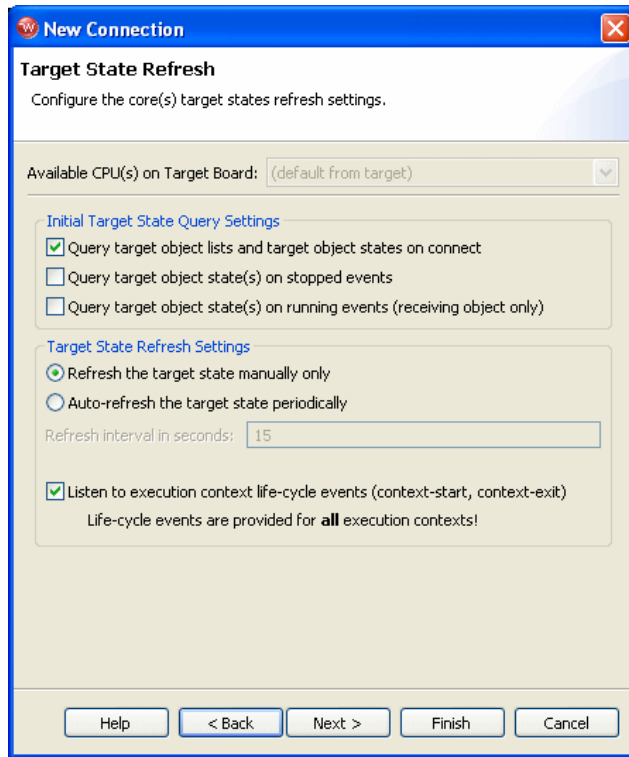
The **Object Path Mappings** dialog appears.



Use this dialog to make sure your pathnames are mapped correctly. To change the pathname mappings, use the **Add**, **Edit** and **Remove** buttons.

7. Click **Next**.

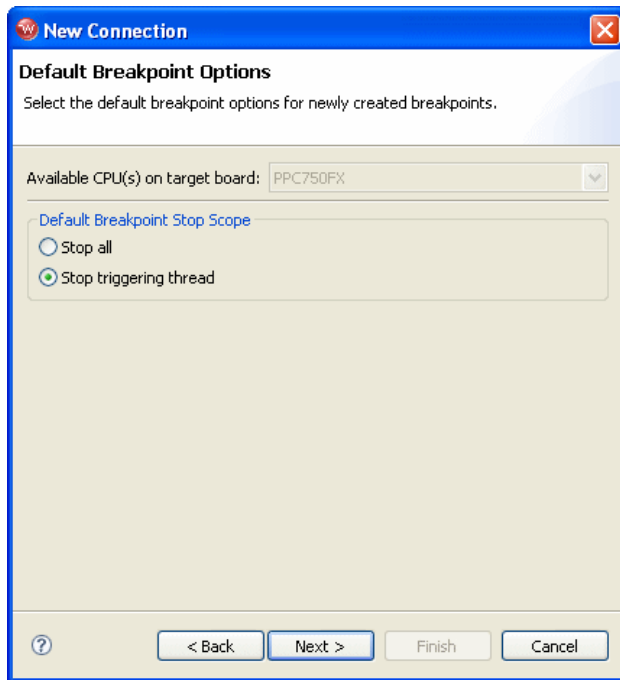
The **Target State Refresh** dialog appears.



8. Use this dialog to specify the target state query and target state refresh parameters.

9. Click **Next**.

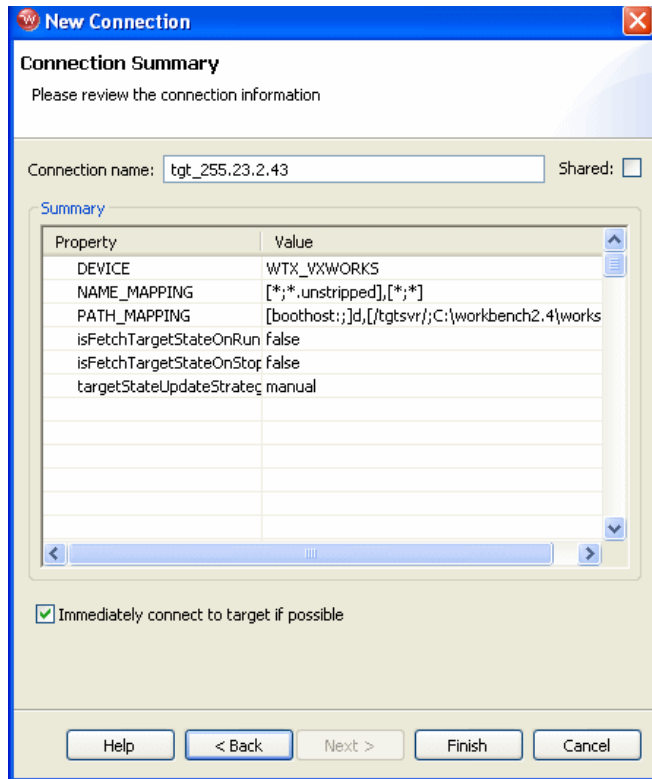
The **Default Breakpoint Options** dialog appears.



Use this dialog to set default breakpoint options for newly created breakpoints.

10. Click **Next**.

A connection summary appears.



11. Check the summary to be sure your settings are correct and click **Finish**.

You are returned to the **Device Debug** perspective. The target server connection is now visible as a target name in the **Target Manager** view.



NOTE: If you do not want to connect to your target now, uncheck the **Immediately connect to target if possible** box. You can connect to your target at any time by right-clicking on the target name in the **Target Manager** view and selecting **Connect**.

11.5 Moving On

Once the target server has been launched successfully, you can use any of the Wind River Workbench tools. Workbench operates in the same fashion regardless of whether the WDB connection is serial, Ethernet, or the Transparent Mode Driver. The only difference you may notice when the Transparent Mode Driver is in use is the speed, which is due to limitations on the BDM/JTAG/EJTAG connection type.

For information on Workbench tools, see the *Wind River Workbench User's Guide*.

12

Internal Software Trace

- 12.1 Overview 219
- 12.2 The Trace View 220
- 12.3 Configuring Trace 225
- 12.4 Tracing Execution 231

12.1 Overview

Internal software trace is only supported for PowerPC 85xx, PowerPC 86xx, and PA Semi PA6T-1682M processors. This chapter applies only to the PowerPC 85xx and 86xx and the PA Semi PA6T-1682M.

Internal trace captures a snapshot of your executing code to a memory array, at full speed. It saves up to hundreds of thousands of machine cycles, displaying addresses and instructions; PPC 85xx and 86xx also display transferred data. Wind River Workbench translates raw machine cycles to assembly code or C/C++ statements, and displays them in the **Trace** view.

Workbench captures only memory cycles, and may not reflect what the core actually executes, especially if cache is enabled.

Wind River Workbench starts and stops trace collection based on user-defined triggering mechanisms.

12.2 The Trace View

To open the **Trace** view, select **Window > Show View > Trace View**.

The **Trace** view appears, unpopulated.

The **Trace** view has two fields: the **Events** field and the **Trace** field.

The **Events** field shows the trace buffer. When code runs, the **Events** field shows the start of trace and the end of trace. It also displays the type of trace event.

For PPC 85xx and 86xx processors, the **Trace** field has five columns, from left to right: Event Occurrences (unlabeled), **Address**, **Abs Time**, **DEL Time**, and Instruction (unlabeled.) For PA Semi PA6T-1682M processors, the **Trace** field is the same except that it does not use the **Abs Time** and **DEL Time** columns.

The Event Occurrences column shows the type of trace event.

The **Address** column shows the address or line number of the trace event.

The **Abs Time** column shows the *absolute time*, that is, the elapsed time since the beginning of trace.

The **DEL Time** column shows the *delta time*, that is, the change in absolute time since the last trace entry.

The Instruction column shows the executed instructions. To set the code display, right-click in the Instruction field and select **Show Code Level**. From the list of options, select **Functions**, **Source**, or **Disassembly**.

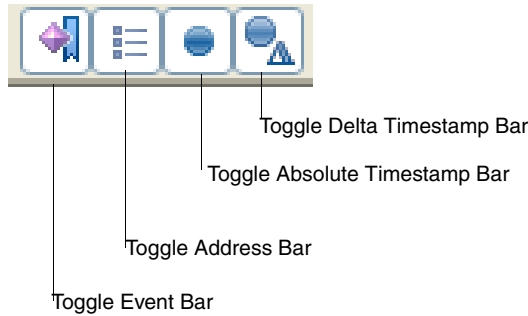
12.2.1 Trace View Buttons

The buttons in the **Trace** view have the following functions:

Collapsing and Expanding Fields

The first four columns in the **Trace** field (or the first two columns for PA Semi PA6T-1682M processors, which do not use the **Abs Time** and **DEL Time** columns) can be collapsed or expanded using the four toggle buttons in the **Trace** view, as shown in [Figure 12-1](#).

Figure 12-1 Trace Toggle Buttons



To collapse any column, click on the toggle button for that field. To re-expand it, click on the toggle button again.

The Instruction column cannot be collapsed.

Toggle Trace/Source view Auto-Sync

Click this button to set the Workbench editor to align itself with any highlighted instruction in the instruction field in the **Trace** view. With this button toggled, clicking on a function in the **Trace** view will cause the editor to jump to that function. To un-sync the **Trace** view and the editor, click the button again.

Clear Trace Buffer

Clear the trace buffer so that previously stored trace data is not included in the next trace that appears. Whenever you add new code, or manually alter the Program Counter value, you should clear the trace buffer before running or stepping, to prevent errors from occurring due to old trace data in the buffer. The button can also be used to trace individual functions by clearing the buffer and then stepping over the function.

The button does not actually flush the trace buffer; it just moves the pointer to the beginning of the buffer, so any previous data is overwritten.

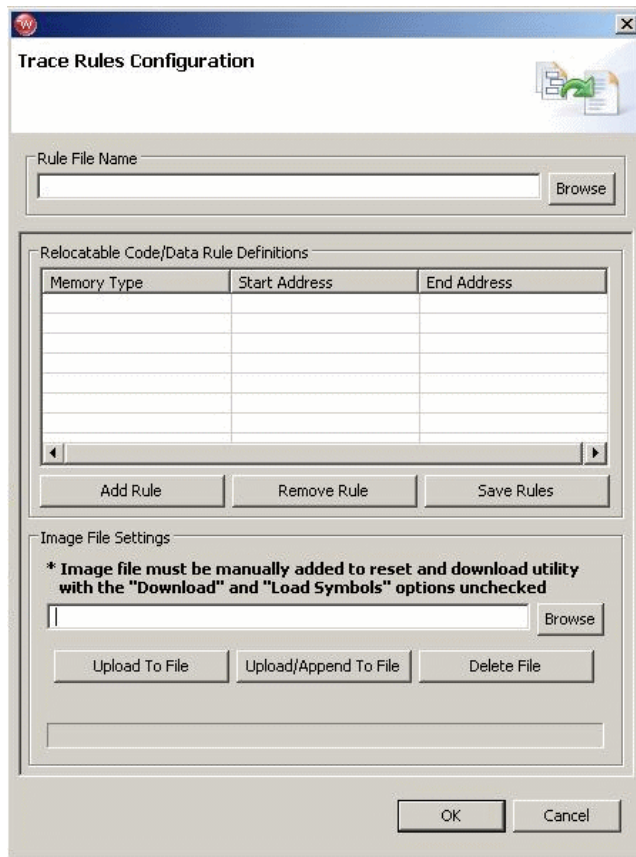
Refresh Trace View

Click this button to refresh the entire **Trace** view, including the **Events** field. Refresh the **Trace** view to display the newest good information.

Open Trace Rules Dialog

Click this button to open the **Trace Rules** dialog, as shown in [Figure 12-2](#).

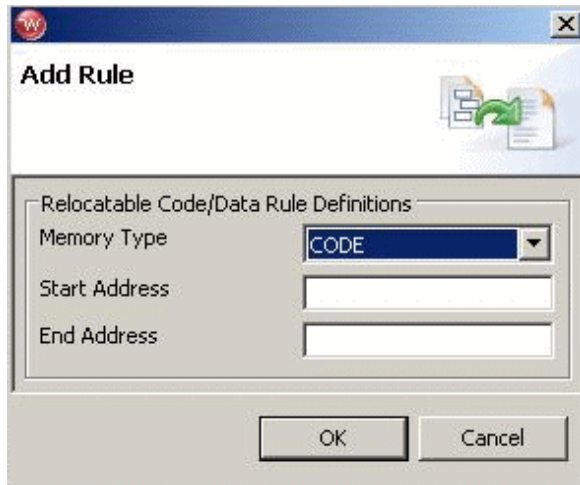
Figure 12-2 **Trace Rules Dialog**



Use the **Trace Rules** dialog to create a trace rules file, for cases where the code is not running in the address range of the download file specified in the **Reset and Download** view, such as an interrupt service routine in flash.

To create a trace rules file, use the following steps:

1. Click **Add Rule**.



2. In the **Memory Type** field, select **CODE** (for executable code) or **DATA**.
3. In the **Start Address** field, enter the memory address you want trace to begin.
4. In the **End Address** field, enter the memory address you want trace to end.
5. Click **Save Rules**.

In the browser window that opens, specify a location and name for your rules file. Rules files must be saved as ASCII files with the extension **.rulesconf**.

Next, use the ASCII file you have just created to generate a binary file using the following steps:

1. Click **Upload File**.
2. In the browser window that opens, specify a location and name for your binary file and click **Open**. Workbench will save the binary file with the extension **.elf**.

If you already have an existing **trace.elf** file, you can click **Upload/Append File** to append the new rule you just created to the end of the existing file.

3. Click **OK** to close the **Trace Rules** dialog.
4. In the **Target Manager** view, right-click on your target connection name and select **OCD Reset and Download**.
5. The **Reset and Download** view appears.
6. Select the **Files** tab.
7. Click **Add Files**.
8. In the browser window that appears, navigate to the **trace.elf** file you have just created and click **Open**.
9. The **trace.elf** file appears in the file list.
10. Uncheck the **Download** and **Load Symbols** fields. Leave the **Verify** field set to **None**.
11. Click **Debug**.

The trace rules are now added to your project.

Filter Visible Trace Events

Filtered trace is not supported for internal software trace.

Save Output to File

This button opens a browser window. Use the browser to specify a file to which you can save the information in the **Trace** view.

This button saves the information from all columns in the **Trace** field (three or five, depending on which processor family you are connected to). It does not save information from the **Events** field.

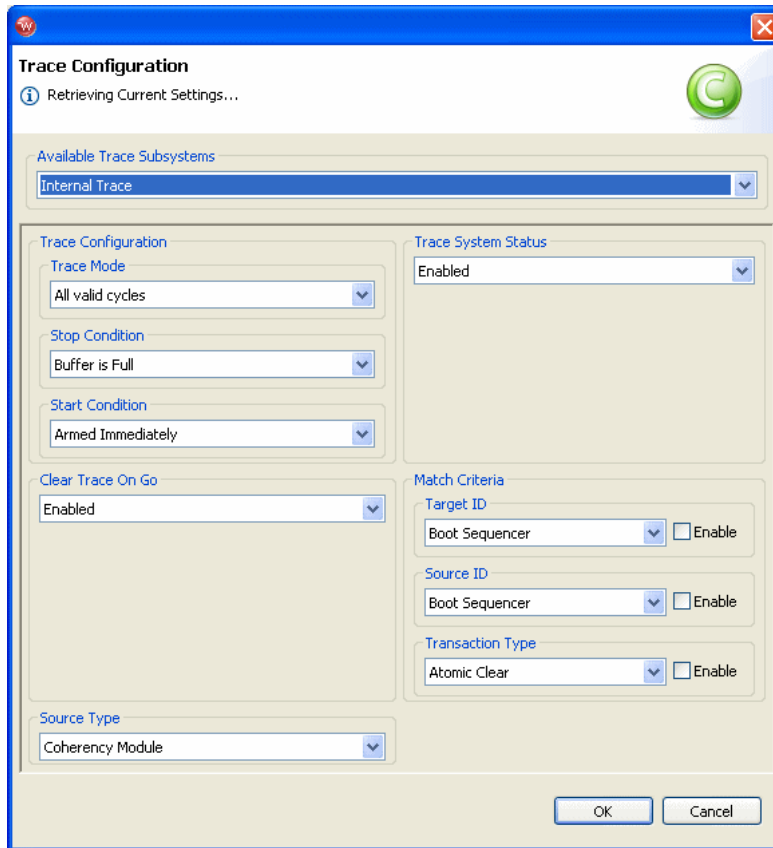
12.3 Configuring Trace

To configure trace-specific configuration options, click the **Configure Trace** button. The **Configure Trace** dialog opens, showing the available options for your target processor.

12.3.1 PowerPC Trace Configuration Options

If you are connected to a PPC 85xx or 86xx processor, the **Configure Trace** dialog opens showing the following options.

Figure 12-3 Internal Trace Options for PowerPC



Trace Configuration

Trace Mode

Select the Trace Mode to determine whether to trace every valid cycle, or only trace when the event is matched:

- **All Valid Cycles** -- Capture any valid bus cycle regardless of what the Match Criteria is set to.
- **Trace Event is detected** -- Capture only those cycles that match the Match Criteria.

If you set up any of the fields in the Match Criteria, use **Trace Event is detected**.

Start Condition

Select the start condition to determine when to start the Trace Buffer capture:

- **Arm Immediately** -- This option starts trace captures as soon as the target starts.
- **Watchpoint Monitor event is detected** -- Start trace capture only when a Watchpoint Monitor event occurs.
- **Trace Buffer Event is detected** -- Start trace capture only when a Trace Buffer Event occurs.
- **Performance Monitor signal overflow** -- Start trace capture when performance monitors overflow (an internal signal indicates that a counter overflow has happened).
- **TRIG_IN transitions from 0 to 1** -- Start capture when TRIG_IN signal goes from 0 to 1.
- **TRIG_IN transitions from 1 to 0** -- Start capture when TRIG_IN signal goes from 1 to 0.
- **Current context ID = Programmed context ID** -- Start trace capture when the programmed context ID register is equal to the current context ID.
- **Current context ID != Programmed context ID** -- Start trace capture when the programmed context ID register is not equal to the current context ID.

Stop Condition

Select a Stop Condition to determine when to stop capturing trace.

- **Trace Buffer is Full** -- Stop trace capture when it reaches the end of the buffer.

- **Watchpoint monitor event is detected** -- Stop capturing any trace when the Watchpoint monitor event is matched.
- **Trace Buffer Event is detected** -- Stop capturing trace when a trace buffer event is matched.
- **Performance Monitor signal overflow** -- Stop capturing trace when performance monitors overflow (an internal signal indicates that a counter overflow has happened).
- **TRIG_IN transitions from 0 to 1** -- Stop capture when TRIG_IN signal goes from 0 to 1.
- **TRIG_IN transitions from 1 to 0** -- Stop capture when TRIG_IN signal goes from 1 to 0.
- **Current context ID = Programmed context ID** -- Stop trace capture when the programmed context ID register is equal to the current context ID.
- **Current context ID != Programmed context ID** -- Stop trace capture when the programmed context ID register is not equal to the current context ID.

Clear Trace On GO

Use this parameter to control where to start saving trace data in the trace memory on a GO command. The trace clear settings YES and NO determine where to start saving the trace data in the trace memory, as explained below.

- **YES** — When a GO command is issued, the trace data will be stored in trace memory starting at the first trace memory location. All previously stored trace data will be overwritten and lost. All newly captured trace data will be stored starting at the beginning of the trace memory.
- **NO** — When GO command is issued, the trace data will be stored in trace memory starting at the next trace memory location. All previously stored trace data will not be overwritten. All newly captured trace data will be stored starting at the next trace memory location.

Source Type

- **Coherency Module** -- This is the default trace source after the processor reset. It provides all the activities on the local bus and the snoop cycles between the core and the L2 cache.
- **DDR SDRAM** -- This is similar to the Coherency Module, except without snoop cycles.
- **PCI** -- Select the trace source of the PCI/PCI-X output interface.

- **Rapid IO** -- Select the internal Rapid I/O outbound interface for the trace source.

Trace System Status

Use this parameter to control the acquiring of trace data to the trace memory on a **GO** command. The trace data will also be acquired when stepping, running to a PC value, or running back to a calling function. The trace acquire settings **ENABLE** and **DISABLE** determine when the trace memory will acquire trace data on a **GO** command, as follows:

- **ENABLE** — When a **GO** command is issued, all trace data will be acquired and saved in the trace memory.
- **DISABLE** — When a **GO** command is issued, no trace data will be acquired and saved in the trace memory.

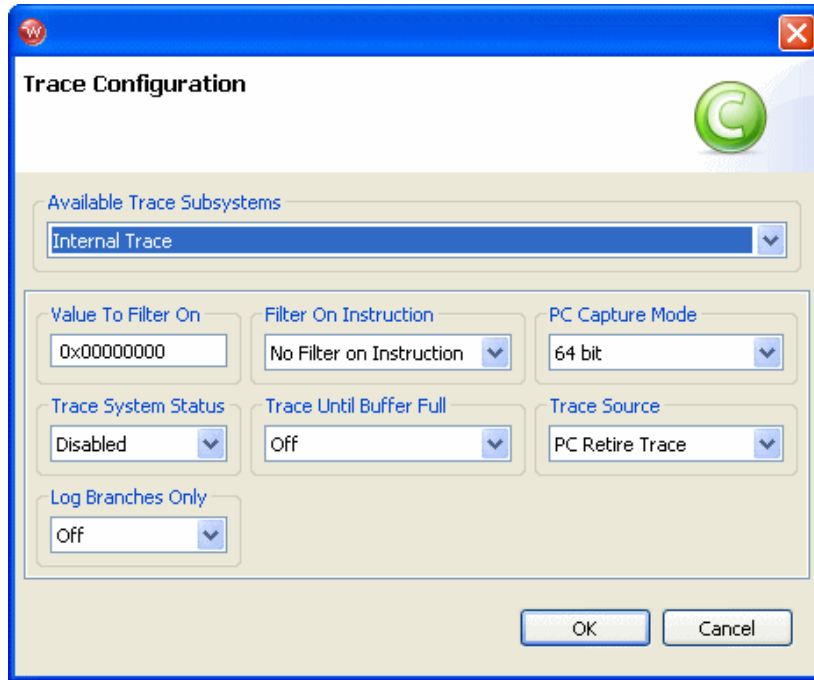
Match Criteria

- **Address with Mask** -- This option will filter the Trace Buffer by matching only the correct address with the address mask. For an instruction fetch bus cycle, the address will appear in the form 0x0, 0x20, 0x40 and the memory variable address will show up as it is. In order to capture the correct address (with mask), an instruction fetch address has to be entered as 0x0, 0x20, 0x40 and the memory cycle's address can be entered without any adjustment.
- **Transaction Type** -- This option will filter trace capture by matching only the correct transaction type. The transaction types are different when the trace source changes. A different set of drop down menu will appear when the trace source changes.
- **Source ID** -- Selecting the Source ID field will capture only the matched cycle with the same Source ID. This ID indicates the source of this cycle.
- **Target ID** -- Selecting the Target ID field will capture only the matched cycle with the same Target ID. This ID indicates the destination of this cycle.

12.3.2 PA Semi Trace Configuration

If you are connected to a PA Semi PA6T-1682M processor, the **Configure Trace** dialog opens showing the following options.

Figure 12-4 Internal Trace Configuration for PA Semi



12

Available Trace Subsystems

The only available trace subsystem is Internal Trace.

Filter on Execution

Use this field to set Workbench to enable or disable trace when a particular instruction is executed. By default this field is set to **No Filter on Instruction**. If you set it to **Enable on Instruction** or **Disable on Instruction**, you must enter the address of the instruction in the **Value to Filter On** field.

Value to Filter On

If you have set the **Filter on Execution** field to **No Filter on Instruction**, ignore this field. If you have set the **Filter on Execution** field to **Enable on Instruction** or **Disable on Instruction**, use this field to specify the address of the instruction you want to use.

PC Capture Mode

Use this field to specify whether Workbench should output captured trace as 32-bit or 64-bit values.

Trace System Status

Use this field to control the acquiring of trace data to the trace memory on a **GO** command. The trace data will also be acquired when stepping, running to a PC value, or running back to a calling function. The trace acquire settings **Enabled** and **Disabled** determine when the trace memory will acquire trace data on a **GO** command, as follows:

- **Enabled** — When a **GO** command is issued, all trace data will be acquired and saved in the trace memory.
- **Disabled** — When a **GO** command is issued, no trace data will be acquired and saved in the trace memory.

Trace Until Buffer Full

Set this option to **On** to set Workbench to stop capturing trace when the trace buffer is full. If this option is set to **Off**, Workbench continues to capture trace when the trace buffer is full, overwriting the contents of the trace buffer with the newly captured trace.

Trace Source

- **PC Retire Trace** - Select the PCI/PCI-X output interface for the trace source.
- **Connexium Bus Trace** - Select the Pa Semi Connexium bus for the trace source
- **Rapid IO Trace** - Select the internal Rapid I/O outbound interface for the trace source.

Log Branches Only

When this option is set to **Off** (the default) the processor outputs all instructions. When this option is set to **On**, the processor outputs only on branch instructions.

12.4 Tracing Execution

You must have downloaded your code to Workbench, either by using the Workbench project management facility or by using the **Reset and Download** view, before you can begin to trace code.

12.4.1 Setting a Tracepoint

Next, set a tracepoint in your code.

If no tracepoints are set, The trace will contain all code up to the point where the target was suspended, either manually or by hitting a breakpoint.

To set a tracepoint, right-click to the left of the editor (in the gutter) and select **Tracepoints > Add Tracepoint**.

The **Line Tracepoint** dialog appears. The options shown in the **Line Tracepoint** dialog (**After Trace Counter**, **Post Trigger Counter**, and so on) are not supported for internal trace.

12

12.4.2 Tracing Execution

Having specified your tracepoint in the **Line Tracepoint** dialog, click **OK**.

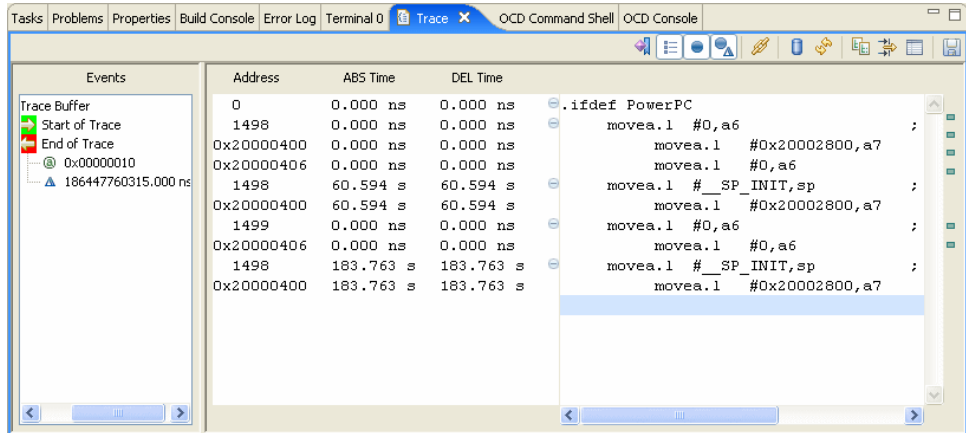
Workbench sets the tracepoint in your code, placing a trace icon in the editor. An entry for the tracepoint appears in the **Breakpoints** view.

In the **Debug** view, click **Resume**. Let some code execute and then click **Suspend**.

In the **Trace** view, click **Refresh View**.

Trace information appears in the **Trace** view, as shown in [Figure 12-5](#).

Figure 12-5 Trace View



(In Figure 12-5, the Event Occurrences column is collapsed to make it easier to read the information in the Instruction field.)

13

Using the CF Options View

- 13.1 Introduction 233
- 13.2 Connecting to a Target 234
- 13.3 Configuring the Target Connection 238
- 13.4 Changing CF Options in the CF Options View 240
- 13.5 Changing CF Options With Low-Level Commands 241
- 13.6 Resetting CF Options 242

13.1 Introduction

Wind River emulators can be configured in several different ways to specify various settings such as electrical properties, connection logic, and clock rate. To configure these settings Workbench uses *configuration options*, or CF options, which you can set in the **CF Options** view.

This chapter provides a tutorial for configuring a target connection using CF options.

What CF options are available depends on the target processor, and also on whether you connect with a Wind River Probe or Wind River ICE SX. For a full description of all Wind River CF options sorted by processor family, see the *Wind River Workbench for On-Chip Debugging Configuration Options Reference*.

13.2 Connecting to a Target

In order to configure CF options, you must have an active target connection.

This tutorial uses a Wind River Probe emulator connected to a Wind River PPMC750FX target.

To connect to your target, use the following steps:

1. Launch Wind River Workbench according to the method for your host.

Linux/Solaris Hosts

From your installation directory, issue the command

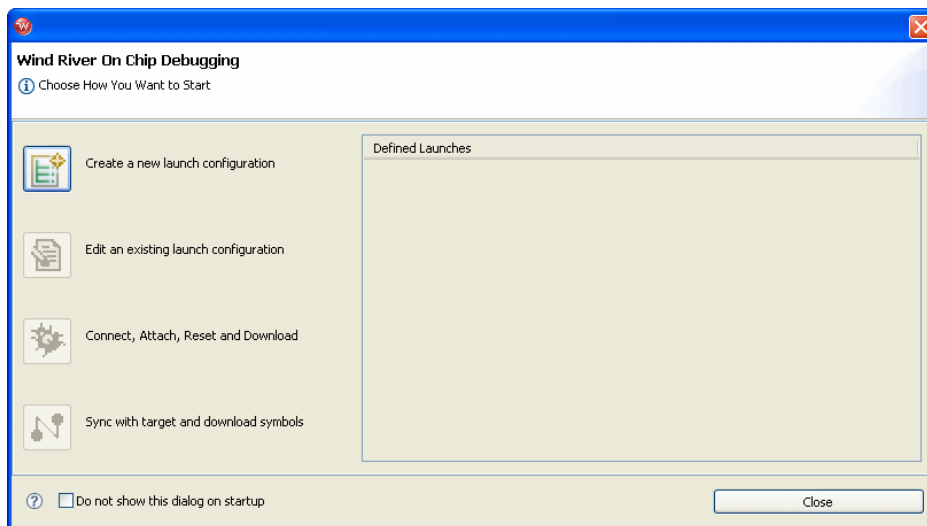
```
$ ./startWorkbench.sh
```

Windows Hosts

Select **Start > All Programs > Wind River > Wind River Workbench version**.

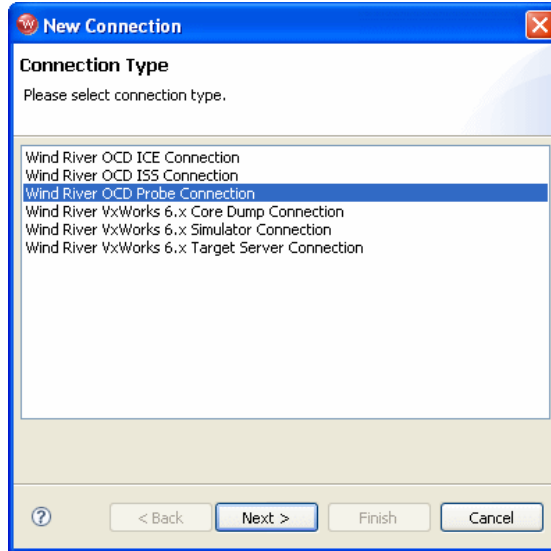
On Windows hosts, Workbench prompts you to specify a workspace location. Linux/Solaris hosts use the default location *installDir/workspace*.

When Workbench opens, the **Quick Target Launch** dialog appears.



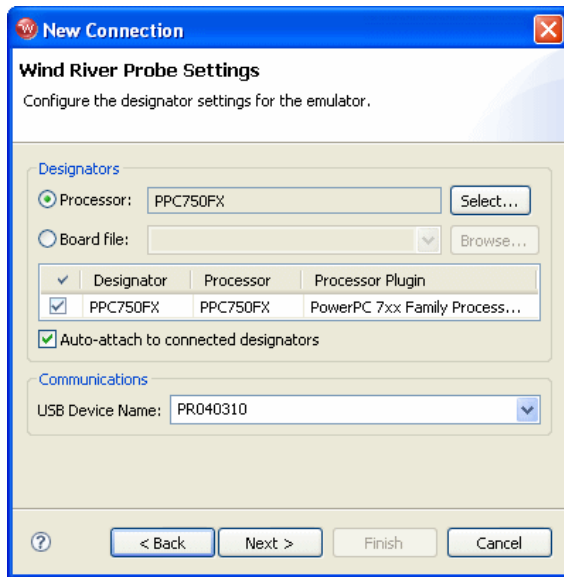
2. Select **Create a new launch Configuration**.

The **Connection Type** dialog appears.

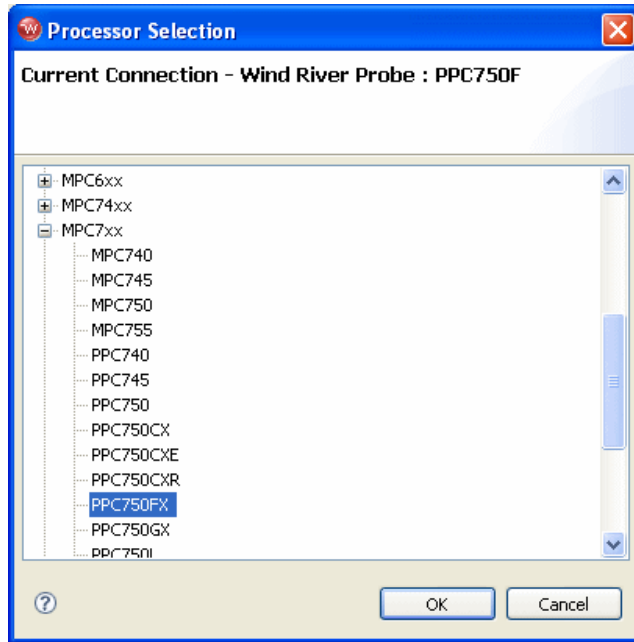


3. Select **Wind River OCD Probe Connection** and click **Next**.

The **Processor Selection** dialog appears.



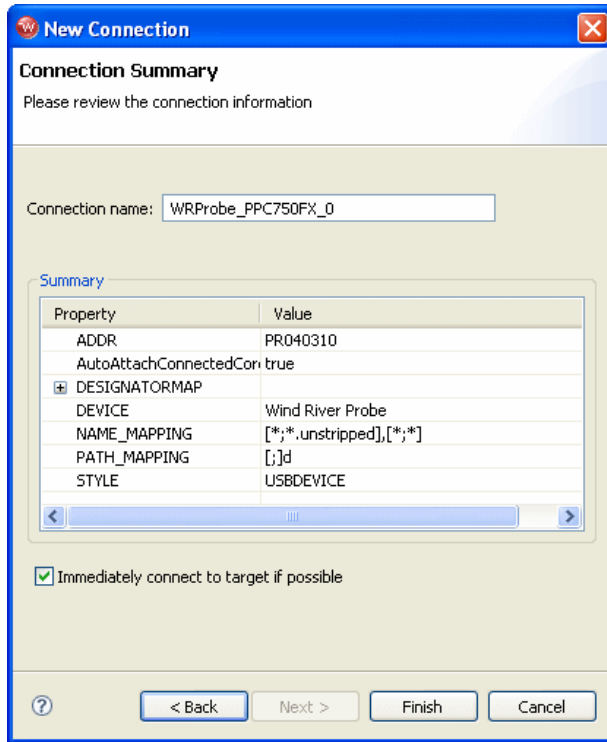
4. Click **Select**. From the list that appears, expand **MPC7xx** and select **PPC750FX**.



5. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

6. Click **Next**.
7. The connection wizard passes through a number of screens that you do not need to configure for this tutorial. Leave all settings at their defaults and click **Next** until you come to the **Connection Summary**.



8. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

Workbench creates a target connection called **WRProbe_PPC750FX** in the **Target Manager** view and opens the **Reset and Download** view.

9. You do not need to download code to configure the target connection, so click **Close** to close the **Reset and Download** view.

13.3 Configuring the Target Connection

In the Workbench toolbar, select **Window > Show View > CF Options**.

Command Name	Current Setting	Parameters	Description
SB	SB	[SB, IHBC]	Set BreakPoint
VECTOR	LOW	[HIGH, LOW, IGNORE]	Vector Table Location
RST	YES	[YES, NO, HALT, RUN]	Monitor Target reset
TAR	750FX	[AUTO, 603E, EC603E, 603...	Target CPU
SLAVE	NONE	[NONE, 8260]	Target CPU(SLAVE)
SLIMMRVAL	AUTO	[AUTO, VALUE]	Slave IMMR reset value
RTP	NO	[YES, NO]	Real time Preservation
LENDIAN	NO	[YES, NO]	Little Endian Mode
MODE	64	[32, 64]	Processor Mode
DLD	NORMAL	[NORMAL, 8]	Download Mode
HRESET	ENABLE	[ENABLE, DISABLE]	Emulator HRESET Control
PAR	NO	[YES, NO]	Data Parity Checking
TGTCONS	BDM	[BDM, COM1, COM2]	Target Console Redirection
TRESET	ACTIVE	[OPENC, ACTIVE]	Drive TReset line
INWCI	YES	[YES, NO]	Invalidate Instruction Cach...
SPOWER	YES	[YES, NO]	Sense Power via HRESET
RESET	HRESET	[HRESET, SRESET, HRESET...	CPU Reset Type
TRPEXP	YES	[YES, NO, SOI, BREAKPOIN...	Trap exception
INCOLD	NO	[YES, NO]	Issue an IN on coldstart
L2WARNING	NO	[YES, NO]	Display L2 Data Cache War...
MMU	DISABLE	[ENABLE, DISABLE]	Memory Management Unit ...
BL	DISABLE	[ENABLE, DISABLE]	Load Boot Table On IN
BRKREP	BRKREP	[REONLY, BRKREP]	Trigger In Report Mode
TMD	DISABLE	[ENABLE, DISABLE]	TMD Mode
CLK	16	[0.025...100,AUTO]	JTAG clock rate (MHz)

The **CF Options** view opens, populated with the available CF options for your emulator and target processor.

The **CF Options** view has four columns: **Command Name**, **Current Setting**, **Parameters**, and **Description**. An example entry is shown below:

Command Name	Current Setting	Parameters	Description
INCOLD	NO	[YES, NO]	Issue an IN on coldstart

Command Name shows the argument associated with this option for the low-level CF command in the **OCD Command Shell**.

Current Settings shows the value to which the option is currently set.

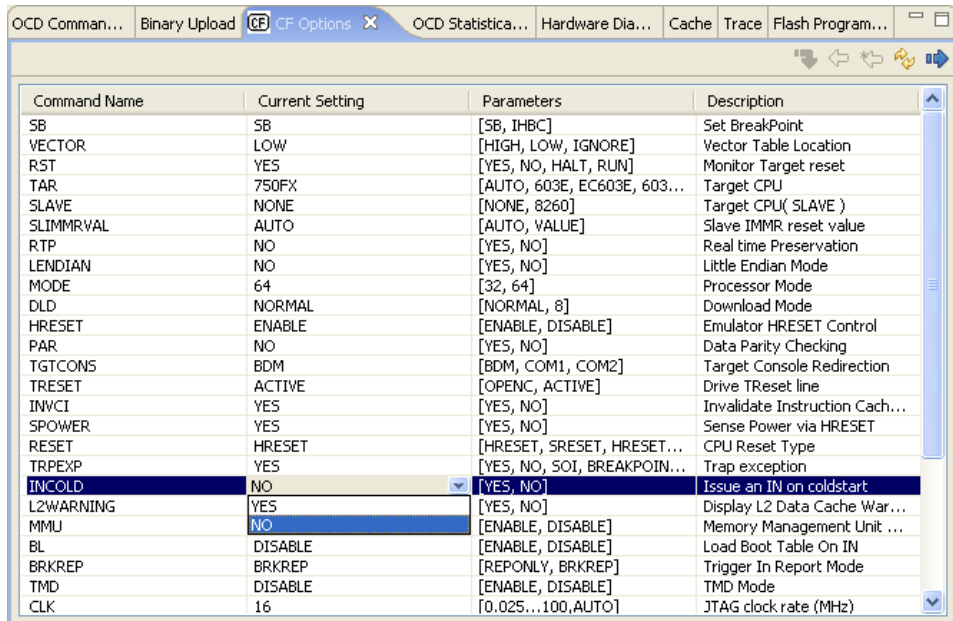
Parameters shows the available range of values to which you can set the option.

Description shows a description of the configuration option.

13.4 Changing CF Options in the CF Options View

To change the value of a CF option in the **CF Options** view, use the following steps.

1. Highlight any CF option and click on its entry in the **Current Settings** column.
A drop-down list appears.



2. In the drop-down list, choose the value you want and click on it to reset the CF option.

For example, suppose you want the emulator to send an **IN** initialization command to the target on every cold start. By default this option is set to **NO**.

- a. In the **Description** column, find **Issue an IN on coldstart**.
 - b. Highlight that CF option and click the **Current Settings** column.
 - c. A drop-down list of available values (in this case **NO** and **YES**) appears.
 - d. Select **YES** to enable the option.
3. Repeat these steps for any CF options you wish to change.

- After you have set all the CF options you want to set, click the **Send All CF Options to Target** button.



NOTE: Setting CF options in the CF options view does not immediately make changes to the target connection. Your changes do not take effect until you issue a target reset, either by using the **Send All CF Options to Target** button in the CF Options view or by issuing an IN or INN command in the **OCD Command Shell**.

13.5 Changing CF Options With Low-Level Commands

You can also work with configuration options by using the CF command in the **OCD Command Shell**. At a >BKM> or >ERR> prompt, enter the command CF. This will bring up a list of your emulator's configuration options.

```
>BKM>cf
Set BreakPoint                               SB[SB, IHBC] = SB
Vector Table Location                         VECTOR[HIGH, LOW, IGNORE] = LOW
Monitor Target reset                          RST[YES, NO, HALT, RUN] = YES
Target CPU                                    TAR[AUTO, 603E, EC603E, 603P, 603R, 740, 745,
750, 750CX, 750CXE, 750FX, 750GX, 755, 7400, 7410] = 750FX
Target CPU( SLAVE )                           SLAVE[NONE, 8260] = NONE
Slave IMMR reset value                       SLIMMRVAL[AUTO, VALUE] = AUTO
JTAG clock rate (MHz)                        CLK[0.025...100, AUTO] = 16
Application IMMR Exclusion Range              AIMMRER[OFF, START and END] = OFF
Application IMMR Value                       AIMMRVAL[VALUE] = 0e000000
Real time Preservation                       RTP[YES, NO] = NO
Little Endian Mode                           LENDIAN[YES, NO] = NO
Processor Mode                               MODE[32, 64] = 64
Download Mode                               DLD[NORMAL, 8] = NORMAL
Emulator HRESET Control                     HRESET[ENABLE, DISABLE] = ENABLE
Data Parity Checking                         PAR[YES, NO] = NO
Set Work Space                               WSPACE[BASE and SIZE] = 00000000 174c
Set Stack Range                             STACK[OFF / LOWER and UPPER] = OFF
Target Console Redirection                  TGTCONS[BDM, COM1, COM2] = BDM
Drive TReset line                           TRESET[OPENC, ACTIVE] = ACTIVE
Invalidate Instruction Cache on GO          INVCI[YES, NO] = YES
Reset Pulse Length N*1ms                    RPL[1..2000] = 1
Sense Power via HRESET                      SPOWER[YES, NO] = YES
Power On Reset Length N*1ms                 PONR[0..500] = 0
CPU Reset Type RESET[HRESET, SRESET, HRESET_UNFILTER, SRESET_UNFILTER] = HRESET
Trap exception                              TRPEXP[YES, NO, SOI, BREAKPOINTONLY] = YES
Issue an IN on coldstart                    INCOLD[YES, NO] = NO
Display L2 Data Cache Warning               L2WARNING[YES, NO] = NO
Memory Management Unit Mode                MMU[ENABLE, DISABLE] = DISABLE
Load Boot Table On IN                       BL[ENABLE, DISABLE] = DISABLE
Trigger In Report Mode                      BRKREP[REONLY, BRKREP] = BRKREP
```

```
TMD Mode                                TMD[ENABLE,DISABLE] = DISABLE
Run Counter Length                      RCL[1000..FFFF] = 1000
Delay after Reset Nms                   DRST[0..10000] = 25
>BKM>
```

Change any CF option using the syntax

CF *CommandName Value*

CommandName is the name given in the **Command Name** column in the **CF Options** view.

Value is the value you wish to change to.

For example, to set the emulator to send an IN initialization command to the target on every cold start, enter the command

```
>BKM>cf incoldd yes
```

Enter the CF command again to see your changes.



NOTE: Setting CF options with the CF command does not immediately make changes to the target connection. Your changes do not take effect until you issue a target reset, either by using the **Send All CF Options to Target** button in the **CF Options** view or by issuing an IN or INN command in the **OCD Command Shell**.

13.6 Resetting CF Options

To restore all CF options to their target defaults, use the **Reset to default target settings** button in the CF Options view.

14

Using Hardware Diagnostics

- 14.1 Introduction 243
- 14.2 Connecting to Your Target 244
- 14.3 Setting a Workspace 248
- 14.4 Hardware Diagnostic Tests 249

14.1 Introduction

The **Hardware Diagnostic** view provides a set of RAM and bus diagnostics and utilities that can be controlled by the emulator or run on the target. For some of the tests, you can run code directly on the target instead of through the emulator by selecting the **Run on Target** checkbox. This allows the test to run at the execution speed of the target processor.

This tutorial uses a Wind River Probe emulator connected to a Wind River PPMC750FX target.

14.2 Connecting to Your Target

To connect to your target, use the following steps:

1. Launch Wind River Workbench according to the method for your host.

Linux/Solaris Hosts

From your installation directory, issue the command

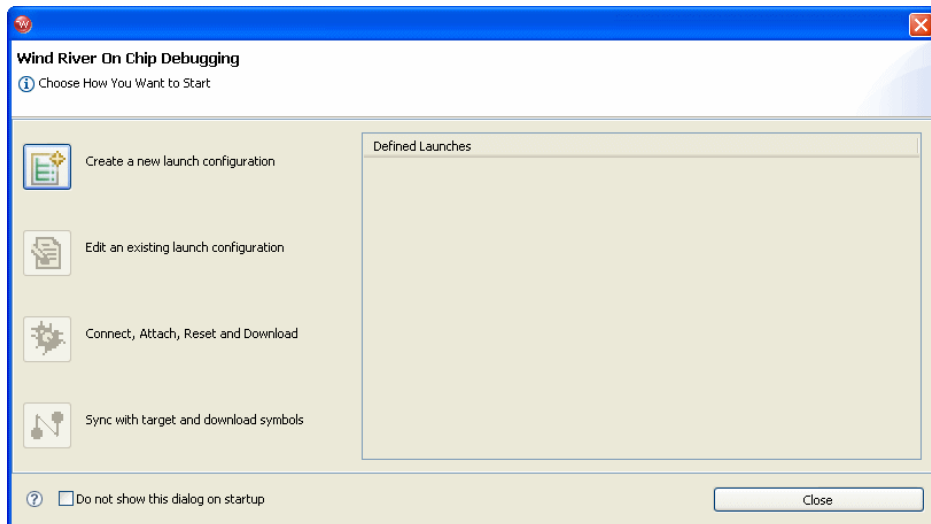
```
$ ./startWorkbench.sh
```

Windows Hosts

Select **Start > All Programs > Wind River > Wind River Workbench version**.

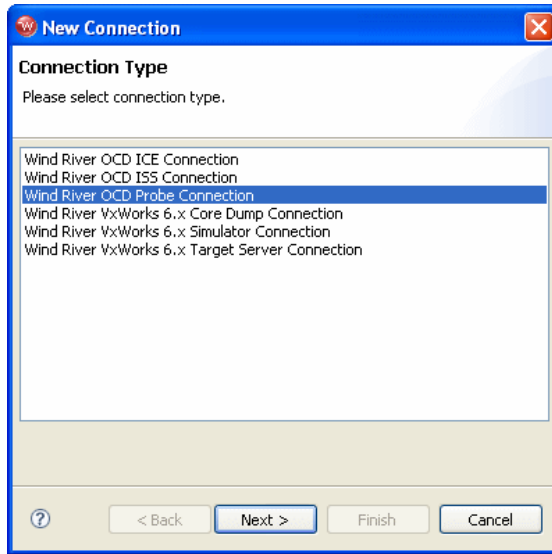
On Windows hosts, Workbench prompts you to specify a workspace location. Linux/Solaris hosts use the default location *installDir/workspace*.

The **Quick Target Launch** dialog appears.



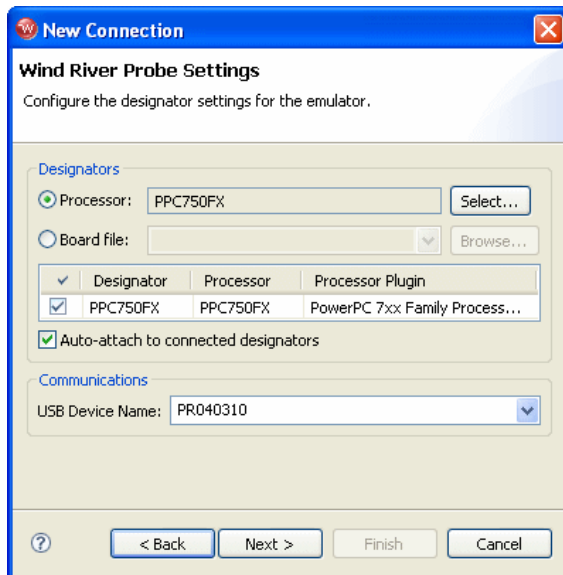
2. In the **Target Manager** view, right-click **default(localhost)** and select **New > Connection**.

The **Connection Type** dialog appears.

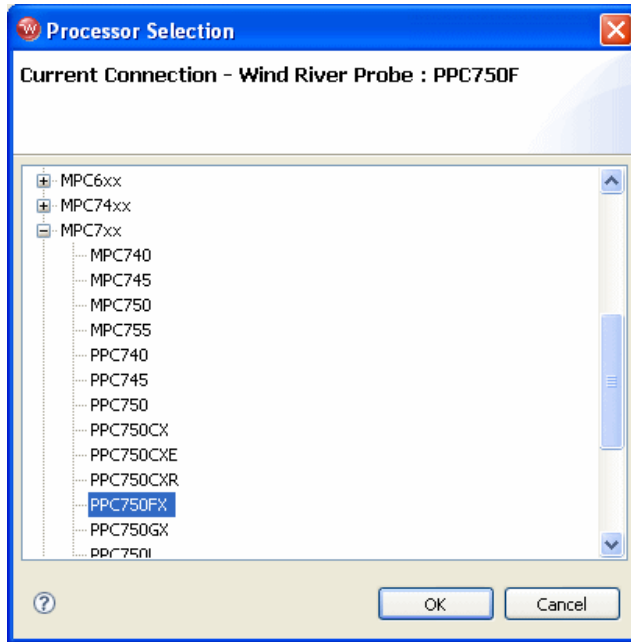


3. Select **Wind River OCD Probe Connection** and click **Next**.
The **Processor Selection** dialog appears.

14



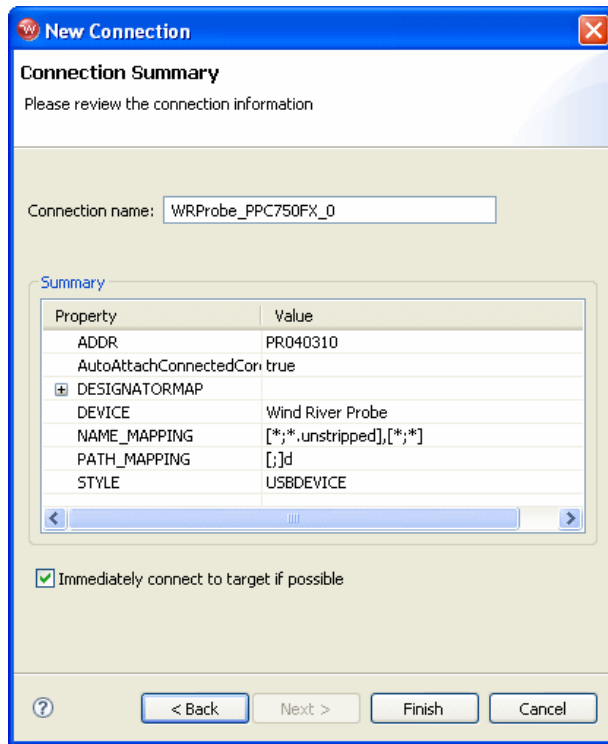
4. Click **Select**. From the list that appears, expand **MPC7xx** and select **PPC750FX**.



5. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

6. Click **Next**.
7. The connection wizard passes through a number of screens that you do not need to configure for this tutorial. Leave all settings at their defaults and click **Next** until you come to the **Connection Summary**.



8. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

Workbench creates a target connection called **WRProbe_PPC750FX** in the **Target Manager** view and opens the **Reset and Download** view.

9. You do not need to download code in order to run hardware diagnostics, so click **Close** to close the **Reset and Download** view.

14.3 Setting a Workspace

→ **NOTE:** The RAM workspace has no relation to the workspace that Workbench uses to store project information.

The RAM workspace is an area of RAM on the target that the emulator uses to download the hardware diagnostic routines and flash programming algorithms. You must tell your emulator where writable RAM is located on your target for this purpose.

→ **NOTE:** Setting a RAM workspace is only necessary if you are running the diagnostics on the target. If you do not select the **Run on Target** checkbox, you do not need to set a RAM workspace. Tests run on the target are slower, so if you select the **Run on Target** checkbox, make sure you specify a small area of memory to be tested.

Depending on the device family and type, this space is limited to under 2 KB. Note that more memory improves the speed of programming.

To configure the workspace, enter the parameters in the **OCD Command Shell**, using the syntax

CF WSPACE *base size*

where *base* is the start address, and *size* is the minimum number of bytes of target RAM required.

To find the *base* and *size* values for a Wind River-supported target, consult your target's **target.ref** file, located in *installDir/vxworks-6.x/target/config/yourTarget*. Alternatively, consult your processor documentation.

For example, on a Wind River PPC750FX target, the base of the workspace is 00000000 and the size is 1770. To set the workspace, in the **OCD Command Shell** enter the command

```
>BKM>cf wspace 0 1770
```

This sets the workspace at address 0 with a size of 1770 bytes.

14.4 Hardware Diagnostic Tests

To run diagnostic tests on your target, use the following steps.

14.4.1 Simple RAM Test

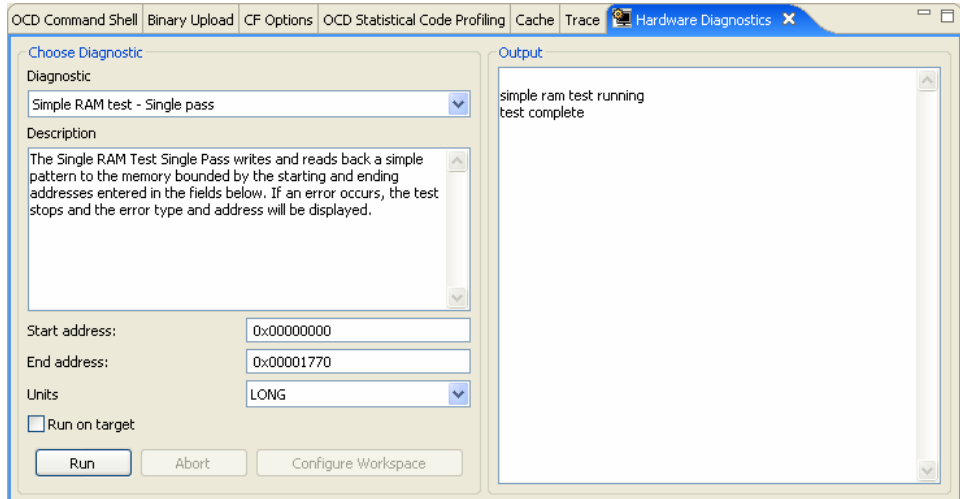
This test writes and reads back a simple pattern to the memory bounded by the starting and ending addresses entered in the **Start Address** and **End Address** fields. If an error occurs, the test stops and the error type and address are displayed in the **Output** field.

The first diagnostic to be run is a Simple Ram Test on the area of memory used by the workspace.

1. In the Workbench toolbar, select **Window > Show View > Hardware Diagnostics**.
2. In the **Diagnostic** field, select **Simple RAM Test – Single Pass**.
3. The workspace cannot be used to test itself, so make sure the **Run on target** checkbox is unchecked.
4. In the **Start Address** field, enter **0**.
5. In the **End Address** field, enter **1770**.
6. In the **Units** field, select **LONG**.
7. Click **Run**.

Workbench displays the test result in the **Output** field. The output of a successful test will resemble that in [Figure 14-1](#).

Figure 14-1 Successful Simple RAM Test



If the test fails, the **Address Bus Test** diagnostic and the **Data Bus Test** diagnostic may determine the cause of the failure; see [14.4.5 Bus Tests](#), p.253.

If the RAM test of the memory used by the workspace passed, you can now test the rest of the memory in the target system at full bus speed.

1. In the **Diagnostic** field, select **Simple RAM Test – Single Pass**.
2. Select the **Run on Target** checkbox.
3. In the **Start Address** field, enter **14000**.
4. In the **End Address** field, enter **20000000**.
5. In the **Units** field, select **LONG**.
6. Click **Run**.

Workbench displays the test result in the **Output** field.

If the message **Test Complete** appears, then the diagnostic passed.

If the test fails, try re-seating the SDRAM module and repeat the test. If the test still fails, then run the **Address Bus Test** diagnostic and the **Data Bus Test** diagnostic to determine the cause of the failure. See [14.4.5 Bus Tests](#), p.253.

14.4.2 Full RAM Tests

A Full RAM test writes a “walking” 1 on each bit of RAM and reads it back. This is a very lengthy test and can detect bus configuration errors, typically on a new printed circuit board.

This test sets and then clears each bit to try to locate memory defects bounded by the starting and ending addresses entered in the **Start Address** and **End Address** fields. If an error occurs, the test stops and the error type and address are displayed in the **Output** field.



NOTE: A complete Full RAM test would take several years to finish, so make sure you specify a very small region of memory to be tested.

Full RAM tests are designed to check for cell disturbance and addressing problems. These tests perform the following actions:

A **Single Pass** test will run the test only once. A **Continuous** test will repeat the test over the same address until you click **Stop**.

1. In the **Diagnostic** field, select **Full RAM Test – Single Pass**.
2. Select the **Run on Target** checkbox.
3. In the **Start Address** field, enter **14000**.
4. In the **End Address** field, enter **00014100**.
5. In the **Units** field, select **LONG**.
6. Click **Run**.

Workbench displays the test result in the **Output** field.

If the message **Test Complete** appears, then the diagnostics passed.

If the test fails, try re-seating the SDRAM module and repeat the test. If the test still fails, then run the **Address Bus Test** diagnostic and the **Data Bus Test** diagnostic to determine the cause of the failure. See [14.4.5 Bus Tests](#), p.253.

14.4.3 CRC Calculation

Workbench and the emulator support the calculation of a Cyclic Redundancy Check (CRC) on all addresses in the range specified. The CRC test will checksum a block of data on the target for the address range you specify in the **CRC Calculation** dialog. The CRC algorithm is based on the following polynomial:

$$x^{16} + x^{15} + x^2 + 1$$

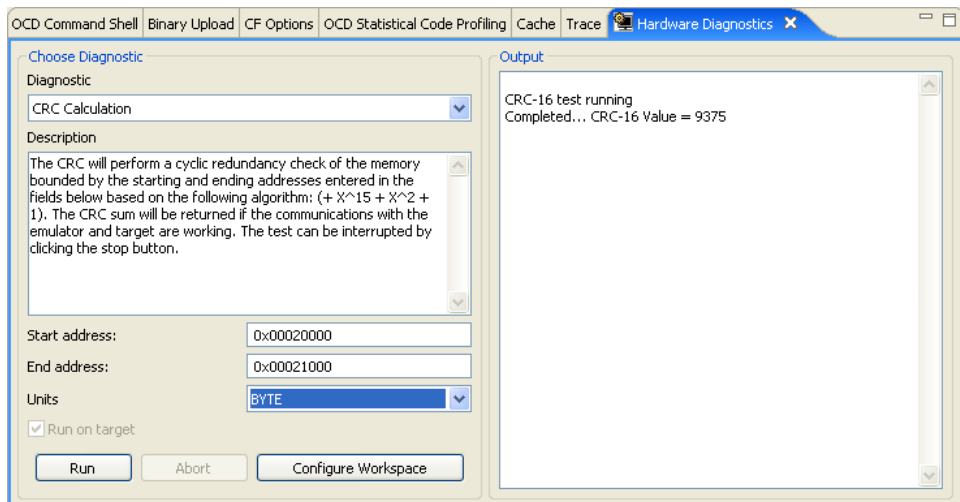
Workbench uses this polynomial as follows:

Workbench reads a location and uses the value read, x , to calculate the CRC. Then Workbench adds the result to the value calculated for the previous address. This process continues until Workbench has checked the entire specified memory range.

1. In the **Diagnostic** field, select **CRC Calculation**.
2. In the **Start Address** field, enter the starting address, for example **20000**.
3. In the **End Address** field, enter the ending address, for example **21000**.
4. Set the **Units** field to **BYTE**.
5. Click **Run**.

To interrupt the test, click **Stop**.

If communications with the emulator and target are working, Workbench returns the CRC sum.



14.4.4 Scope Tests

Read From Location

The **Read From Location** Scope Test performs a memory read of designated length from the address entered in the **From Address** field.

Write To Location

The **Write To Location** Scope Test performs a memory write of designated length of the value entered in the **Data Value** field to the address in the **To Address** field.

Write and Complement

The **Write and Complement** Scope Test performs a memory write of designated length of the value entered in the **Data Value** field to the address in the **To Address** field; the value is then complemented.

Write Rotating Value

The **Write Rotating Value** Scope Test performs a memory write of the value entered in the **Data Value** field to the address in the **To Address** field. The value is then rotated through all of the bit positions with respect to the designated length of the memory address.

Write Then Read

The **Write then Read** Scope Test performs a memory write of designated length of the value entered in the **Data Value** field to the address in the **To Address** field; the value is then read back.

14.4.5 Bus Tests

Address Bus Test

This test detects faults in the address bus over the range bounded by the starting and ending addresses entered in the **Start Address** and **End Address** fields. This test can be interrupted by clicking the **Abort** button.

Data Bus Test

This test detects faults in the data bus over the range bounded by the starting and ending addresses entered in the **Start Address** and **End Address** fields. This test can be interrupted by clicking the **Abort** button.

15

OCD Statistical Code Profiling

- 15.1 Introduction 255
- 15.2 Connecting to the Target 256
- 15.3 Creating a Project 266
- 15.4 Profiling Your Code 272

15.1 Introduction

The **OCD Statistical Code Profiling** view provides built-in performance analysis and code coverage features that allow you to profile your software's performance and view a symbolic display in chart or histogram format. These features help identify system bottlenecks and let you optimize your application software.

This chapter provides a tutorial for using the **OCD Statistical Code Profiling** view to profile your code.

To populate the **OCD Statistical Code Profiling** view, you must have an active project and an active target connection.

This tutorial uses the Wind River Instruction Set Simulator and the C Demonstration Program, both of which are included in your Workbench installation.

15.2 Connecting to the Target

First, open Workbench according to the method for your host computer.

Linux/Solaris Hosts

From your installation directory, issue the command

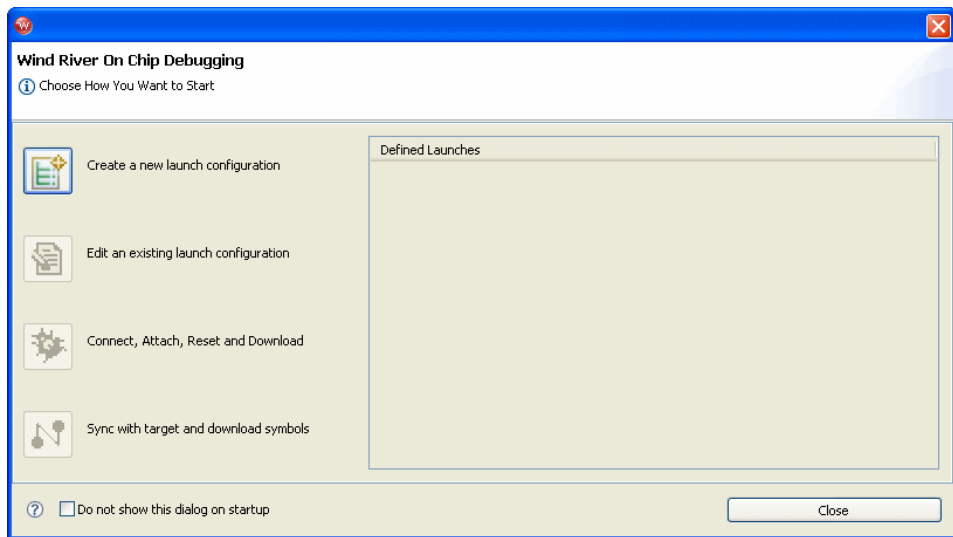
```
$ ./startWorkbench.sh
```

Windows Hosts

Select **Start > All Programs > Wind River > Wind River Workbench** *version*.

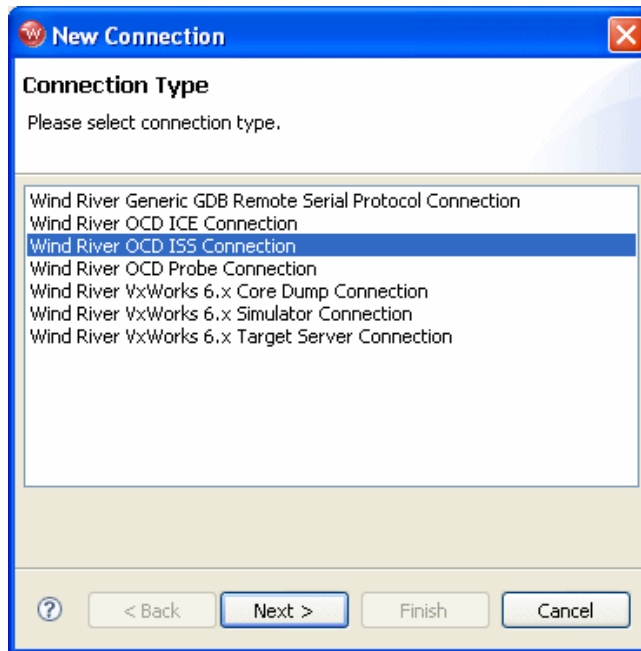
On Windows hosts, Workbench prompts you to specify a workspace location. Linux/Solaris hosts use the default location *installDir/workspace*.

When Workbench opens, the **Quick Target Launch** dialog appears.

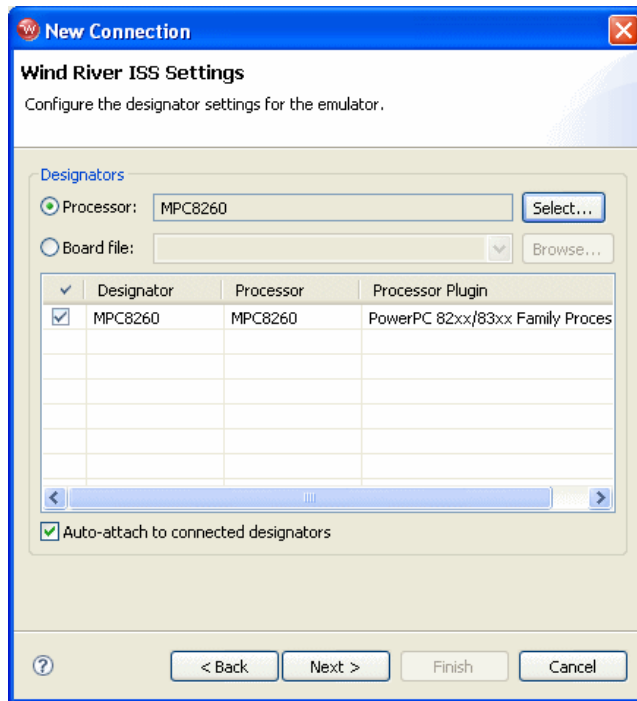


1. Select **Create a new launch configuration**.

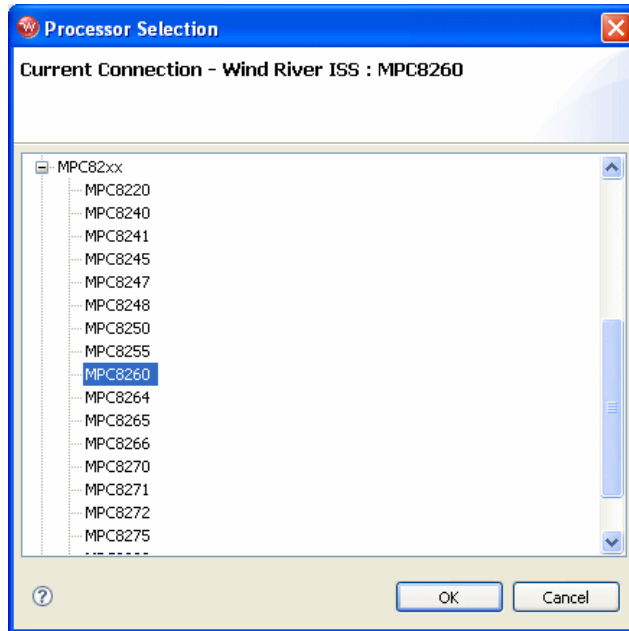
The **Connection Type** dialog appears.



2. Select **Wind River OCD ISS Connection** and click **Next**.
The **Processor Selection** dialog appears.



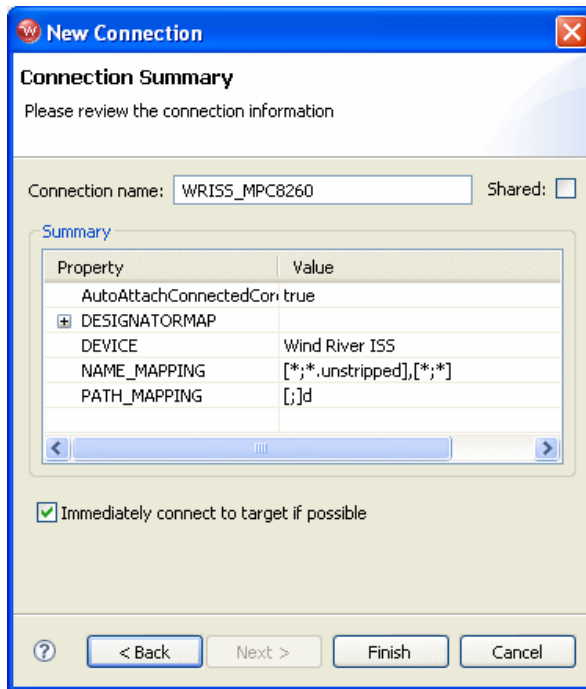
3. Click **Select**. From the list that appears, expand **MPC82xx** and select **MPC8260**.



4. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

5. Click **Next**.
6. The connection wizard passes through a number of screens that you do not need to configure for this tutorial. Leave all settings at their defaults and click **Next** until you come to the **Connection Summary**.



7. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

Workbench creates a target connection called **WRISS_MPC8260** in the **Target Manager** view.

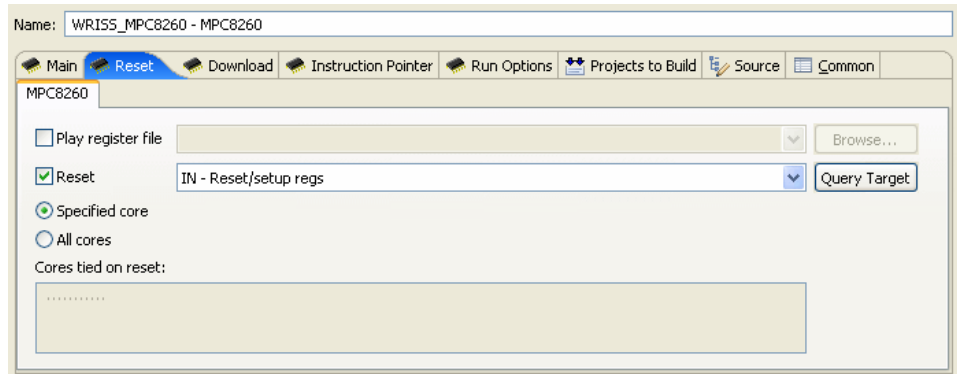


NOTE: On Windows hosts, Workbench starts **WindISS.exe** and opens a command shell. Do not close this shell or terminate **WindISS.exe** while your target connection is running. Workbench automatically terminates **WindISS.exe** and closes the shell when you disconnect from the target connection.

The **Reset and Download** view appears.

8. Choose how you want to proceed:
 - a. If you want to create a project in which to run and debug your code, skip the rest of this section and proceed to [15.3 Creating a Project](#), p.266.

- b. If you want to run and debug your code without creating a project, continue with this section.
9. In the **Reset and Download** view, select the **Reset** tab.



10. If you want to configure the target register values with a register file, select **Play Register File** and browse for the file you want to use.

Register files for many Wind River-supported targets are located in *installDir/workbench-2.x/dfw/build/host/registers*.

If you do not want to reconfigure your target registers, leave this box unchecked.

11. Choose the type of reset initialization you want to perform.

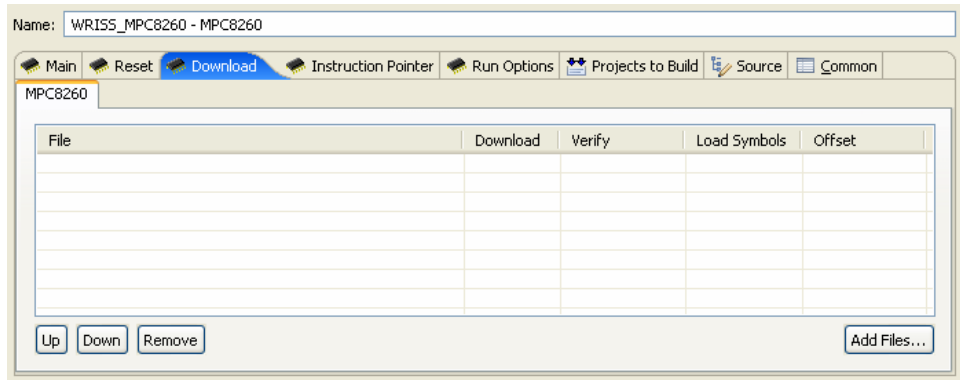
You can use the **IN** or **INN** initialization commands. For a full discussion of these two commands, see the *Wind River Workbench for On-Chip Debugging Command Reference*.

You can also choose not to perform an initialization by clearing the **Reset** box.



CAUTION: If you are manually changing registers on your target, be aware that issuing an **IN** or **INN** initialization command will overwrite your changes.

12. Select the **Download** tab.



13. Click **Add Files**.

In the browser window that appears, navigate to the executable file you want to run.

The file you select appears in the **Filename** field. Repeat this process as many times as necessary.

The file at the top of the list will download to the target first, followed by the others from the top down. You can edit the order of the list by clicking on any filename to highlight it and using the **Up**, **Down**, and **Delete** buttons.

14. Use the other fields to configure the download.

Download

The **Download** field is checked by default. If you clear it, the file will remain on the list but will not download data to the target. This is useful if, for example, you only want to download symbol information and not data.

Verify

The **Verify** field configures the extent to which the file you are downloading will be compared to a file that may already be on the target. There are three options: **Full**, **Compare**, and **None**.

When this field is set to **Full**, a write/read verify will occur for every download. Workbench writes to the target and then verifies that the write to the target and the read from the target are identical. This is slower than a normal download, but it is a useful security option.

When the field is set to **Compare**, Workbench will verify that the image has been downloaded correctly (that is, that the image on the host is the same as the image on the target.) This is useful for programming flash.



NOTE: You should only set the **Verify** field to **Compare** if an image already exists on the target. If you set the field to **Compare** when there is no image on the target, Workbench will look for a file to compare and not find one, and the reset and download operation will fail.

When the field is set to **None**, Workbench will perform no verification.

The **Verify** field is set to **None** by default.

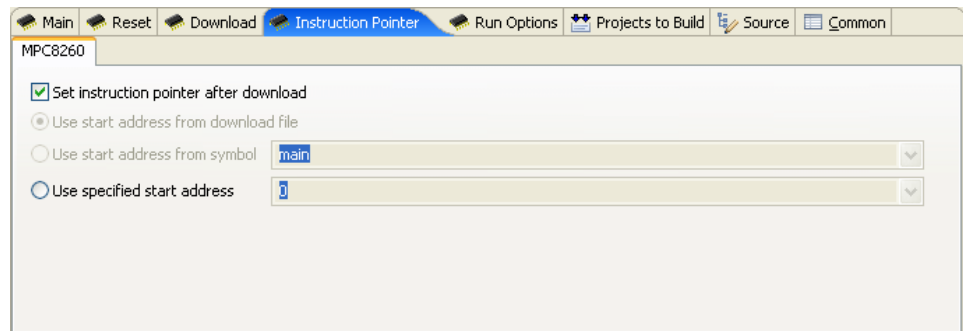
Load Symbol

The **Load Symbol** field, which is checked by default, determines whether the file's symbol information is downloaded to the target.

Offset

In the **Offset** field, you can enter a value in hex to set a memory offset bias for your application file. If you do not enter a value, Workbench uses the default value **0x00000000**.

15. Select the **Instruction Pointer** tab.



16. Set the starting point for your file.

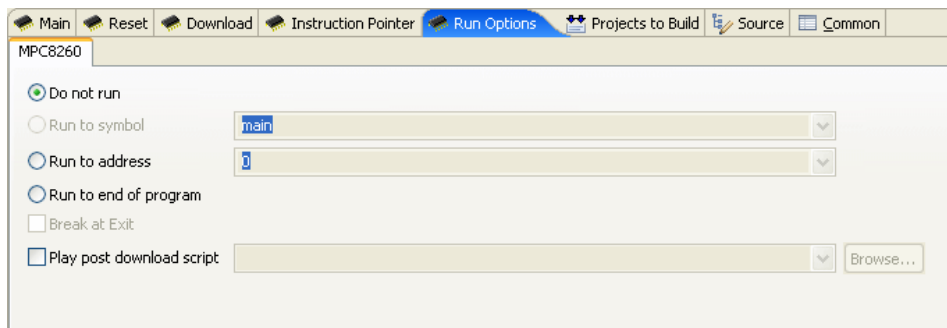
By default, the instruction pointer is set to use the starting address from the download file.

You can set the instruction pointer to start the file from the first occurrence of a particular symbol (for example, **main**) or you can just specify a starting

address by entering the address value in hex in the **Use Specified Start Address** field.

If you do not want to set a starting point, clear the **Set Instruction Pointer After Download** box.

17. Select the **Run Options** tab.



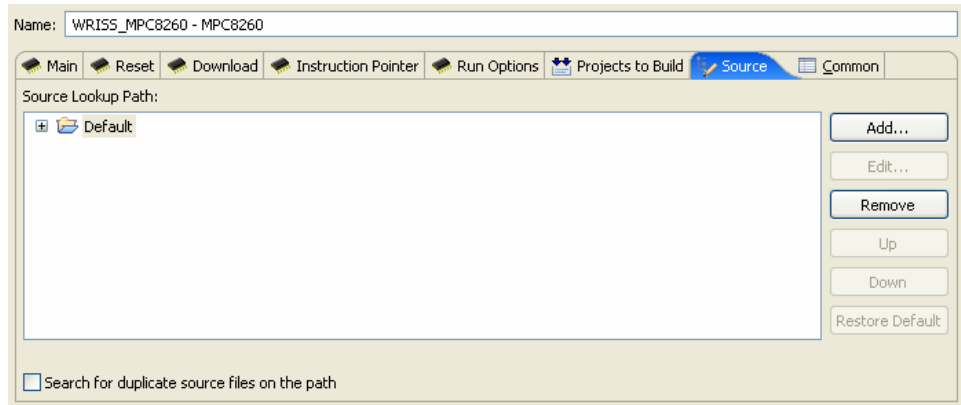
18. Determine how you want your file to run.

By default, the **Reset and Download** view is set not to run the file after downloading. If you want the file to run, you have several options to determine where it should break:

- You can set it to break at the first occurrence of a symbol (for example, **main**) by selecting **Run to Symbol** and entering the symbol in that field.
- You can set it to break at the end of your program by selecting **Run to end of program**.
- You can set it to break at a given memory address by selecting the **Run to Address** box and entering the address in hex in that field.
- You can set it to break at an **_exit** routine by selecting the **Break at Exit** box.

If you need to perform a post-initialization, you can define it here. Select the **Play post download script** box and click **Browse**. In the browser window that appears, navigate to your initialization file.

19. Select the **Source** tab.

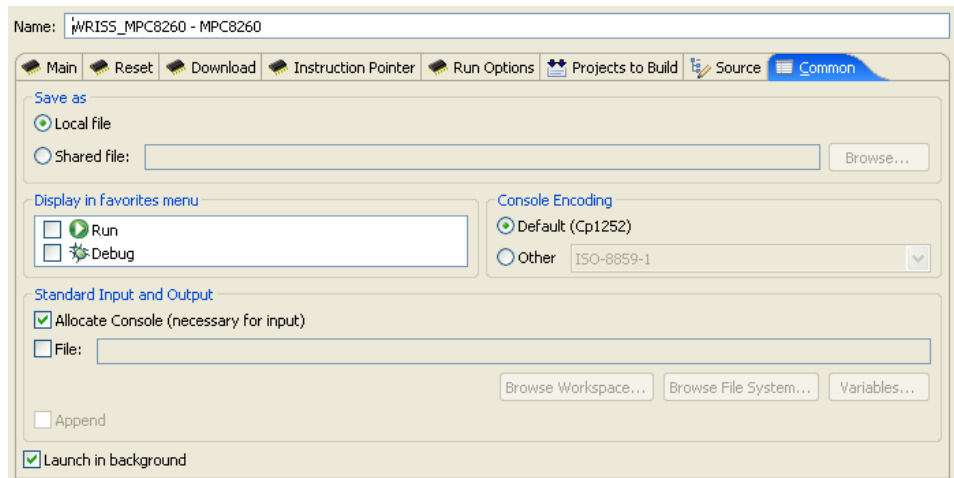


20. Use the **Source** tab to configure the source path of your file.

Workbench uses the input path of the local file system by default. Unless you need to use a different path, you do not need to do anything in the **Source** tab.

If you need to use a different path, click **Add...** and use the **Add Source** dialog to configure the appropriate search path for your project.

21. Select the **Common** tab.



22. Specify whether your launch configuration is local or shared.

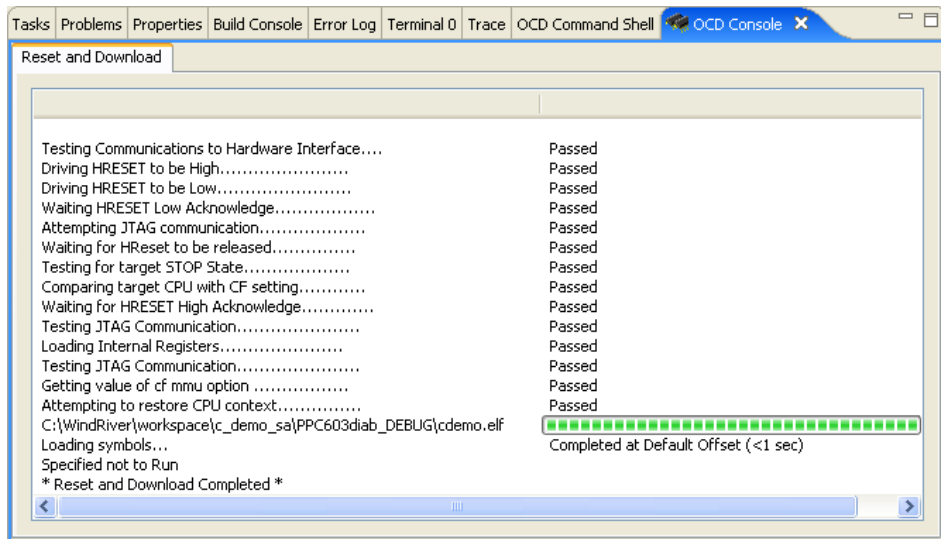
The configuration is local by default. To make it shared, click **Shared file:** and browse to the shared directory where you want the configuration to be located.

You have now fully defined your reset and download operation.

23. Click **Debug.**

Workbench initializes the target board, then downloads the file, then runs the file.

The **OCD Console** view opens to show the progress of the reset and download operation.



Proceed to [15.4 Profiling Your Code](#), p.272.

15.3 Creating a Project



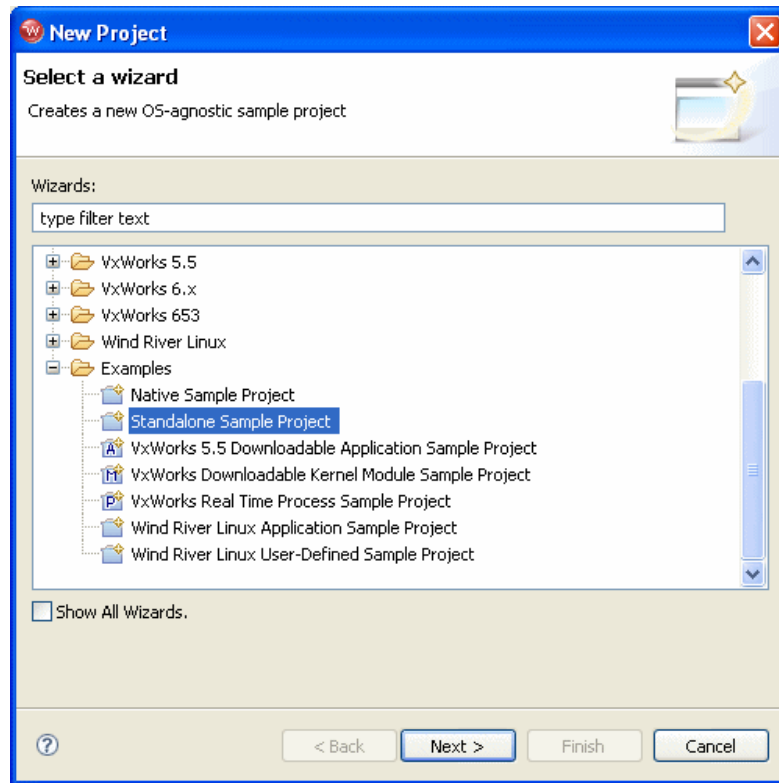
NOTE: If you do not plan to build or edit your source files within Workbench, skip this section and proceed to [15.4 Profiling Your Code](#), p.272.

In the **Reset and Download** view, click **Close.**

To create the C Demonstration Project, use the following steps.

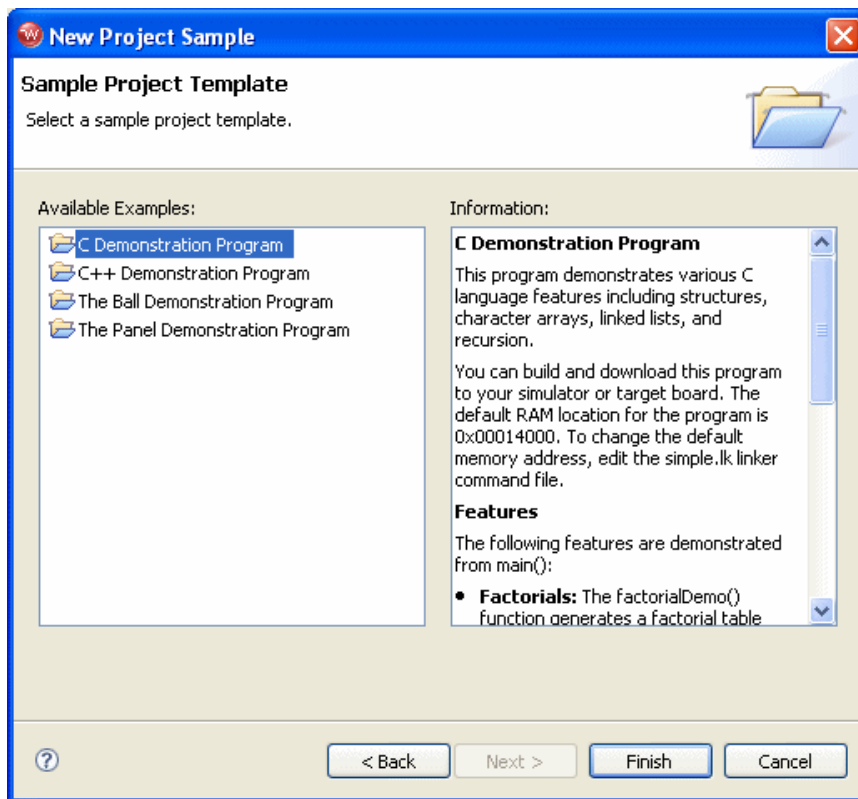
1. In the Workbench toolbar, select **File > New > Project**.

The **New Project** wizard appears.



2. Expand the **Examples** folder and select **Standalone Sample Project**.
3. Click **Next**.

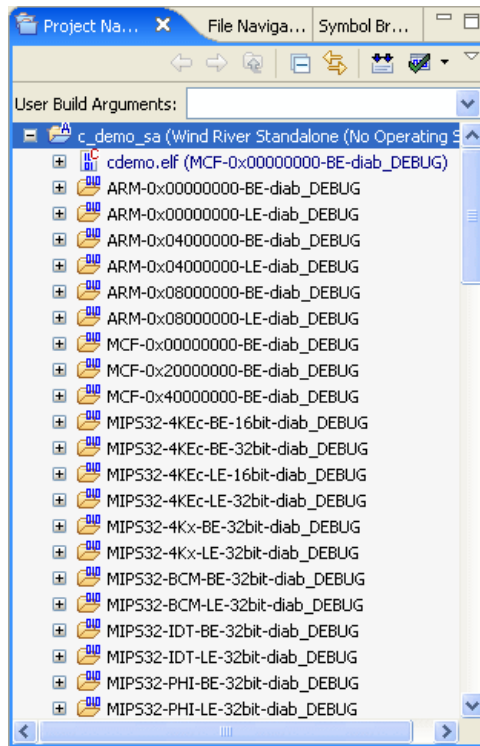
A sample project template appears.



4. Select **C Demonstration Program** and click **Finish**.

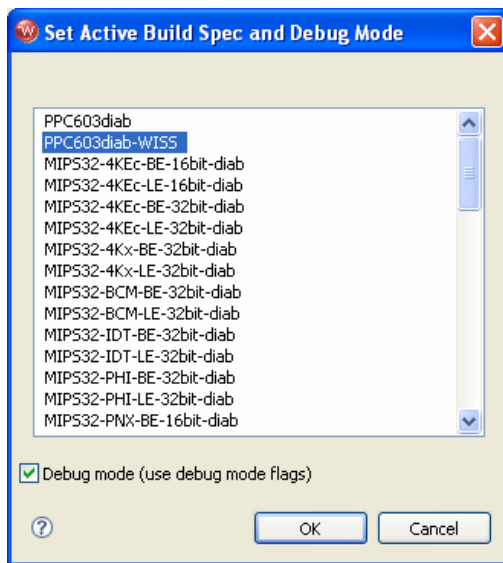
Workbench creates the sample project in the default workspace folder and opens the **Application Development** perspective.

5. In the **Project Navigator** view, expand the `c_demo_sa` project.



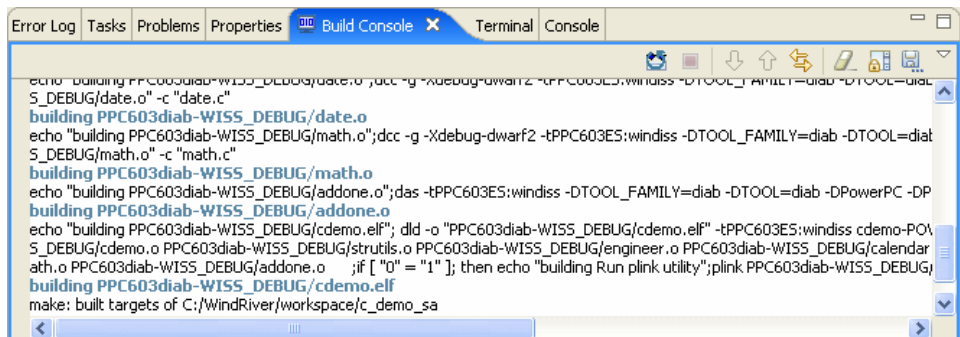
6. To build the sample project for use with the Wind River Instruction Set Simulator (WISS), right-click on the `c_demo_sa` top-level folder and select **Build Options > Set Active Build Spec**.

The **Set Active Build Spec and Debug Mode** dialog appears.



7. Scroll to the top and highlight **PPC603diab-WISS**.
8. Select **Debug mode (use debug mode flags)** so Workbench will generate symbolic debug information.
9. Click **OK**.
10. Right-click on the project name and select **Rebuild Project**.

Workbench builds the sample project. The results of the project build appear in the **Build Console** view.

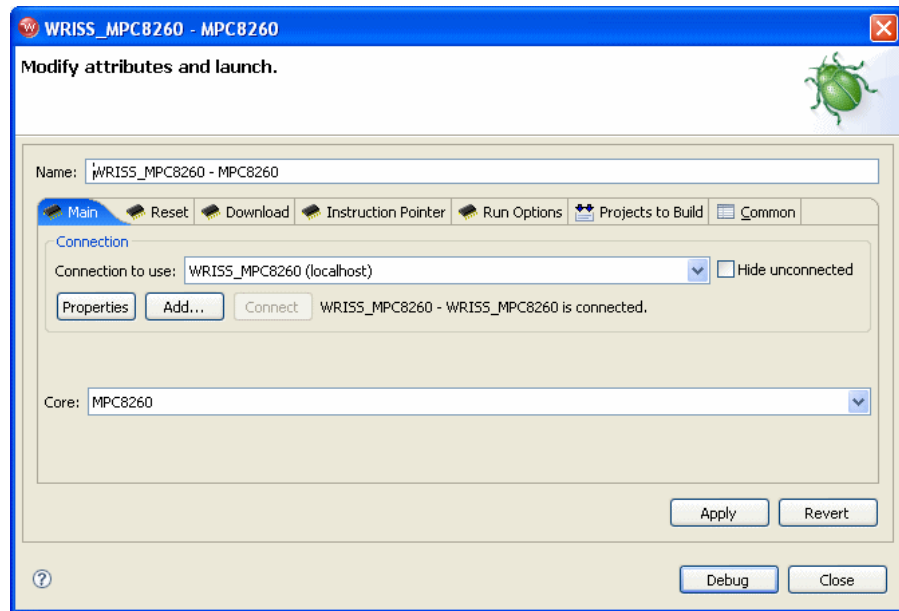


15.3.1 Downloading the Sample Code

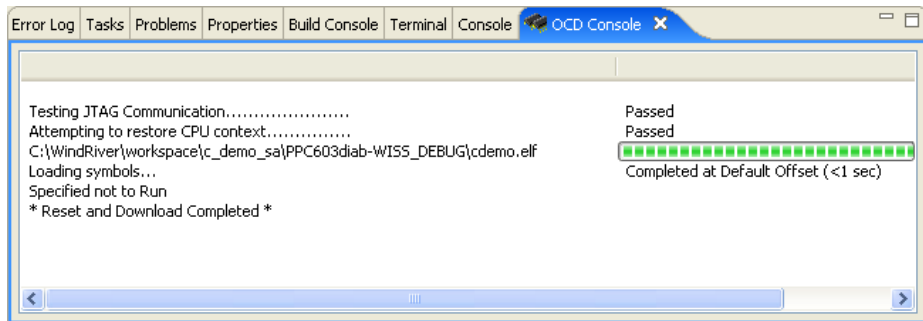
To run the sample code, use the following steps:

1. In the **Target Manager**, highlight the target connection name **WRISS_MPC8260**.
2. In the **Project Navigator** view, right-click on **cdemo.elf** and select **Reset and Download**.

The **Reset and Download** view appears.



3. Leave all settings at their defaults and click **Debug**.
The **OCD Console** view opens.



The **OCD Console** view shows the progress of the download operation, as Workbench downloads the sample code to the Wind River Instruction Set Simulator.

Proceed to [15.4 Profiling Your Code](#), p.272.

15.4 Profiling Your Code

To perform statistical profiling analysis on your code, use the following steps:

1. In the Workbench toolbar, select **Window > Show View > OCD Statistical Code Profiling**.

The **OCD Statistical Code Profiling** view opens.

The screenshot shows the 'OCD Statistical Code Profiling' window with a table of function data. The table has five columns: Function Name, Module Name (File B...), Start Address, End Address, and CPU Percentage. The functions listed are sorted by their start address in ascending order.

Function Name	Module Name (File B...	Start Address	End Address	CPU Percentage
main	C:/WindRiver/works...	0x00014030	0x00014338	0.00
strcpy	C:/WindRiver/works...	0x00014338	0x0001439c	0.00
strcmp	C:/WindRiver/works...	0x0001439c	0x00014404	0.00
engineers	C:/WindRiver/works...	0x00014404	0x00014584	0.00
dayOfYear	C:/WindRiver/works...	0x00014584	0x00014674	0.00
dateForDayNum	C:/WindRiver/works...	0x00014674	0x00014740	0.00
daysBetween	C:/WindRiver/works...	0x00014740	0x000147b0	0.00
calendar	C:/WindRiver/works...	0x000147b0	0x00014938	0.00
addCell	C:/WindRiver/works...	0x00014938	0x00014984	0.00
swapCells	C:/WindRiver/works...	0x00014984	0x00014a00	0.00
linkList	C:/WindRiver/works...	0x00014a00	0x00014bd8	0.00
send_month	C:/WindRiver/works...	0x00014bd8	0x00014c50	0.00
date	C:/WindRiver/works...	0x00014c50	0x00014d1c	0.00
factorial	C:/WindRiver/works...	0x00014d1c	0x00014d64	0.00
factorialDemo	C:/WindRiver/works...	0x00014d64	0x00014e10	0.00

The view populates with the functions from your code. (Note that the view only populates when there is code on the target. If you opened the view without downloading code as described above, the view would be empty.)

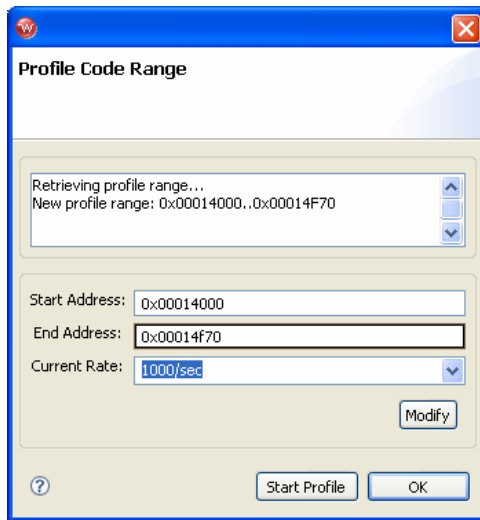
The functions are arranged in five columns, listing the function name, the full path to the function, the start and end addresses of the function, and the percentage of CPU time each function used. Since you have not yet run the code, the **CPU Percentage** column just reads **0.00**.

To sort the functions, click on a column heading. In the above image, the arrow in the heading of the **Start Address** column shows that the functions are sorted by lowest starting address.



NOTE: By default, the **Module Name** field shows the path to where the function was built. If you have a function that is not physically located where it was built, then in the **OCD Statistical Code Profiling** view you can click **Toggle local build/source paths in table** to make the **Module Name** field show the full path to the file's physical location.

2. In the **OCD Statistical Code Profiling** view, click **Configure PFA Code Range**. The **Profile Code Range** dialog appears.



3. Use the **Start Address** and **End Address** fields to set the desired code range.

To obtain the start address, click the heading of the **Start Address** column. (The arrow in the heading of the **Start Address** column should be pointing down, to show that the first address is at the top of the column. If it is pointing up, click the **Start Address** column heading again.) For the C demonstration program, the first address is the start of the function **main** at 0x00014030.



NOTE: You can set the **Profile Code Range** dialog to populate with the beginning of your downloaded code automatically, by clicking the **Show PFA preference page** button and selecting the **Set profile range to beginning of downloaded code** checkbox in the dialog that appears.

To obtain the end address, click the heading of the **End Address** column. (The arrow in the heading of the **Start Address** column should be pointing up, to show that the last address is at the top of the column. If it is pointing down, click the **End Address** column heading again.) For the C demonstration program, the last address is the end of the function **abs** at 0x00014f64.

4. In the **Start Address** field, enter 0x00014030.
5. In the **End Address** field, enter 0x00014f64.
6. Click **Modify**.
7. Click **OK**.

8. In the **OCD Statistical Code Profiling** view, click **Start PFA Profiling**.

This starts your code running in PFA mode. A **>PFA>** prompt appears in the **OCD Command Shell**.

9. In the **OCD Statistical Code Profiling** view, click **Stop PFA Profiling**.

This returns the target to Background Mode. The **OCD Statistical Code Profiling** view is now populated with a list of the functions called while the code was running, showing the percentage of run time for each function.

The screenshot shows the 'OCD Statistical Code Profiling' window with the 'Profile Data' tab selected. The table below represents the data shown in the screenshot:

Function Name	Module Name (File B...	Start Address	End Address	CPU Percentage
main	C:/WindRiver/works...	0x00014030	0x00014338	7.43
strcpy	C:/WindRiver/works...	0x00014338	0x0001439c	11.89
strcmp	C:/WindRiver/works...	0x0001439c	0x00014404	31.85
engineers	C:/WindRiver/works...	0x00014404	0x00014584	5.10
dayOfYear	C:/WindRiver/works...	0x00014584	0x00014674	25.69
dateForDayNum	C:/WindRiver/works...	0x00014674	0x00014740	3.40
daysBetween	C:/WindRiver/works...	0x00014740	0x000147b0	1.49
calendar	C:/WindRiver/works...	0x000147b0	0x00014938	2.76
addCell	C:/WindRiver/works...	0x00014938	0x00014984	0.64
swapCells	C:/WindRiver/works...	0x00014984	0x00014a00	0.42
linkList	C:/WindRiver/works...	0x00014a00	0x00014bd8	4.46
send_month	C:/WindRiver/works...	0x00014bd8	0x00014c50	0.85
date	C:/WindRiver/works...	0x00014c50	0x00014d1c	2.76
factorial	C:/WindRiver/works...	0x00014d1c	0x00014d64	0.21
factorialDemo	C:/WindRiver/works...	0x00014d64	0x00014e10	0.00

To see graphic representations of the code profile, click the **Profile Plot** tab.

The **OCD Statistical Code Profiling** view can display information in any of four graph types. To cycle between these graph types, use the **Show next graph type** and **Show previous graph type** buttons.

To change the color assigned to each function, use the **Change graph colors** button.

Figure 15-1 Two-dimensional Bar Graph

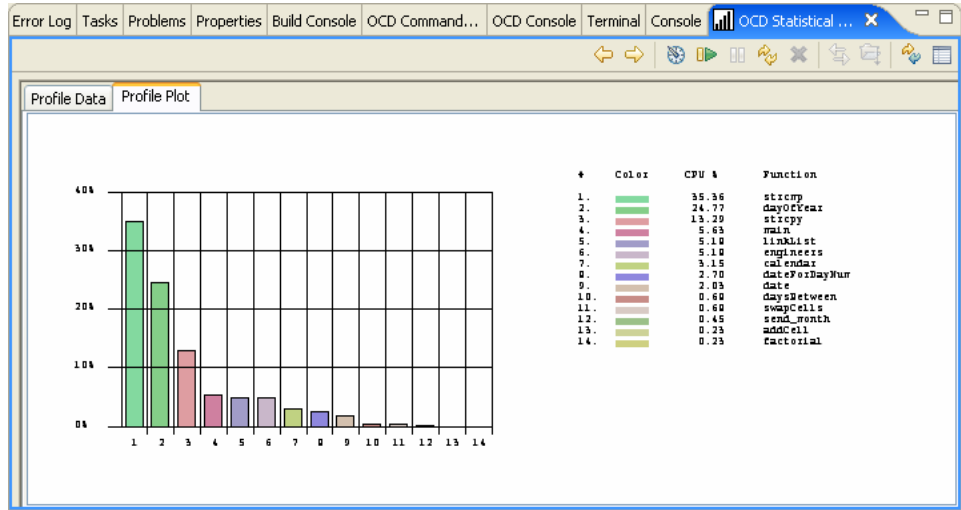


Figure 15-2 Three-dimensional Bar Graph

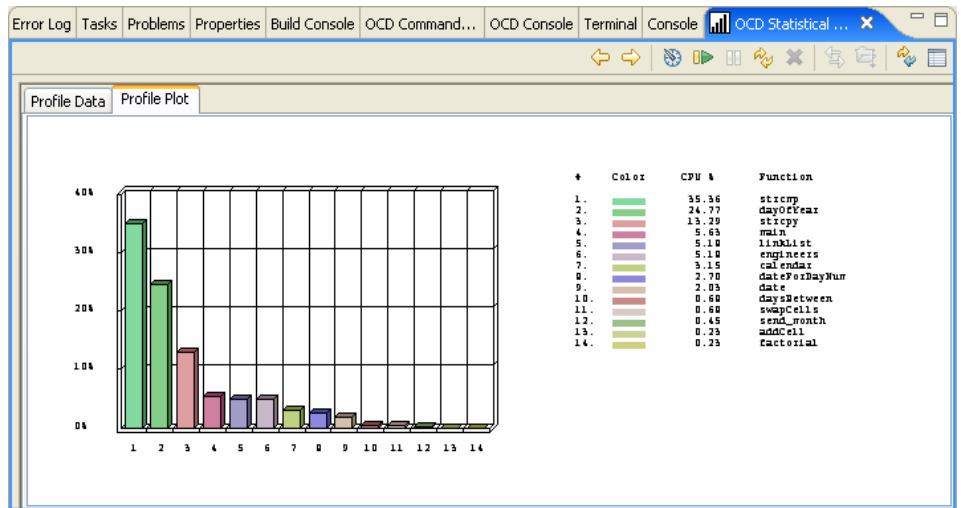


Figure 15-3 Two-dimensional Pie Graph

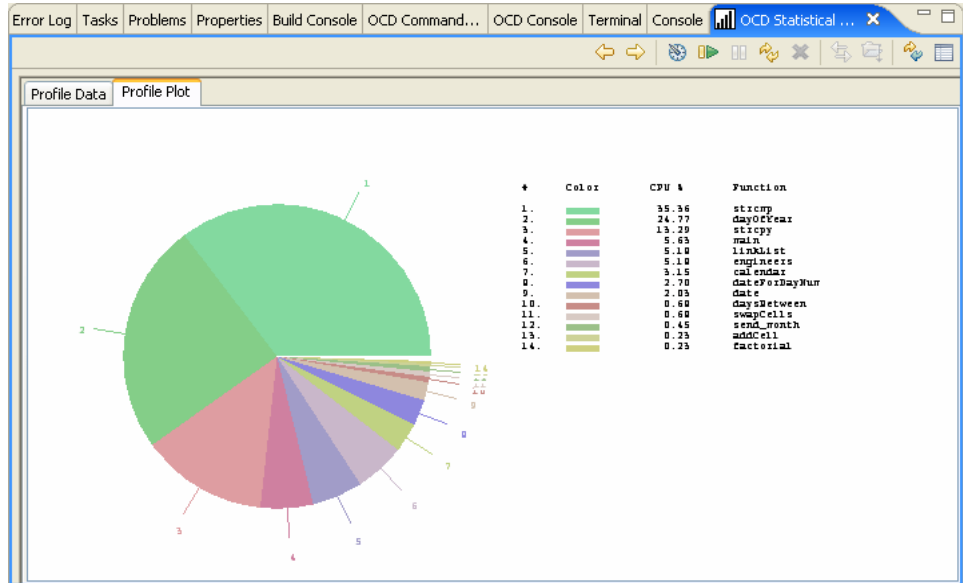
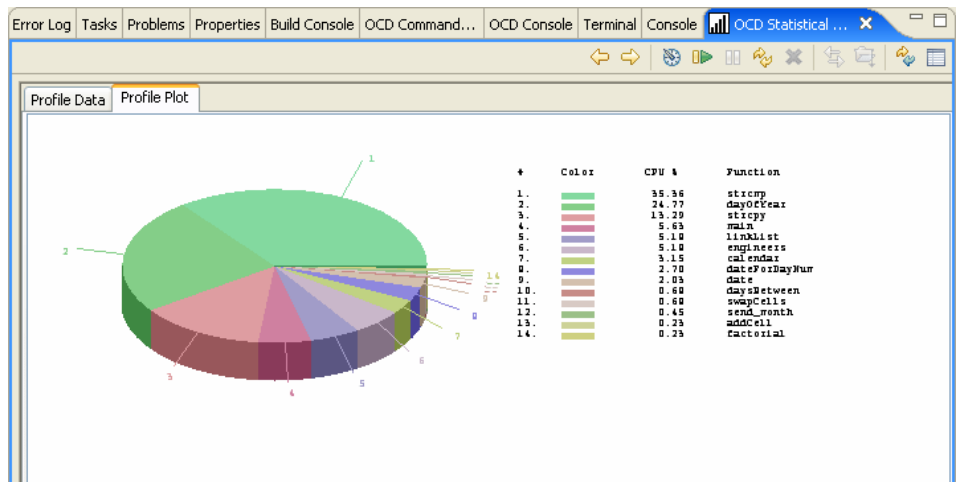


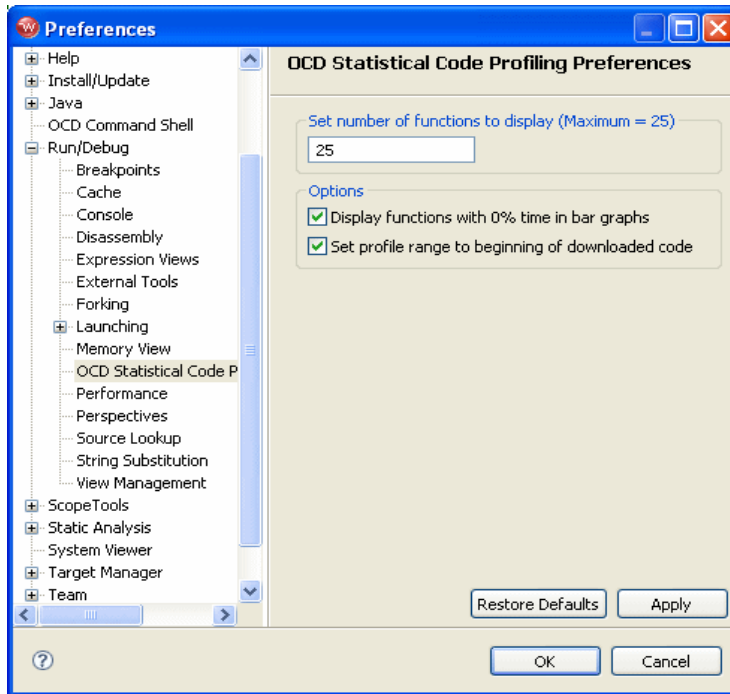
Figure 15-4 Three-dimensional Pie Graph



By default, these graphs show only functions that used CPU time while your code was running. You can also set the bar graphs (though not the pie graphs) to show

all functions, whether they were called or not, by clicking the **Show PFA preference page** button, setting the **Set number of functions to display** field to 25 (the maximum), selecting the **Display functions with 0% time in bar graphs** checkbox, and clicking **Apply**.

Figure 15-5 **OCD Statistical Code Profiling Preferences Dialog**



15.4.1 Profiling Selected Functions

You can also set the **OCD Statistical Code Profiling** view to profile only selected functions. For example, say you are only interested in the function **calendar**. In the **Start Address** and **End Address** columns you can see that the function **calendar** starts at 0x00014740 and ends at 0x00014938. Enter those values in the **Start Address** and **End Address** fields in the **Profile Code Range** dialog, and click **Modify** and then **OK**. Now when you click **Start PFA Profiling**, the **OCD Statistical Code Profiling** view returns data only for the function **calendar**.

15.4.2 **Browsing Functions in Source**

To see the source for any function, highlight the function in the **Profile Data** tab and click **Browse PFA function in source**. The selected function is displayed in the Workbench editor.

15.4.3 **Updating the Profile Data**

To update profile data, use the **Refresh/Update PFA grid data** button. You can see the output for each function in the **OCD Command Shell**.

15.4.4 **Removing Functions**

To delete a function from the **Profile Data** tab, highlight the function and click **Remove selected function**.

16

Using the Cache View

- 16.1 Introduction 281
- 16.2 Connecting to the Target 282
- 16.3 Creating a Project 293
- 16.4 Examining Cache 299
- 16.5 Viewing Cache Source 303
- 16.6 Comparing Memory 303
- 16.7 Reconfiguring the Cache 305
- 16.8 Exporting Cache Information 305
- 16.9 Using Processors Without Cache Lines 306

16.1 Introduction

Use the **Cache** view to view instructions and data stored in cache.

The **Cache** view has two tabs: **Cache Lines** and **Advanced Control and Status**. **Cache Lines** is the default tab, for processors (the majority) that organize their cache memory in logical blocks, each of which contains an address index, an address tag, and a given number of bytes of data; each such block of memory is called a *cache line*.

For targets that do not use cache lines, such as Freescale ColdFire, the **Cache** view cannot display cache information. However, the **Advanced Control and Status** tab provides cache control options for these targets.

In either case, the **Cache** view displays tabs for the **Instruction Cache** and the **Data Cache**. (If your target processor uses a multi-level cache, there may be additional tabs visible, such as **L2 Data Cache**, **L2 Instr Cache**, and so on.) Before you perform any operation in the **Cache** view, make sure you are in the appropriate tab.

To populate the **Cache** view, you must have an active target connection.

16.2 Connecting to the Target

First, open Workbench according to the method for your host computer.

Linux/Solaris Hosts

From your installation directory, issue the command

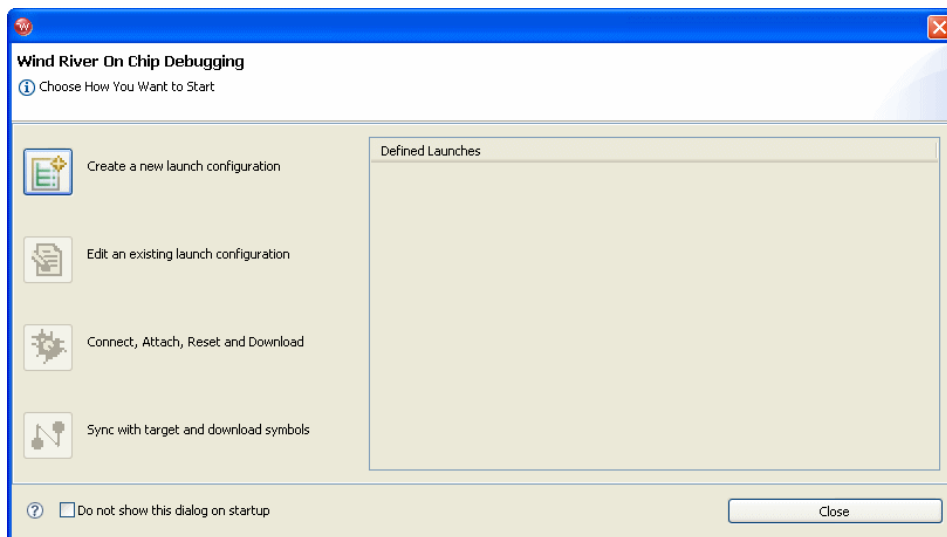
```
$ ./startWorkbench.sh
```

Windows Hosts

Select **Start > All Programs > Wind River > Wind River Workbench version**.

On Windows hosts, Workbench prompts you to specify a workspace location. Linux hosts use the default location *installDir/workspace*.

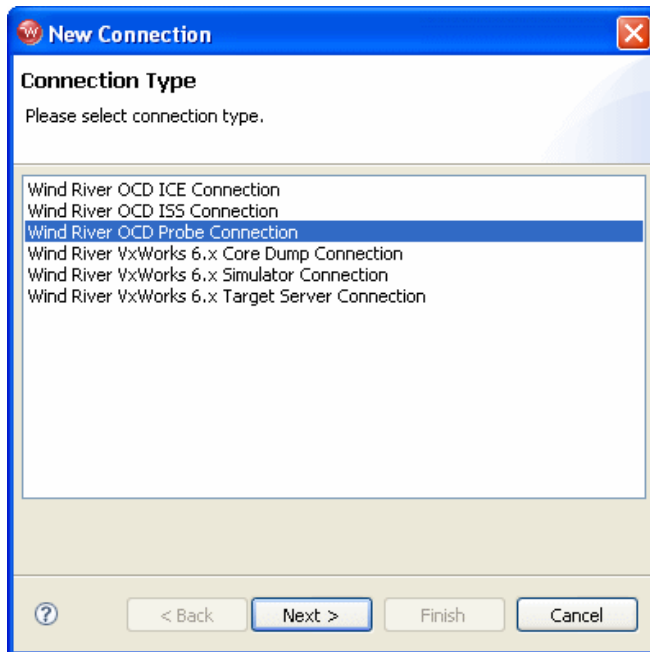
When Workbench opens, the **Quick Target Launch** dialog appears.



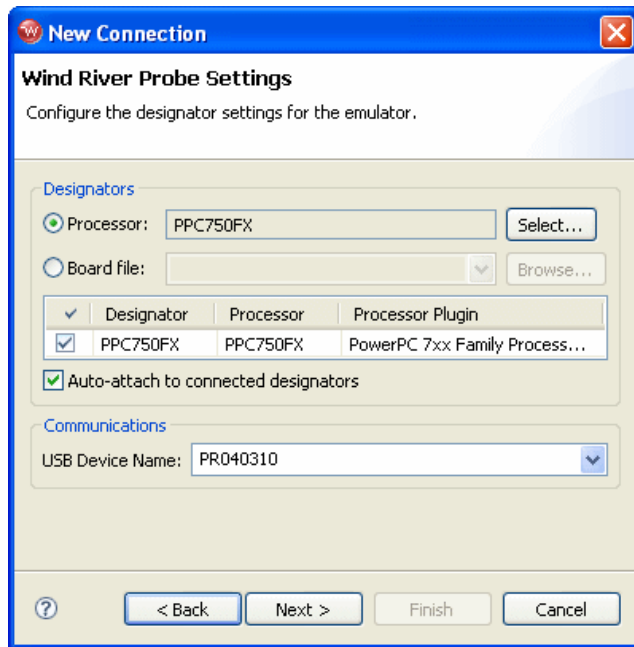
1. Select **Create a new launch configuration**.

The **Connection Type** dialog appears.

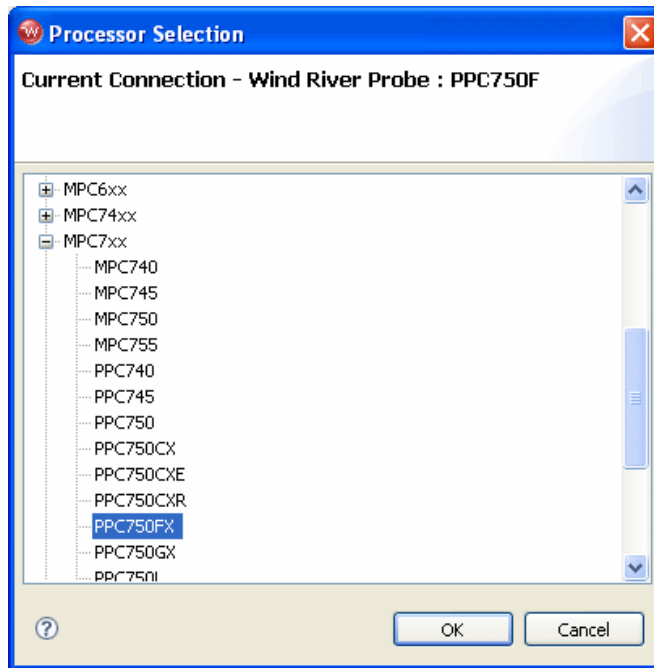
This tutorial uses a Wind River Probe emulator connected to a Wind River PPMC750FX target.



2. Select **Wind River OCD Probe Connection** and click **Next**.
The **Processor Selection** dialog appears.



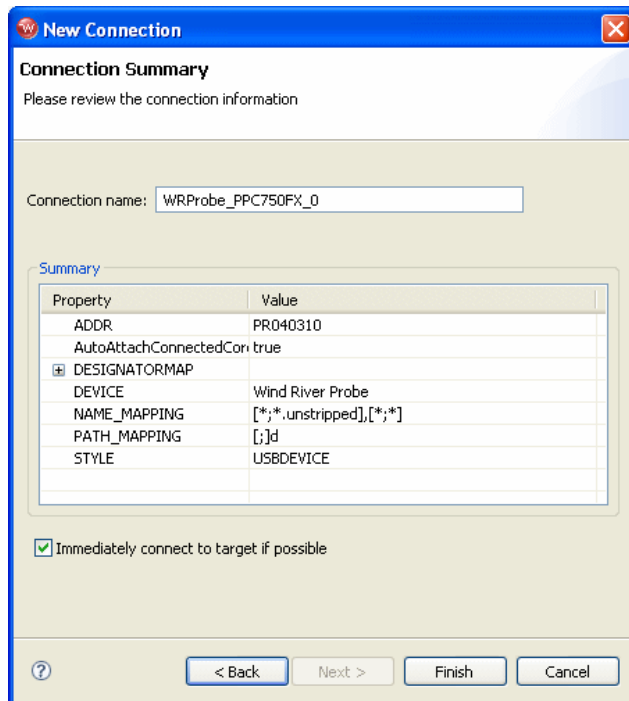
3. Click **Select**. From the list that appears, expand **MPC7xx** and select **PPC750FX**.



4. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

5. Click **Next**.
6. The connection wizard passes through a number of screens that you do not need to configure for this tutorial. Leave all settings at their defaults and click **Next** until you come to the **Connection Summary**.

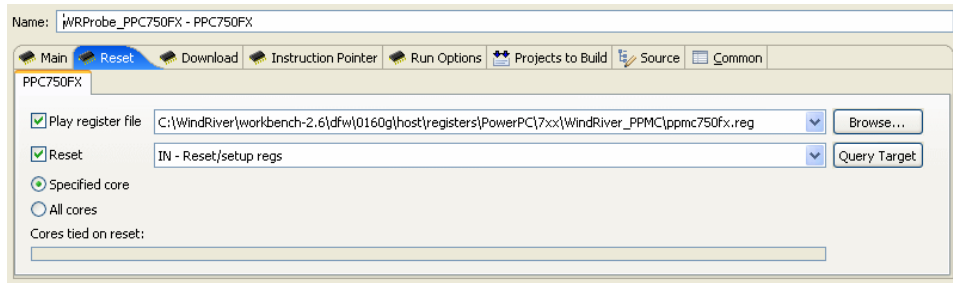


7. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

Workbench creates a target connection called **WRProbe_PPC750FX** in the **Target Manager** view.

The **Reset and Download** view appears.

8. Choose how you want to proceed:
 - a. If you want to create a project in which to run and debug your code, skip the rest of this section and proceed to [16.3 Creating a Project](#), p.293.
 - b. If you want to run and debug your code without creating a project, continue with this section.
9. In the **Reset and Download** view, select the **Reset** tab.



10. If you want to configure the target register values with a register file, select **Play Register File** and browse for the file you want to use.

Register files for many Wind River-supported targets are located in *installDir/workbench-2.x/dfw/build/host/registers*.

If you do not want to reconfigure your target registers, leave this box unchecked.

11. Choose the type of reset initialization you want to perform.

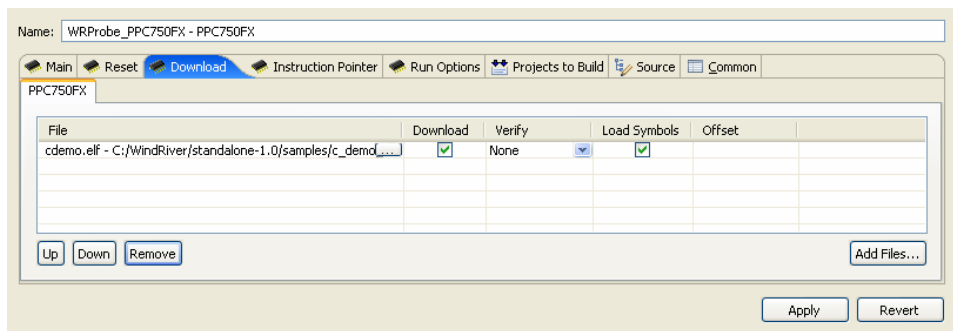
You can use the **IN** or **INN** initialization commands. For a full discussion of these two commands, see the *Wind River Workbench for On-Chip Debugging Command Reference*.

You can also choose not to perform an initialization by clearing the **Reset** box.



CAUTION: If you are manually changing registers on your target, be aware that issuing an **IN** or **INN** initialization command will overwrite your changes.

12. Select the **Download** tab.



13. Click **Add Files**.

In the browser window that appears, navigate to the executable file you want to run.

The file you select appears in the **Filename** field. Repeat this process as many times as necessary.

The file at the top of the list will download to the target first, followed by the others from the top down. You can edit the order of the list by clicking on any filename to highlight it and using the **Up**, **Down**, and **Delete** buttons.

14. Use the other fields to configure the download.

Download

The **Download** field is checked by default. If you clear it, the file will remain on the list but will not download data to the target. This is useful if, for example, you only want to download symbol information and not data.

Verify

The **Verify** field configures the extent to which the file you are downloading will be compared to a file that may already be on the target. There are three options: **Full**, **Compare**, and **None**.

When this field is set to **Full**, a write/read verify will occur for every download. Workbench writes to the target and then verifies that the write to the target and the read from the target are identical. This is slower than a normal download, but it is a useful security option.

When the field is set to **Compare**, Workbench will verify that the image has been downloaded correctly (that is, that the image on the host is the same as the image on the target.) This is useful for programming flash.



NOTE: You should only set the **Verify** field to **Compare** if an image already exists on the target. If you set the field to **Compare** when there is no image on the target, Workbench will look for a file to compare and not find one, and the reset and download operation will fail.

When the field is set to **None**, Workbench will perform no verification.

The **Verify** field is set to **None** by default.

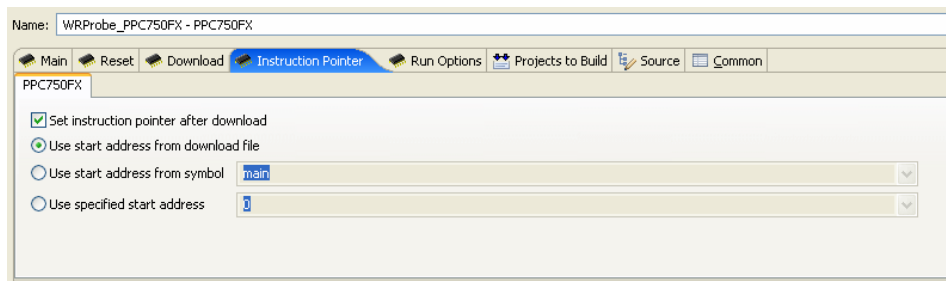
Load Symbol

The **Load Symbol** field, which is checked by default, determines whether the file's symbol information is downloaded to the target.

Offset

In the **Offset** field, you can enter a value in hex to set a memory offset bias for your application file. If you do not enter a value, Workbench uses the default value **0x00000000**.

15. Select the **Instruction Pointer** tab.



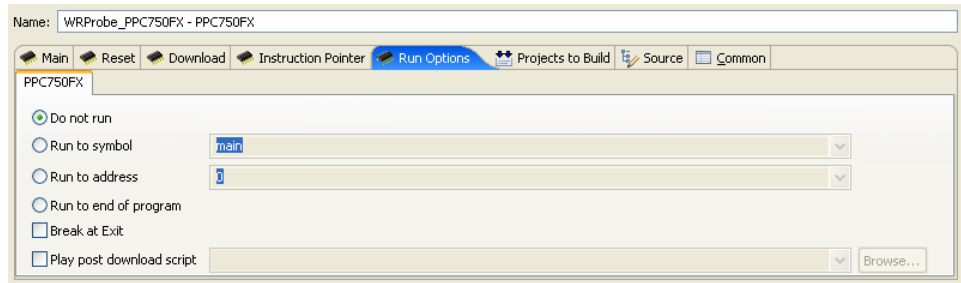
16. Set the starting point for your file.

By default, the instruction pointer is set to use the starting address from the download file.

You can set the instruction pointer to start the file from the first occurrence of a particular symbol (for example, **main**) or you can just specify a starting address by entering the address value in hex in the **Use Specified Start Address** field.

If you do not want to set a starting point, clear the **Set Instruction Pointer After Download** box.

17. Select the **Run Options** tab.



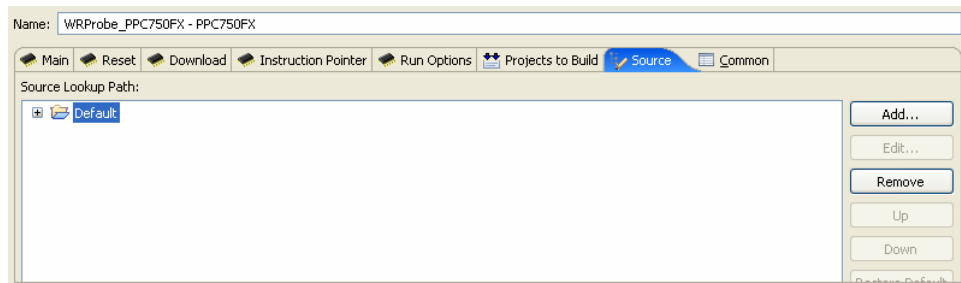
18. Determine how you want your file to run.

By default, the **Reset and Download** view is set not to run the file after downloading. If you want the file to run, you have several options to determine where it should break:

- You can set it to break at the first occurrence of a symbol (for example, **main**) by selecting **Run to Symbol** and entering the symbol in that field.
- You can set it to break at the end of your program by selecting **Run to end of program**.
- You can set it to break at a given memory address by selecting the **Run to Address** box and entering the address in hex in that field.
- You can set it to break at an **_exit** routine by selecting the **Break at Exit** box.

If you need to perform a post-initialization, you can define it here. Select the **Play post download script** box and click **Browse**. In the browser window that appears, navigate to your initialization file.

19. Select the **Source** tab.

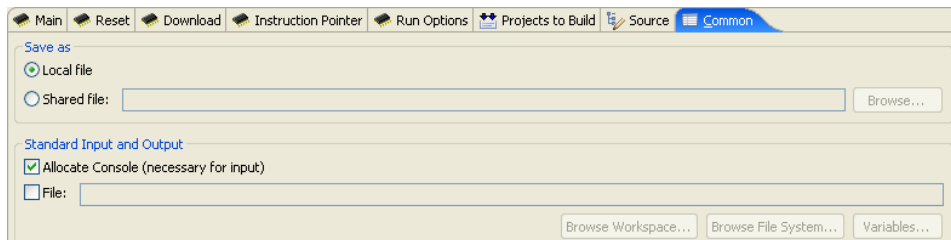


20. Use the **Source** tab to configure the source path of your file.

Workbench uses the input path of the local file system by default. Unless you need to use a different path, you do not need to do anything in the **Source** tab.

If you need to use a different path, click **Add...** and use the **Add Source** dialog to configure the appropriate search path for your project.

21. Select the **Common** tab.



22. Specify whether your launch configuration is local or shared.

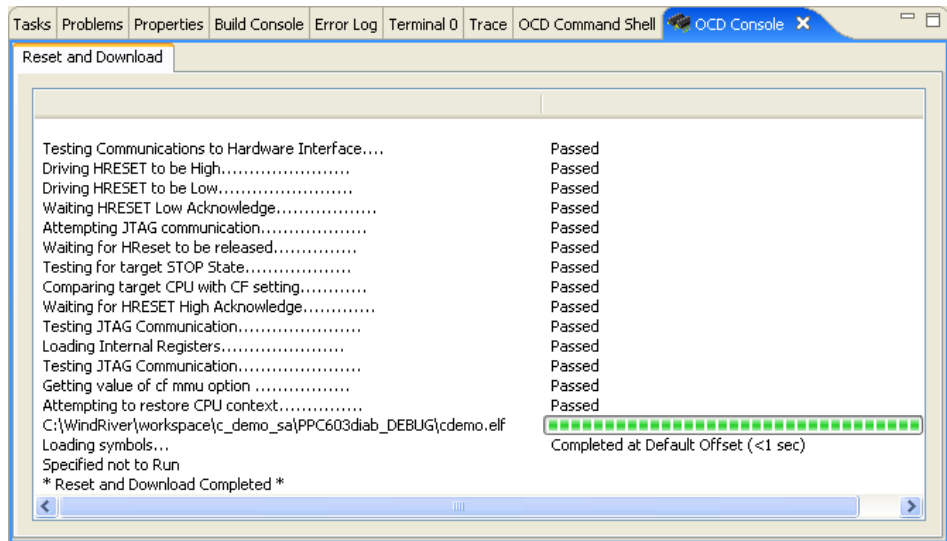
The configuration is local by default. To make it shared, click **Shared file:** and browse to the shared directory where you want the configuration to be located.

You have now fully defined your reset and download operation.

23. Click **Debug**.


Workbench initializes the target board, then downloads the file, then runs the file.

The **OCD Console** view opens to show the progress of the reset and download operation.



Proceed to [16.4 Examining Cache](#), p.299.

16.3 Creating a Project

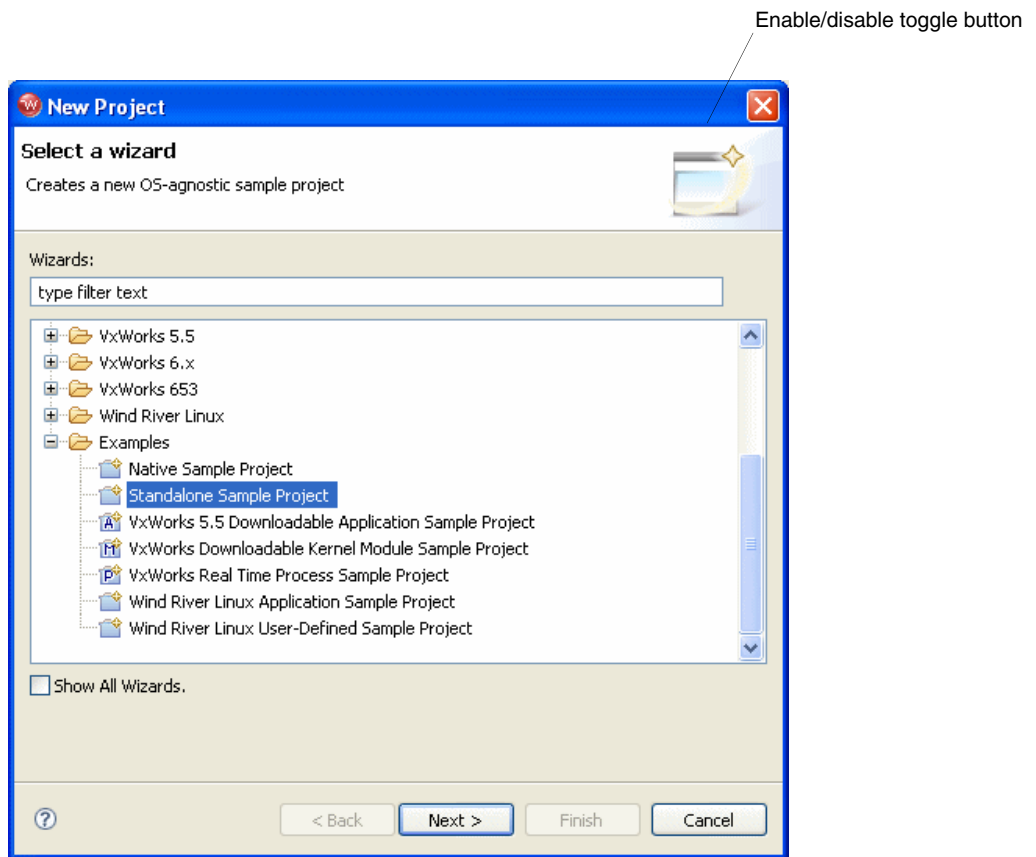
 **NOTE:** If you do not plan to build or edit your source files within Workbench, skip this section and proceed to [16.4 Examining Cache](#), p.299.

Click **Close** in the **Reset and Download** view.

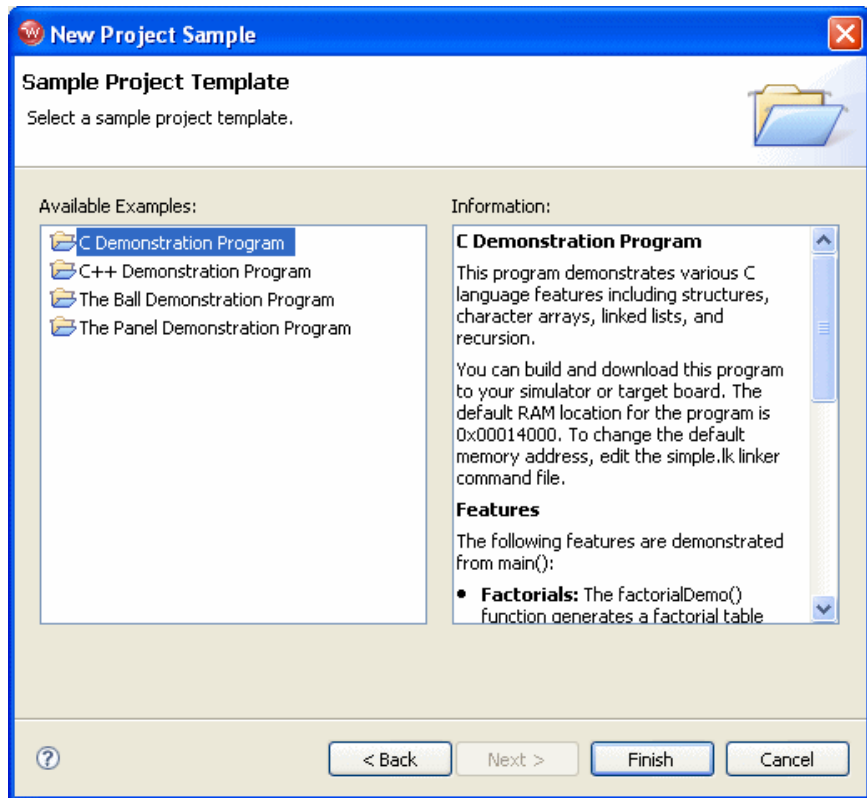
This tutorial uses the C Demonstration Program, which is included in your Workbench installation.

1. In the Workbench toolbar, select **File > New > Project**.

The **New Project** wizard appears.



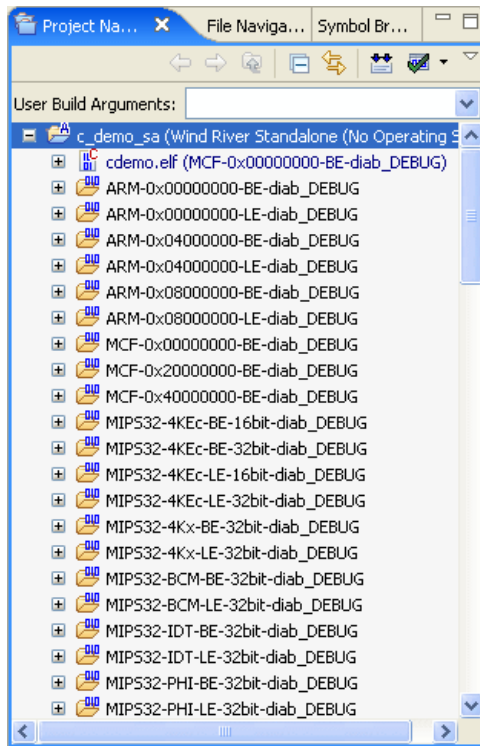
2. Expand the **Examples** folder and select **Standalone Sample Project**.
3. Click **Next**.
A sample project template appears.



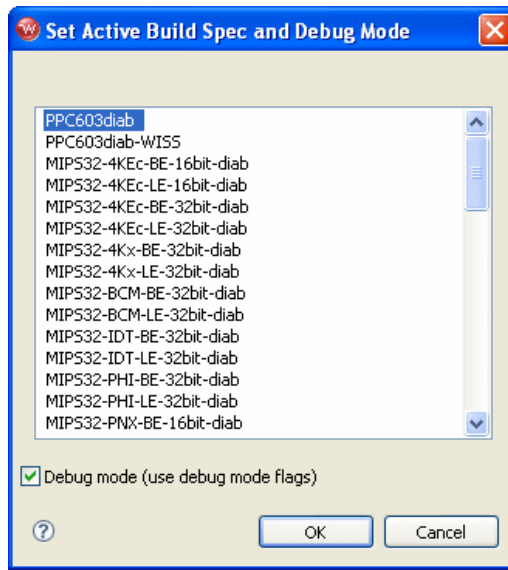
4. Select **C Demonstration Program** and click **Finish**.

Workbench creates the sample project in the default workspace folder and opens the **Application Development** perspective.

5. In the **Project Navigator** view, expand the **c_demo_sa** project.

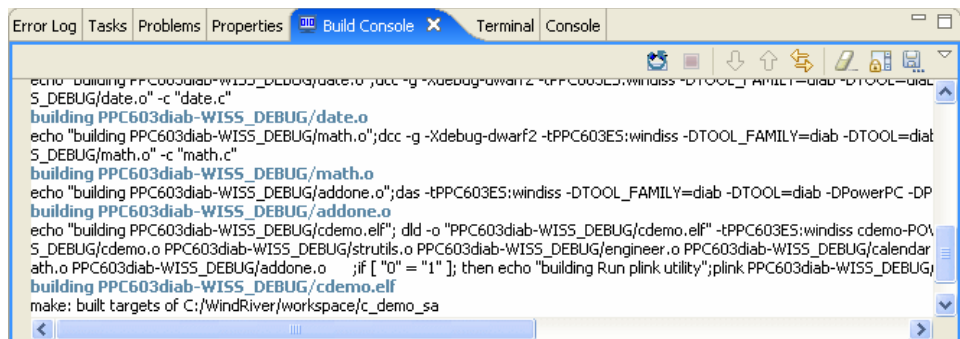


6. To build the sample project for use with a PowerPC target, right-click on the `c_demo_sa` top-level folder and select **Build Options > Set Active Build Spec**.
The **Set Active Build Spec and Debug Mode** dialog appears.



7. Scroll to the top and highlight **PPC603diab**.
8. Select **Debug mode (use debug mode flags)** so Workbench will generate symbolic debug information.
9. Click **OK**.
10. Right-click on the project name and select **Rebuild Project**.

Workbench builds the sample project. The results of the project build appear in the **Build Console** view.

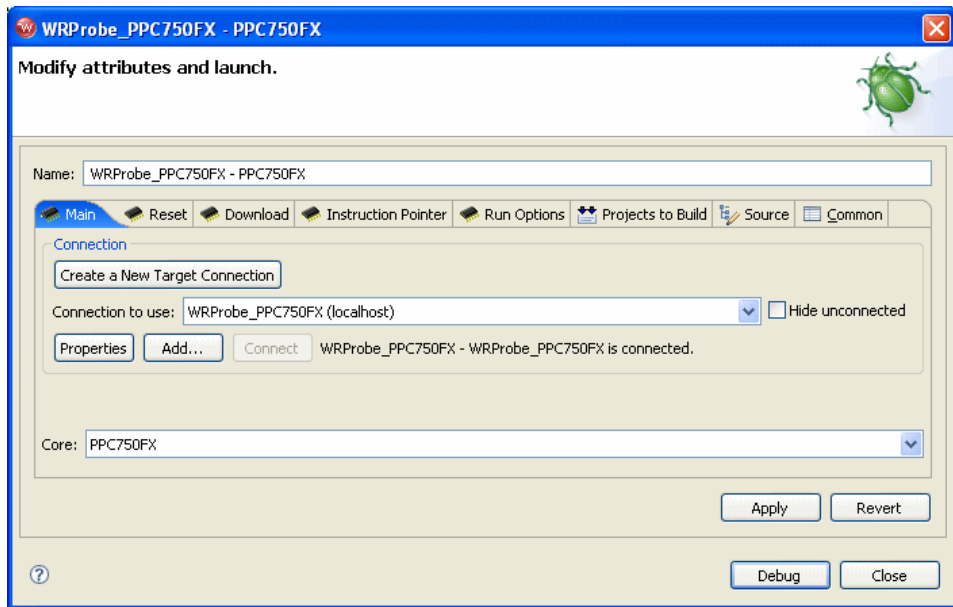


16.3.1 Downloading the Sample Code

To run the sample code, use the following steps:

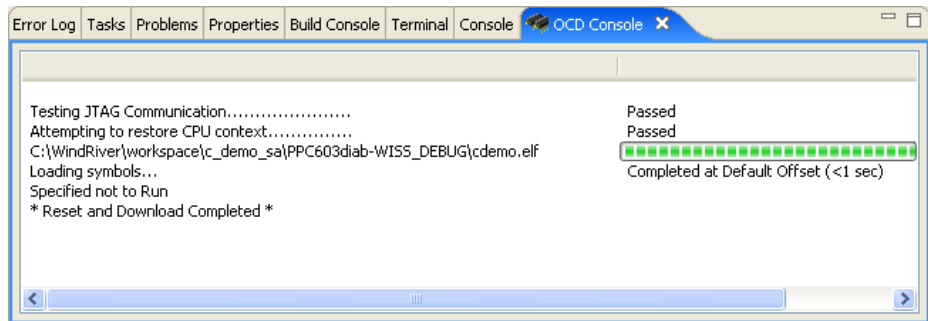
1. In the **Target Manager**, highlight the target connection name **WRProbe_PPC750FX**.
2. In the **Project Navigator** view, right-click on **cdemo.elf** and select **Reset and Download**.

The **Reset and Download** view appears.



3. Leave all settings at their defaults and click **Debug**.

The **OCD Console** view opens.



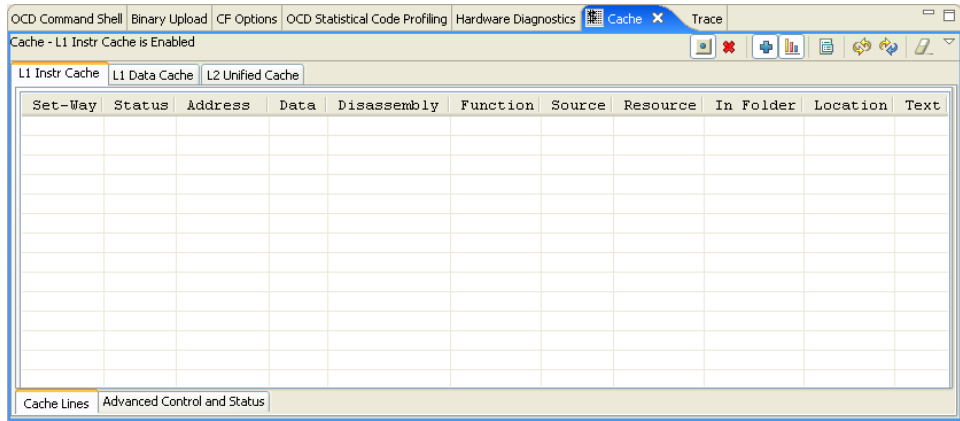
The **OCD Console** view shows the progress of the download operation.

16.4 Examining Cache

Use the instructions in this section to examine cache.

16.4.1 Instruction Cache

1. In the Workbench toolbar, select **Window > Show View > Cache**.
2. Click on the **Instr Cache** tab.
3. Click the **Enable Instr Cache** toggle button.

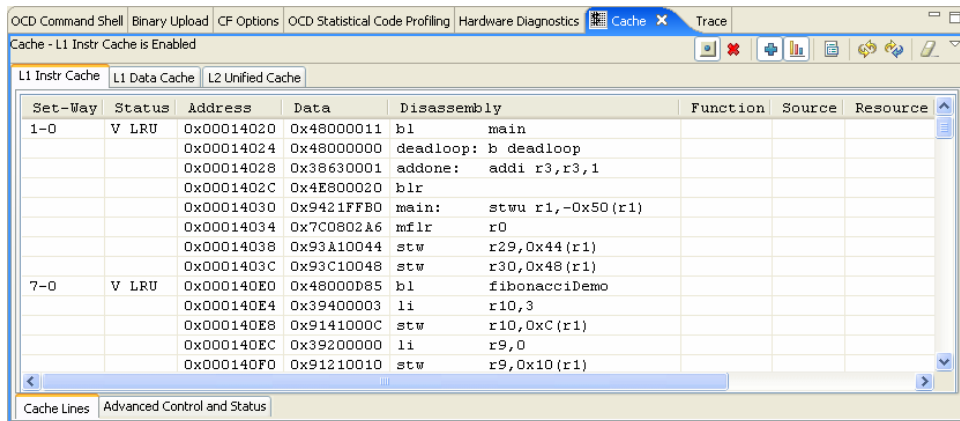


4. In the **Debug** view, click **Resume**.
5. Let the code run and then click **Suspend**.
6. In the **Cache** view, click **Refresh**.



NOTE: The cache view has two refresh buttons: **Refresh** and **Refresh All**. **Refresh** updates the cache with only the 100 most recent instructions. **Refresh All** updates the entire cache. A **Refresh All** operation may take significantly longer.

The **Cache** view populates with the instructions stored in cache while your code was running.



whether valid or invalid. To see only valid instructions and data, click the **Valid** button in the **Cache** view. This flushes all invalid information and shows only valid instructions and data.

To mark all instructions or data currently in the cache as invalid, click the **Invalidate** button in the **Cache** view. This does not disable the cache.

This column also shows the algorithm the cache uses to return cached information to RAM. If a cache has limited storage, which it usually does, information will have to be periodically ejected to make room for a new entry. The decision on what to eject is handled by a heuristic algorithm, the *replacement policy*. A popular replacement policy is **LRU**, which replaces the Least Recently Used entry.

Address

This column displays the address in RAM to which this cache line corresponds.

Data

This column displays the bits of data the cache line contains. This column only populates if you select Data Mode by clicking the **Data** button in the **Cache** view.

Disassembly

This column displays the operation codes that the data in the **Data** column represent (if any.) This column only populates if you select Data Mode by clicking the **Data** button in the **Cache** view.

Function

This column displays the associated function for each instruction in the **Cache** view. This column is only visible in the instruction cache. It only populates if you select Source Mode by clicking the **Source** button in the **Cache** view.

Source

This column displays the source for each instruction in the **Cache** view. This column is only visible in the instruction cache. It only populates if you select Source Mode by clicking the **Source** button in the **Cache** view.

Text

This column displays ASCII strings that the data in the **Data** column represent (if any.)

To disable the cache, select the **Instr Cache** tab or the **Data Cache** tab and click the **Enable/Disable** toggle button again.

To clear the contents of the **Cache** view, click the **Clear** button.

16.5 Viewing Cache Source

Source Mode is only available in the instruction cache.

With Source Mode enabled, the instruction cache shows the function and source of each instruction in the **Cache** view, under the headings **Function** and **Source**.

To see the contents of a cache line in source, use the following steps:

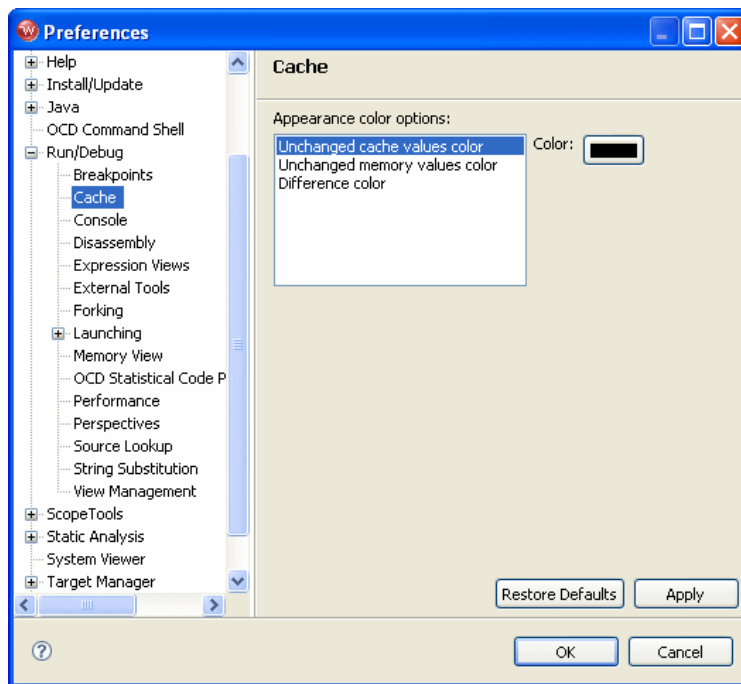
1. Select the **Instruction Cache** tab.
2. Click the **Source** button to enable Source Mode.
3. Right-click on an instruction and select **Go To**.

Workbench brings up the source of the instruction in the editor.

16.6 Comparing Memory

The **Cache** view allows you to compare the information in the cache to information stored in RAM.

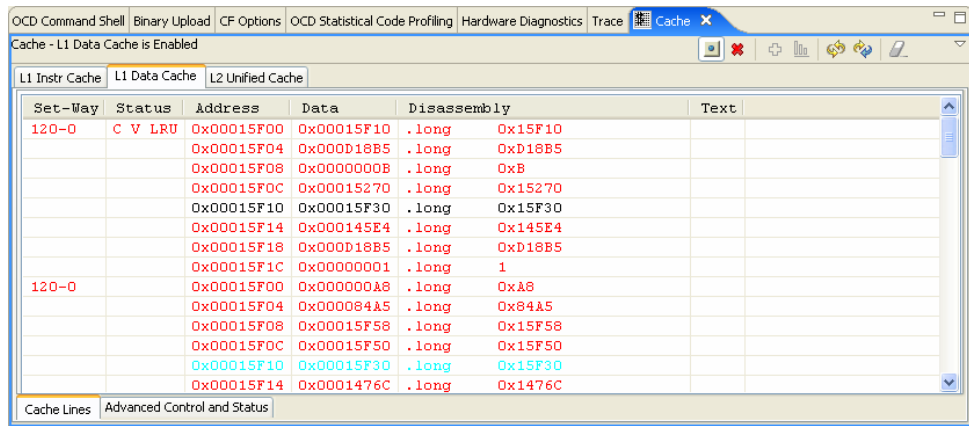
1. Right-click in the **Cache** view and select **Display Cache Properties**.
The **Preferences** dialog appears.



Use the **Preferences** dialog to set the color of the data stored in cache.

1. Click on **Unchanged cache values color**.
2. Click the **Color** button.
3. On the color palette, choose the color black.
4. Click on **Unchanged memory values color**.
5. On the color palette, choose the color blue.
6. Click on **Difference color**.
7. On the color palette, choose the color red.
8. Click **OK**.
9. Open the drop-down menu in the upper right corner of the **Cache** view and select **Compare Memory**.

Workbench displays any differences between cache and memory in red.



16.7 Reconfiguring the Cache

If you are using a target board that has programmable variable cache, you can specify changes in the **Cache** view.

1. Program the cache with a user-supplied file or program.
2. Open the drop-down menu in the upper right corner of the **Cache** view and select **Reconfigure** for your changes to take effect.



NOTE: If you are using a target board with fixed cache, choosing the **Reconfigure** option will have no visible results.

16.8 Exporting Cache Information

To export the information in the **Cache** view to a text file, right-click in the **Cache** view and select **Export**.

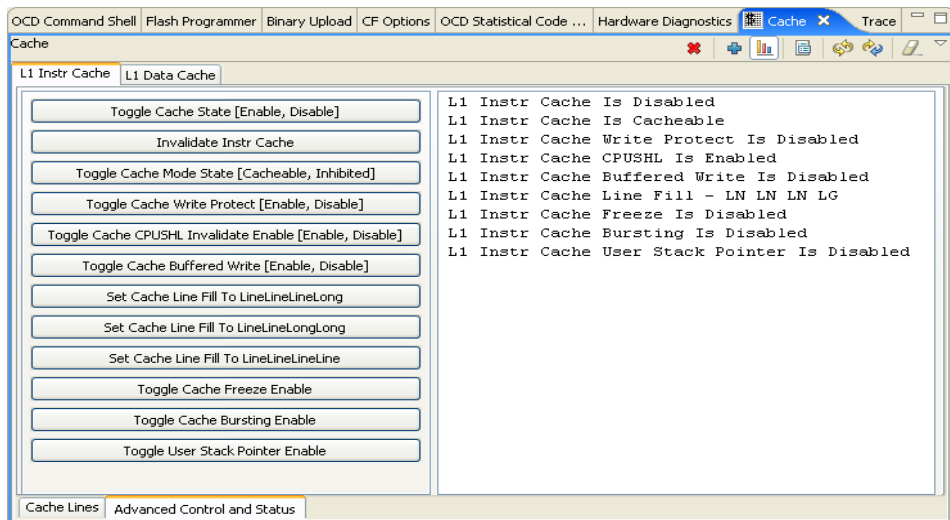
You can also export only selected parts of the information in the **Cache** view by highlighting the information you choose, right-clicking in the **Cache** view, and selecting **Export Selected**.

16.9 Using Processors Without Cache Lines

If you are connected to a processor that does not organize its cache in cache lines, you can manipulate cache using the **Advanced Control and Status** tab. The **Advanced Control and Status** tab does not display the contents of cache; it only allows you to perform operations on it. What operations are available varies by target processor.

Instruction Cache

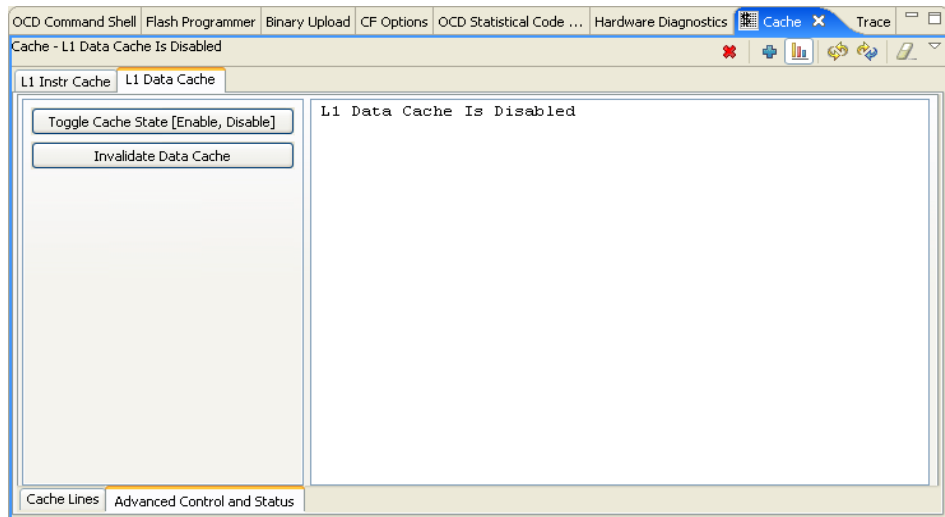
1. In the Workbench toolbar, select **Window > Show View > Cache**.
2. Select the **Instr Cache** tab.
3. Select the **Advanced Control and Status** tab.



The **Advanced Control and Status** tab shows available cache operations for the connected processor. This example shows the available instruction cache operations for a ColdFire MCF5208 processor.

Data Cache

1. Select the **Data Cache** tab.
2. Select the **Advanced Control and Status** tab.



This example shows the available data cache operations for a ColdFire MCF5208 processor.

17

Uploading Target Memory to a Binary File

- 17.1 Introduction 309
- 17.2 Uploading Memory 309
- 17.3 Comparing Memory 311

17.1 Introduction

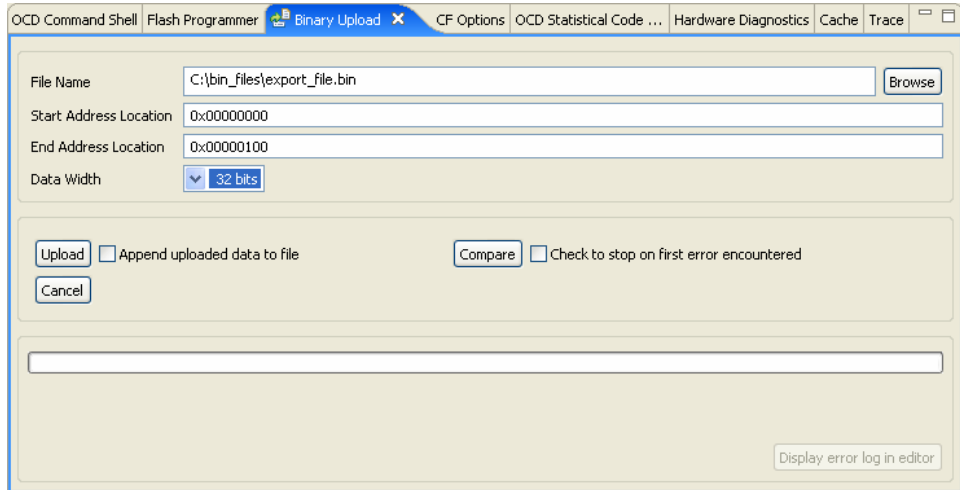
The **Binary Upload** view allows you to upload segments of target memory into a raw binary file. For example, you can use this view to back up your boot loader parameters to a file so they can be reflashed on other boards, or to back up a running boot loader prior to an upgrade. You can also use the view to compare the contents of a raw binary file with target memory.

17.2 Uploading Memory

To use the **Binary Upload** view, use the following steps:

1. In the Workbench toolbar, select **Window > Open Perspective > On Chip Debug**.

2. In the Workbench toolbar, select **Window > Show View > Binary Upload**.
The **Binary Upload** view appears.



3. In the **File Name** field, specify the file to which you want the data to be uploaded.
If the file does not already exist, Workbench creates it in the location you specify.
4. In the **Start Address Location** field, enter the start address of the memory segment you want to upload.
5. In the **End Address Location** field, enter the end address of the memory segment you want to upload.
6. In the **Data Width** field, specify the data bus width for the upload operation.
For example, if you select **32 bits**, Workbench uploads the data to your file in a series of 32-bit memory reads.
7. Click **Upload**.
This uploads the specified memory segment to your binary file. To cancel the upload, click **Cancel**.

By default, Workbench overwrites data in your binary file every time you click **Upload**. To append data to the file instead of overwriting it, select the **Append uploaded data to file** checkbox before you click **Upload**.

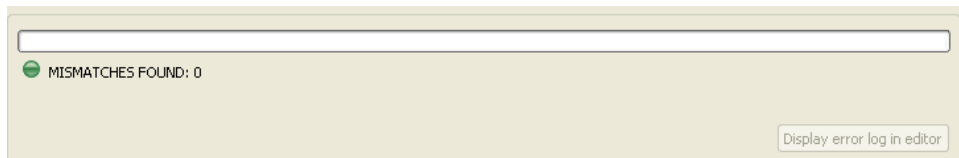
17.3 Comparing Memory

You can use the **Binary Upload** view to compare the contents of a raw binary file with memory, in order to detect memory corruption; for example, to check that a region of flash memory has not been modified by incorrect execution.

1. Make sure the **Start Address** and **End Address** locations are the same locations you specified for the upload.
2. Click **Compare**.

Workbench compares the contents of the file with the memory at the specified location. Any mismatches are displayed below the progress bar in the **Binary Upload** view.

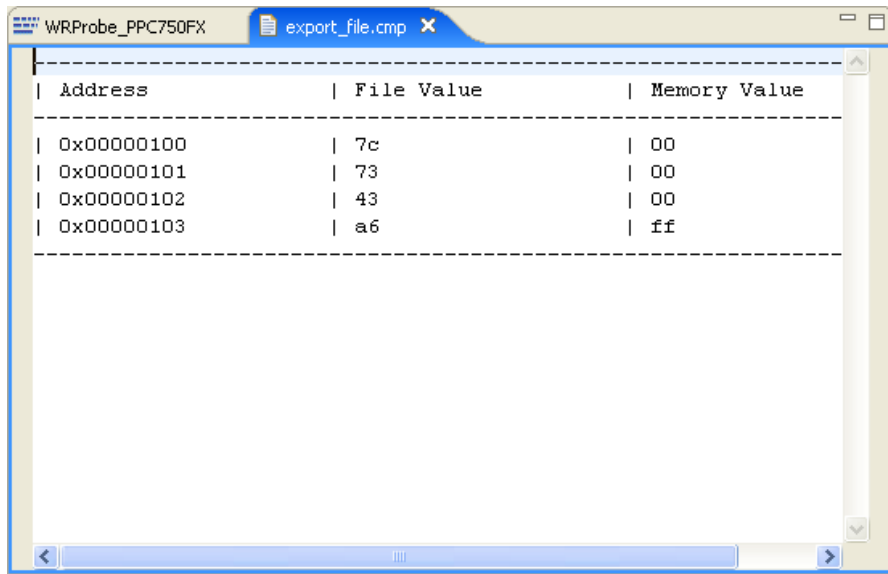
If the contents of memory and your file are the same, Workbench shows the following result:



Suppose the value at 0x00000100 has changed since you uploaded the file. Workbench shows the following result:



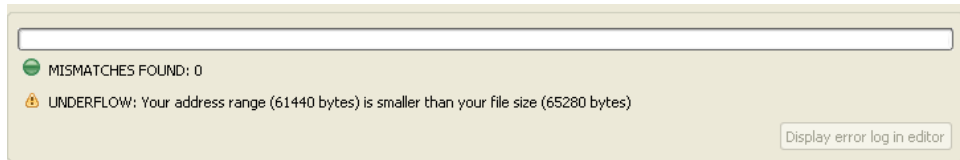
3. If you want Workbench to stop the compare operation the first time it finds a mismatch, select the **Check to stop on first error encountered** checkbox.
4. To see the difference between the memory and your file, click **Display error log in editor**.



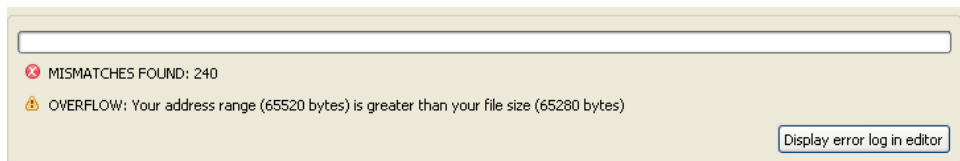
Address	File Value	Memory Value
0x00000100	7c	00
0x00000101	73	00
0x00000102	43	00
0x00000103	a6	ff

If you specify a different start or end address for your compare operation, Workbench returns one of the following results.

If the memory area you specify is smaller than the binary file:



If the memory area you specify is greater than the binary file:



18

Using the Instruction Set Simulator

[18.1 Introduction](#) 313

[18.2 Connecting to the Simulator](#) 314

18.1 Introduction

The Wind River Instruction Set Simulator (ISS) is a simulated hardware target for use in testing and prototyping. The simulator allows you to develop, run, and test applications on your host system, reducing the need for target hardware during development.



NOTE: Wind River Workbench supports the Instruction Set Simulator only for On-Chip Debugging functionality.

The ISS is also useful for demonstrations, as it allows you to run applications on your host system without needing an emulator or target processor.

The ISS has no input/output functionality, so certain Workbench views are not accessible when using it. The **Cache** view, for example, since the simulation has no cache; also the **Flash Programmer** view, since the simulation has no flash memory.

The Instruction Set Simulator runs a program by simulating the effects of each instruction on a target processor, one instruction at a time. Instead of generating an ordinary executable file, the ISS executes the entire instruction set.

The Instruction Set Simulator is not currently supported for ARM targets.

18.2 Connecting to the Simulator

To define a Wind River ISS connection, use the following steps:

First, open Workbench according to the method for your host computer.

Linux/Solaris Hosts

From your installation directory, issue the command

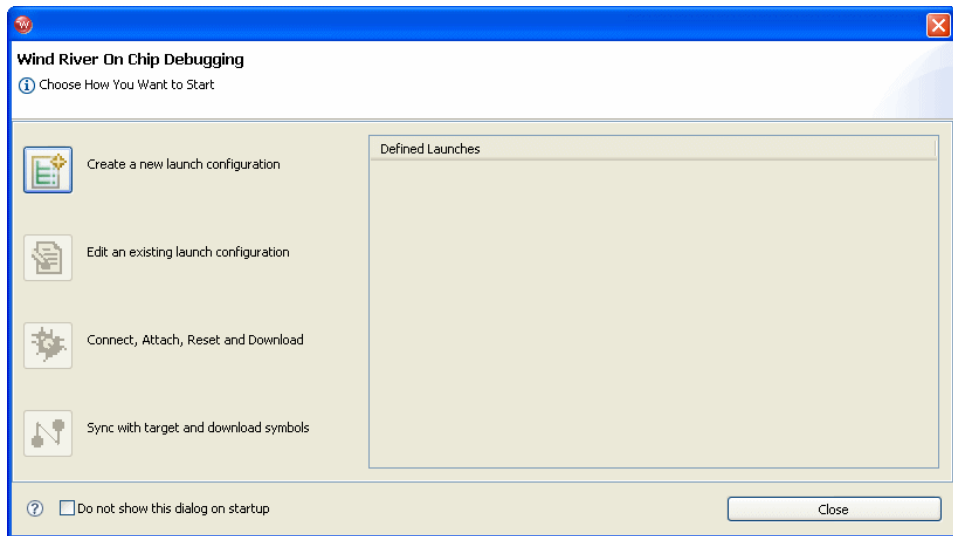
```
$ ./startWorkbench.sh
```

Windows Hosts

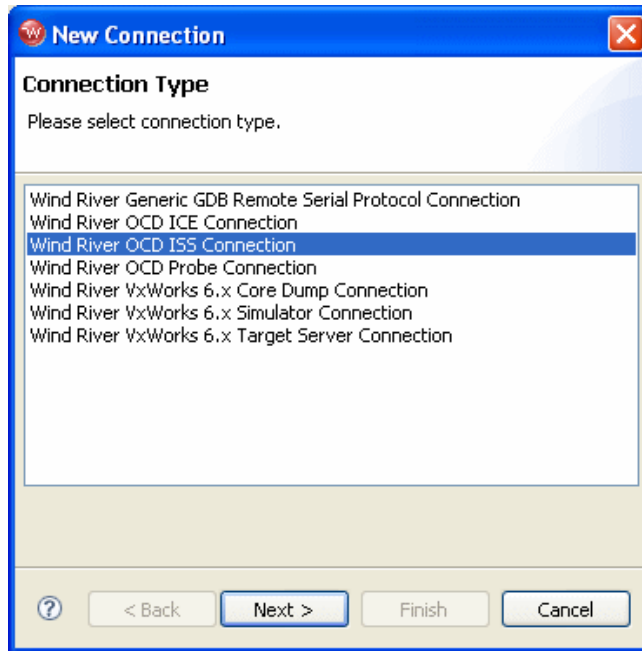
Select **Start > All Programs > Wind River > Wind River Workbench version**.

On Windows hosts, Workbench prompts you to specify a workspace location. Linux/Solaris hosts use the default location *installDir/workspace*.

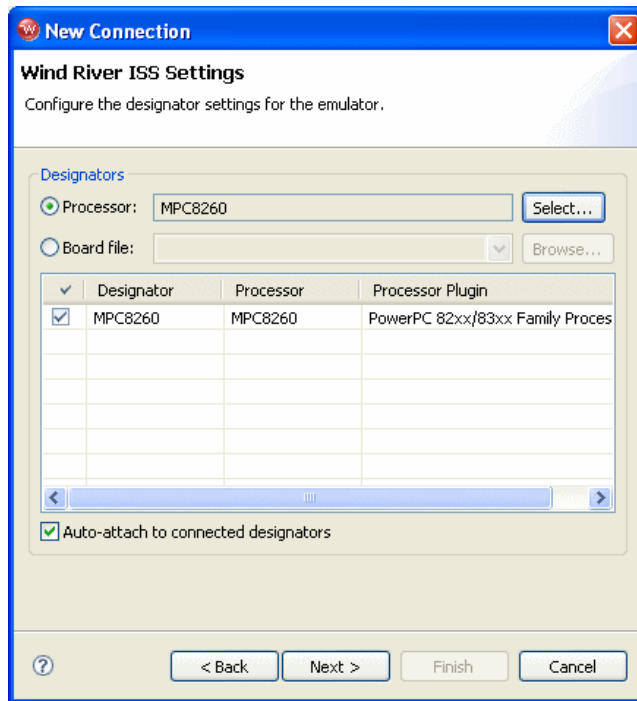
When Workbench opens, the **Quick Target Launch** dialog appears.



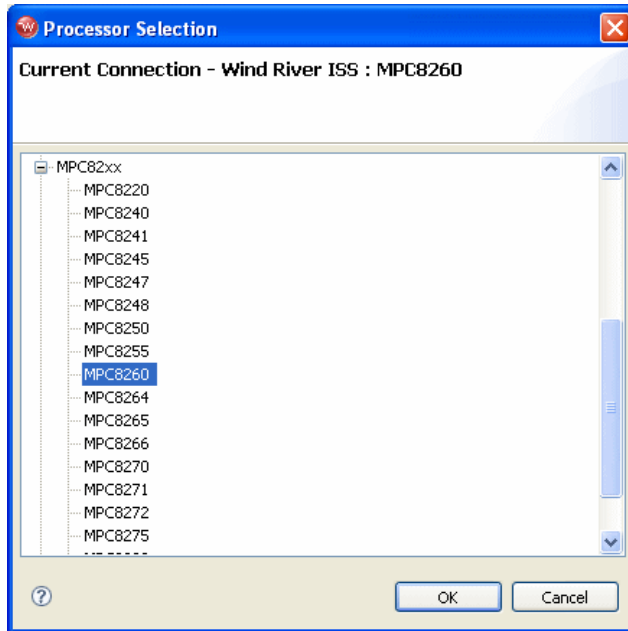
1. Select **Create a new launch configuration**.
The **Connection Type** dialog appears.



2. Select **Wind River OCD ISS Connection** and click **Next**.
The **Processor Selection** dialog appears.



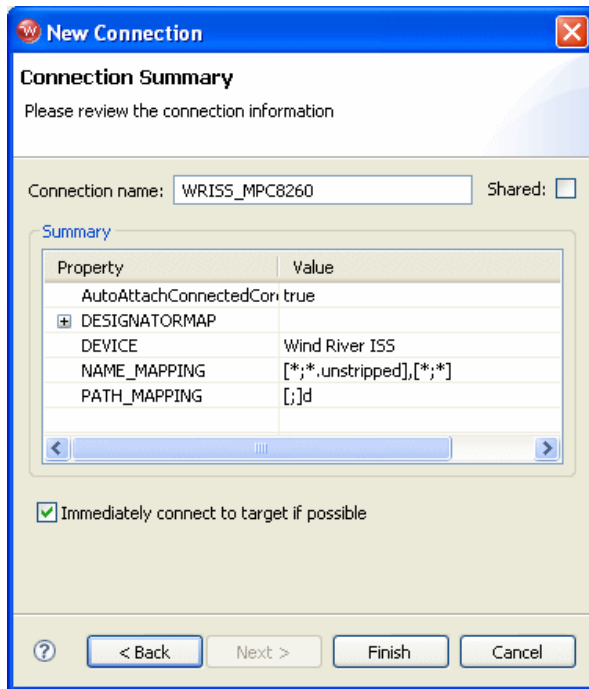
3. Click **Select**. From the list that appears, expand **MPC82xx** and select **MPC8260**.



4. Make sure the **Auto-attach to connected designators** checkbox is selected and click **OK**.

You are returned to the **Processor Selection** dialog.

5. Click **Next**.
6. The connection wizard passes through a number of screens that you do not need to configure, since you are not connecting to a real target. Leave all settings at their defaults and click **Next** until you come to the **Connection Summary**.



7. Make sure that the **Immediately connect to target if possible** checkbox is selected and click **Finish**.

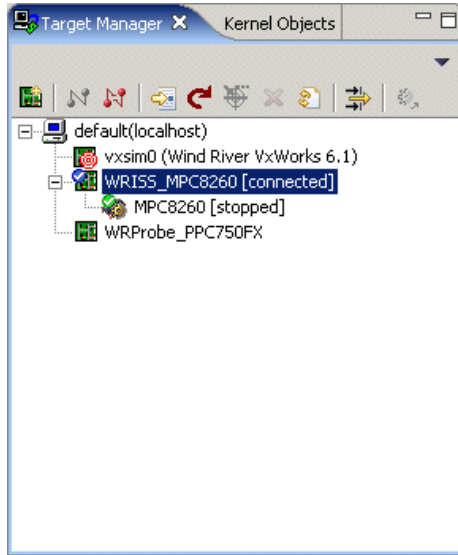
Workbench creates a target connection called **WRISS_MPC8260** in the **Target Manager** view.



NOTE: On Windows hosts, Workbench starts **WindISS.exe** and opens a command shell. Do not close this shell or terminate **WindISS.exe** while your target connection is running. Workbench automatically terminates **WindISS.exe** and closes the shell when you disconnect from the target connection.

Your connection is now visible in the **Target Manager**, as shown in [Figure 18-1](#).

Figure 18-1 ISS Connection



You can now download files and run application code through the simulator as you would if you were connected to a target.

19

Programming a VxWorks Boot ROM into Flash Memory

- [19.1 Introduction 321](#)
- [19.2 Configuring The Target 322](#)
- [19.3 Flashing the Boot ROM 324](#)

19.1 Introduction

This chapter describes how to use Wind River Workbench and your Wind River ICE SX or Wind River Probe to program a boot ROM into the flash memory on your target board.

The purpose of a boot ROM is to load a kernel image, set boot parameters, and pass control to the loaded kernel image.

Programming flash can be a complicated process, and is specific to your target board. If you are using a Wind River-supported target, all of the target-specific information is included in a file that is specific to your target. The file is located in your Workbench installation directory under the path

installDir/vxworks-6.x/target/config/yourTargetBoard/target.ref

The file includes flash addressing information, switch and jumper settings, and any other information that is specific to your target. If you are using a custom target, make sure you have detailed specifications about your target board. You

will need to know the size of your RAM workspace, where it is located, what type of flash device is on your target, and where it is located.

19.2 Configuring The Target

Make sure you have the following:

- A working Wind River ICE SX or Wind River Probe with all required cables.
- A target with a power supply (make sure the target is turned off before attaching the power supply).
- An Ethernet cable to connect your target to a LAN.
- A serial cable to connect your target to a host computer.
- Wind River Workbench installed on your host computer.

Depending on which target you are using, you may need to modify several switches and jumpers so that your board will work correctly. These switch and jumper settings are specific to the target you are using.

Hardware-specific information for each of the boards supported in this release is included in the file **target.nr** or **target.ref**, located in *installDir/vxworks-6.x/target/config/yourTargetBoard*. It includes all the switch and jumper settings that are required to flash and run the boot ROM. Familiarize yourself with the board-specific information and use the information provided in that section to configure the switches and jumpers on your target correctly.

If you are not using a Wind River-supported target board, make sure that the switches and jumpers are set to choose the flash device that you want to use and to use the proper clock frequency. If your target board has a switch that controls whether a debugger can attach to it, make sure that it is set to allow debugger control.

19.2.1 Making Physical Connections

The documentation for your Wind River ICE SX or Wind River Probe debugging tool explains how to make all physical connections correctly. For information about making the connection, and about applying power and establishing

communications with your target board, see the *Wind River ICE SX for Wind River Workbench Hardware Reference* or *Wind River Probe for Wind River Workbench Hardware Reference*.

In the Workbench toolbar, click **Window** and select **Show View > OCD Command Shell**.

If you have followed the instructions in your emulator's *Hardware Reference*, you should see a Background Mode (>BKM>) prompt appear in the **OCD Command Shell** once you have successfully established communications.

19.2.2 Testing Memory and Breakpoints

The flash programming algorithm needs to run on the target. This requires a RAM workspace, to which the algorithm will download, and breakpoints, which are used to stop an erase and program operation at completion.

Reading and Writing Memory

Once you have established communications with the target, use the following procedure to make sure you can write to and read from the target. In this example we assume that the RAM workspace is 0x00000000.



NOTE: A RAM workspace address of 0x00000000 is not appropriate for all targets. For Wind River-supported targets, you can find the necessary RAM workspace in your target's **target.ref** file, located in *installDir/vxworks-6.x/target/config/yourTargetBoard/target.ref*.

Wherever the RAM workspace is located on your target, you must make sure that memory is writable there.

At the >BKM> prompt, enter **dm 00000000** and press ENTER. Doing so displays the memory on your target at address 0.

Next, enter **sm 00000000 1234** and press ENTER to set the memory at address 0 to the value 1234. Enter **dm 00000000** to display the memory at that address again.

If you are communicating properly with your target, output is similar to that shown below:

```
>BKM>dm 00000000
00000000: FF7C EFFE FEFF E3FE 0D01 0FBE F0FD BFB6      .|.....
>BKM>sm 00000000 1234
>BKM>dm 00000000
00000000: 1234 EFFE FEFF E3FE 0D01 0FBE F0FD BFB6 .4.....
>BKM>
```

Testing Breakpoints

Occasionally, you may have difficulty programming flash memory on your target if software breakpoints are not being hit properly. Test this functionality before you continue. Some basic tests are provided with Wind River tools to test this functionality.

To use the test, enter the following commands at the **>BKM>** prompt in the **OCD Command Shell**:

```
>BKM>df e 0

>BKM>di 0 6
$00000000 : 0x60000000 :ppc nop
$00000004 : 0x60000000 :ppc nop
$00000008 : 0x60000000 :ppc nop
$0000000C : 0x60000000 :ppc nop
$00000010 : 0x7C0004AC :ppc sync
$00000014 : 0x4BFFFFFF0 :ppc b           0x4

>BKM>go 0
>RUN>dr pc
PC = 00000004
>RUN>dr pc
PC = 00000010
>RUN>sb 8

>RUN>

!BREAK! - [msg12000] Software breakpoint; PC = 0x00000008 [EVENT Taken]
>BKM>
>BKM>rb
>BKM>
```

19.3 Flashing the Boot ROM

Before you begin, make sure that a **>BKM>** prompt is visible in the **OCD Command Shell** in Workbench.

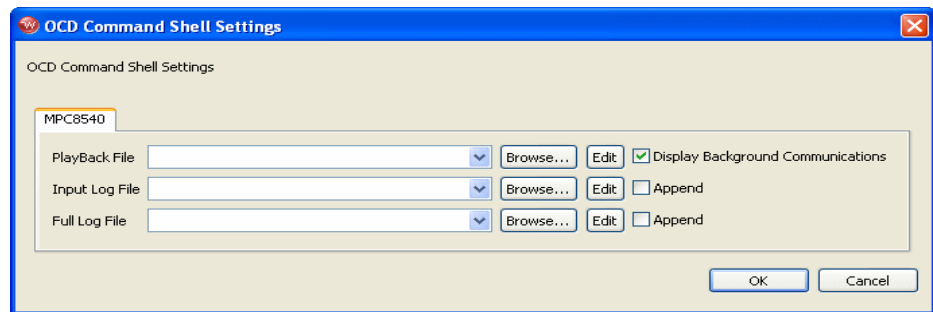
19.3.1 Playing a Register File

Play back the register file for your target board. The register file sets up access to memory and sets configuration options correctly (this can affect the setting of breakpoints). In some cases it also sets up the default flash algorithm.

Click on the **Playback** icon in the **OCD Command Shell**.

The **OCD Command Shell Settings** dialog appears, as shown in [Figure 19-1](#).

Figure 19-1 **OCD Command Shell Settings**



Next to the **PlayBack File** field, click **Browse**.

Browse for the appropriate register file for your target and click **OK**.

19.3.2 Setting Up Chip Select 0 and Programming the Reset Configuration Word

Depending on the target board you are using, you may need to program the reset configuration word and set up chip select 0 for your target. For example, most 82xx targets need to have the reset configuration word set correctly.

Not all targets use the reset configuration word and chip selects. Check *installDir/vxworks-6.x/target/config/yourTargetBoard/target.ref* for information about whether these are necessary for your target.

If you are not using a Wind River-supported target, you must determine the correct reset configuration word for the boot ROM. The reset configuration word is determined by the way you configure the boot ROM. Use a hex editor to open your boot ROM file and read the reset configuration word from that file. Your processor documentation should provide information about how to read the reset configuration word from that file.

19.3.3 Unlocking Flash

On some target boards, you may need to unlock Sector 0 of the flash memory before you can program flash. Some targets do this with a series of commands; some use jumpers or on-board switches; and some do not need to be unlocked. The data sheet for the flash device on your target board should tell you whether unlocking is necessary, and also supply the necessary settings or commands.

To run commands, click **Window** in the toolbar and select **Show View > OCD Command Shell**. Make sure a **>BKM>** prompt is visible in the **OCD Command Shell**.

For example, to unlock the flash chip on a Sandpoint 8245 target, you would type the following commands:

```
>BKM>sct picr1 fee00000 ff041a88
>BKM>sct errdr1 feee00003 00
>BKM>mml fec00000 4c580080
>BKM>mml fee00000 00130400
>BKM>mml fec00000 4c580080
>BKM>dml fee00000 1
FEE00000: 00130400      ....
>BKM>
```

Once you enter these commands, Sector 0 is unlocked and you can program the flash memory.

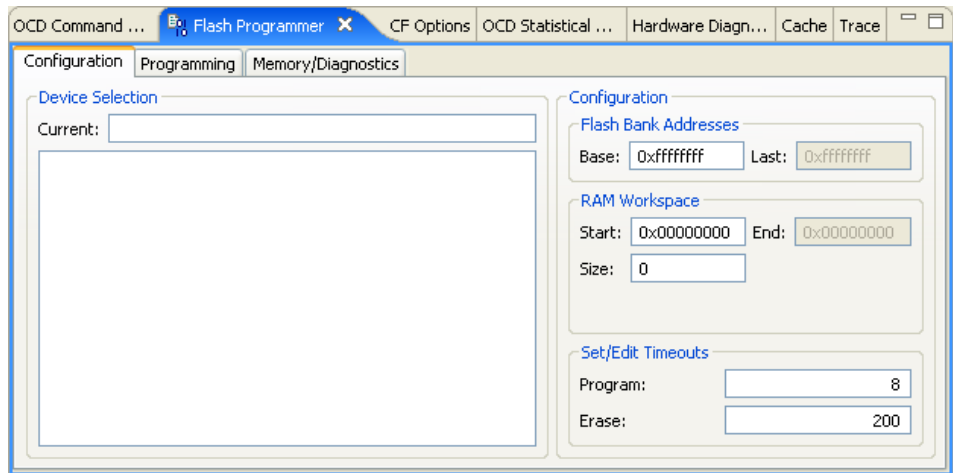
19.3.4 Programming Flash

Once your target is set up, you can program the boot ROM into flash memory on your target.

1. Open Wind River Workbench.
2. In the toolbar, select **Window > Open Perspective > Device Debug**.
3. In the toolbar, select **Window > Show View > Flash Programmer**.

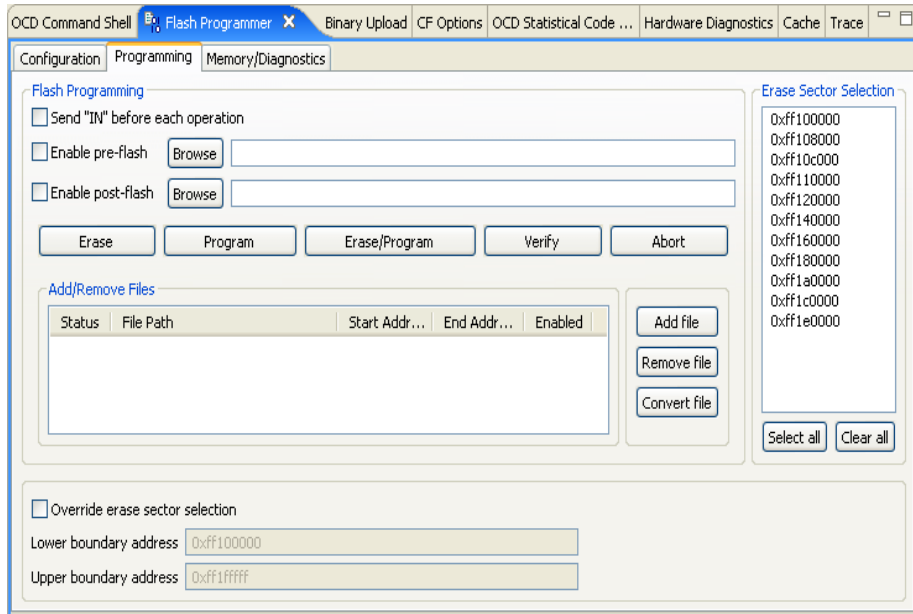
The **Flash Programmer** view appears, as shown in [Figure 19-2](#).

Figure 19-2 Flash Programmer View



4. Select the **Configuration** tab.
Use the **Configuration** tab to set the base address and workspace for your target board.
5. In the **Device Selection** area, select the correct flash device for your target.
The physical characteristics of your flash bank should be included in the board specification and schematics that came with your target board.
6. In the **Configuration** area, enter the base register for your target.
7. In the **RAM Workspace** area, set the **Start** and **Size** fields to the correct value for your target, as described in [9. Programming Flash Memory](#).
8. Select the **Programming** tab.
The **Programming** tab appears, as shown in [Figure 19-3](#).

Figure 19-3 Programming Tab



9. Click **Add File**.

A browser window appears.

10. Navigate to the boot ROM file for your target board.

The boot ROM files for Wind River-supported targets are located in *installDir/vxworks-6.x/target/config*.

11. In *installDir/vxworks-6.x/target/config*, find the directory for your target board.

This directory contains the **bootrom.bin** or **bootrom.hex** file for your target board.

12. Select the **bootrom.bin** file for your board and click **Open**. The filename appears in the **Add/Remove Files** area.

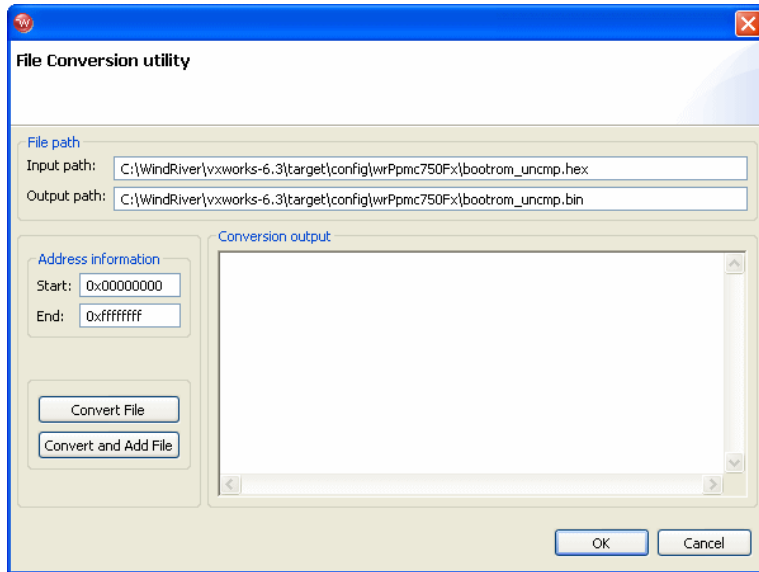
Workbench needs a **.bin** file to program flash. If your directory only contains a **.hex** file, you will need to convert it to **.bin** format by following Steps **a** through **d**. Otherwise, proceed to Step 13.

- a. Return to the **Programming** tab and click **Convert File**.

- b. Navigate to the folder for your target board and select the **bootrom.hex** file.

The **File Conversion Utility** dialog appears. [Figure 19-4](#) shows the **bootrom.hex** file for a PowerQUICC II board.

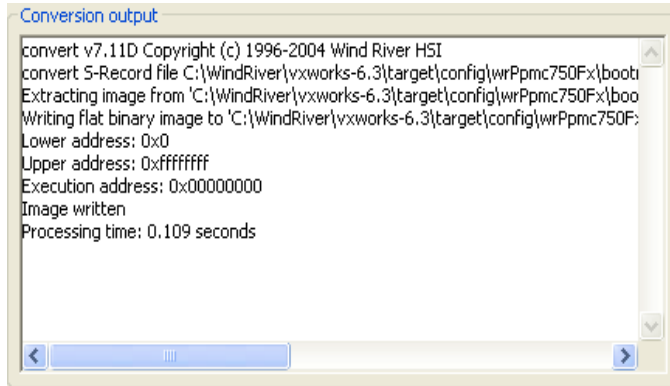
Figure 19-4 **File Conversion Utility**



- c. Click **Convert and Add File**.

The conversion output appears in the **Conversion Output** field.

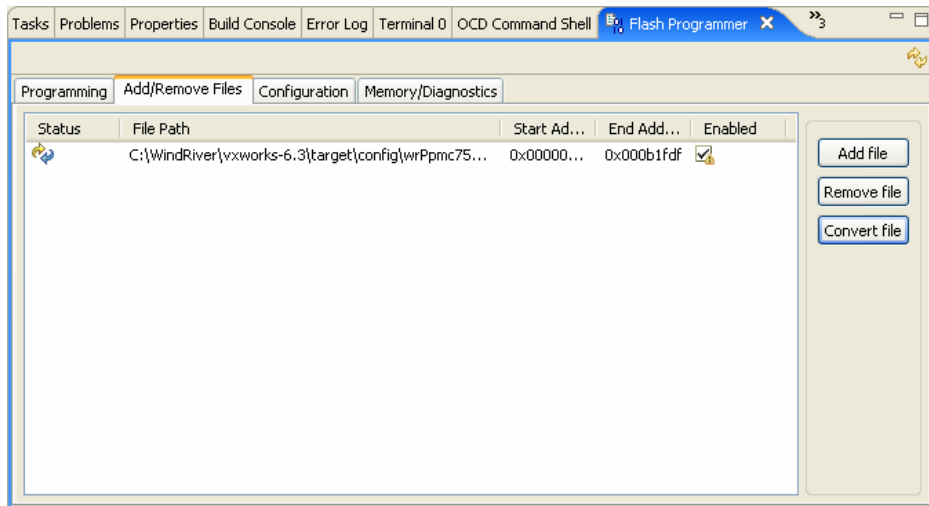
Figure 19-5 **Conversion Output**



d. Click **OK**.

The converted file appears in the **Add/Remove Files** tab.

Figure 19-6 **File Added to Add/Remove Files Tab**



13. Click on the value under the **Start Address** heading to highlight it.

Before you program the file into flash, you need to set a memory offset bias so the boot ROM will begin at the base address of your flash bank.

The necessary bias can be found in
installDir/vxworks-6.x/target/config/yourTargetBoard/target.ref.

14. Select the check box under the **Enabled** heading to enable the file.
15. Click on the **Configuration** tab.
16. In the **Configuration** tab, click **Select All**.

This selects all memory sectors for erasure. Wind River recommends that you do a full erase when programming a boot ROM.

17. Click on the **Programming** tab.
18. Verify that the **Flash Driver**, **Flash Bank**, and **Workspace** fields in the **Flash Settings** area of the **Programming** tab are set to the correct values.
19. Click **Erase/Program**.

Workbench erases the flash memory on your target and programs the flash with your boot ROM image. When it completes, the status bar at the bottom of the **Programming** tab shows that the download is complete.

20. Click the **Memory/Diagnostics** tab.
21. Click **Refresh**.

The contents of the flash memory with the boot ROM image loaded appear.

At this point, your boot ROM image is successfully programmed into the flash memory on your target.

After you have programmed the boot ROM into flash, set the program counter at the first instruction and run and debug it as you would any other program. For information on how to do this, see the *Wind River Workbench User's Guide*.

20

Programming a Linux Bootloader into Flash Memory

- 20.1 Introduction 333
- 20.2 Installing the Bootloader 334
- 20.3 Configuring and Building the Bootloader 334
- 20.4 Configuring the Target 338
- 20.5 Flashing the Bootloader 340

20.1 Introduction

This chapter describes how to use Wind River Workbench and your emulator to program a Linux bootloader into the flash memory on your target board.

The purpose of a bootloader is to initialize target hardware, load the Linux kernel image, set boot parameters, and pass control to the loaded Linux image.

There are many different bootloaders available, and different bootloaders (and different versions of the same bootloader) support different target boards. For demonstration purposes this chapter uses **uboot**, a bootloader commonly used for PowerPC targets.

Programming flash can be a complicated process, and is specific to your target board. If you are using a Wind River-supported target, all of the target-specific information is included in a **target.ref.linux** file that is specific to your target. The file includes flash addressing information, switch and jumper settings, and any

other information that is specific to your target. To find the **target.ref.linux** file for your target, go to <http://www.windriver.com/support>.

If you are using a custom target, make sure you have detailed specifications about your target board. You will need to know the size of your RAM workspace, where it is located, what type of flash device is on your target, and where it is located.

This chapter provides detailed instructions about programming **uboot** into the flash memory on your target board using Workbench.

20.2 Installing the Bootloader

Download the bootloader you wish to use. For example, **uboot** is available for download at <http://sourceforge.net/projects/u-boot/>.



NOTE: Different versions of U-Boot are supported for different kernel versions and architecture types. Make sure you download the correct version of **uboot** for your target board.

To unpack the file and install the bootloader on your host, enter the following commands:

```
$ tar xjf bootloader_file
```

For example, if you downloaded **u-boot-1.1.1.tar.bz2**, type:

```
$ tar xjf u-boot-1.1.1.tar.bz2
```

This command unpacks the files and creates the bootloader directory structure.

Change directories so that you are in the new bootloader directory.

20.3 Configuring and Building the Bootloader

Wind River provides a website with detailed information about building and configuring a bootloader. This section provides a brief overview about how to

configure and build **uboot**. For more detailed information, go to <http://www.windriver.com/support>.

Wind River Workbench supports the bootloader in two different ways.

- **Patching the Standard Installation**

Wind River includes **uboot** patches for some of the supported targets at <http://www.windriver.com/support>. To configure and build a bootloader using a Wind River patch, follow the instructions provided there.

- **Manually Modifying the boardConfig.h file**

Some of the supported targets require only minor modification to the standard bootloader configuration files.

For boards supported in this manner, the following section describes how to modify the configuration file in a standard installation for your target.

20.3.1 Configuring and Building the Bootloader Manually

Modifying the boardConfig.h File

To create a **uboot** image that you can download to your target, **uboot** uses a *boardConfig.h* file, which contains specific hardware settings. The name of the *boardConfig.h* file is specific to your target and is located in the *ubootInstallDir/include/configs* directory.

If your specific board settings do not match the defaults in the *boardConfig.h* file, edit the file appropriately.

The steps in this section describe how to edit the file for your target.



NOTE: The bootloader initializes your target to run a Linux kernel. Any errors in **uboot** configuration may prevent the kernel from booting correctly or may cause it to fail later.

1. From the top level of the U-Boot directory, type the following:

```
$ cd include/configs
```

2. Open the *boardConfig.h* file for your target.

All hardware options in this file must match your target. The next steps describe specific items that may need to change for your target.



NOTE: The items described are not a comprehensive list. Review the contents of the entire file carefully and make any changes that are required for your target.

- **Clock Speed**

Verify that the configuration information pertaining to clock speed (oscillator frequency) is correct for your target.

For example, the **sbc8260.h** file contains the following two lines:

```
#define CONFIG_8260_CLKIN (66 * 1000 * 1000)
```

and

```
#define CFG_SBC_MODCK_H 0x05
```

These values are correct for a 66 MHz target. If you are using a 33 MHz target, change the lines to read:

```
#define CONFIG_8260_CLKIN (33 * 1000 * 1000)
```

and

```
#define CFG_SBC_MODCK_H 0x01
```

- **Baud Rate**

Configure the baud rate to a value that works for your system. The console will run faster with a higher baud rate.

- **Ethernet MAC Address**

Change the Ethernet MAC address to the address of the Ethernet port on your target board.



CAUTION: Make sure that the Ethernet MAC address you choose is unique on your network. Duplicate Ethernet MAC addresses on the same network will cause problems when you try to load the kernel onto your target.

After you have examined the options and verified that they are correct for your target board, you can build a downloadable bootloader file for your target.

Building a Downloadable U-Boot File

Follow these steps to build a downloadable U-Boot file for your target:

1. Change directories until you are at the top level of your **uboot** directory.
2. Type the following command:

```
$ make distclean
```

This ensures that there are no build results remaining from any previous configurations.

3. Configure U-Boot for your target.

```
$ make ARCH=ppc CROSS_COMPILE=CrossCompilePrefix yourConfigString
```

The values of *CrossCompilePrefix* and *yourConfigString* are based on your processor and flash device.

For example, if you want to configure U-Boot to use the Wind River SBC 8260 target, enter the following:

```
$ make ARCH=ppc CROSS_COMPILE=ppc_82xx- sbc8260_config
```

4. Make the U-Boot downloadable image.

```
$ make ARCH=ppc CROSS_COMPILE=CrossCompilePrefix dep  
$ make ARCH=ppc CROSS_COMPILE=CrossCompilePrefix all
```

For example, using the Wind River SBC 8260 target, enter the following:

```
$ make ARCH=ppc CROSS_COMPILE=ppc_82xx- dep  
$ make ARCH=ppc CROSS_COMPILE=ppc_82xx- all
```

Once these commands finish executing, three new files are included in the **uboot** directory, as shown in:

- **u-boot**
- **u-boot.bin**
- **u-boot.srec**

The files generated during the build process are described in [Table 20-1](#).

Table 20-1 U-Boot Files

File Name	Description
u-boot	A .elf file image of uboot .
u-boot.bin	A flat binary executable image of uboot .
u-boot.srec	A standard Motorola format of uboot for flash programming.

u-boot, **u-boot.bin**, and **u-boot.srec** can be used to program the flash memory on your target. Use the file that works with Workbench and your emulator.

20.4 Configuring the Target

Make sure you have the following:

- A working emulator with all required cables.
- A target with a power supply (make sure the target is turned off before attaching the power supply).
- An Ethernet cable to connect your target to a LAN.
- A serial cable to connect your target to a host computer.
- Wind River Workbench installed on your host computer.

Depending on which target you are using, you may need to modify several switches and jumpers so that your board will work correctly. These switch and jumper settings are specific to the target you are using.

Hardware-specific information for each of the Wind River-supported boards in this release is included in the file **target.ref.linux**. Find the file specific to your board at <http://www.windriver.com/support>. It includes all the switch and jumper settings that are required to flash and run the bootloader. Familiarize yourself with the board-specific information and use the information provided in that section to configure the switches and jumpers on your target correctly.

If you are not using a supported target board, make sure that the switches and jumpers are set to choose the flash device that you want to use and to use the proper clock frequency. If your target board has a switch that controls whether a debugger can attach to it, make sure that it is set to allow debugger control.

20.4.1 Making Physical Connections

The documentation for your Wind River ICE SX or Wind River Probe debugging tool explains how to make all physical connections correctly. For information about making the connection, and about applying power and establishing communications with your target board, see the *Wind River ICE SX for Wind River Workbench Hardware Reference* or *Wind River Probe for Wind River Workbench Hardware Reference*.

In the Workbench toolbar, select **Window > Show View > OCD Command Shell**.

If you have followed the instructions in your emulator's Hardware Reference, you will see a Background Mode (>BKM>) prompt appear in the **OCD Command Shell** once you have successfully established communications.

20.4.2 Testing Memory and Breakpoints

The flash programming algorithm needs to run on the target. This requires a RAM workspace, to which the algorithm will download, and breakpoints, which are used to stop an erase and program operation at completion.

Reading and Writing Memory

Once you have established communications with the target, use the following procedure to make sure you can write to and read from the target. In this example we assume that the RAM workspace is 0x00000000.



NOTE: A RAM workspace address of 0x00000000 is not appropriate for all targets. For Wind River-supported targets, you can find the necessary RAM workspace in your target's **target.ref.linux** file, located at <http://www.windriver.com/support>.

Wherever the RAM workspace is located on your target, you must make sure that memory is writable there.

At the **>BKM>** prompt, enter **dm 00000000** and press **ENTER**. Doing so displays the memory on your target at address 0.

Next, enter **sm 00000000 1234** and press **ENTER** to set the memory at address 0 to the value 1234. Enter **dm 00000000** to display the memory at that address again.

If you are communicating properly with your target, output is similar to that shown below:

```
>BKM>dm 00000000
00000000: FF7C EFFE FEFF E3FE 0D01 0FBE F0FD BFB6      .|.....
>BKM>sm 00000000 1234
>BKM>dm 00000000
00000000: 1234 EFFE FEFF E3FE 0D01 0FBE F0FD BFB6  .4.....
>BKM>
```

Testing Breakpoints

Occasionally, you may have difficulty programming flash memory on your target if software breakpoints are not being hit properly. Test this functionality before you continue. Some basic tests are provided with Wind River tools to test this functionality.

To use the test, enter the following commands at the **>BKM>** prompt in the **OCD Command Shell**:

```
>BKM>sy prog1 0

>BKM>di 0 6
$00000000 : 0x60000000 :ppc nop
$00000004 : 0x60000000 :ppc nop
$00000008 : 0x60000000 :ppc nop
$0000000C : 0x60000000 :ppc nop
$00000010 : 0x7C0004AC :ppc sync
$00000014 : 0x4BFFFFFF0 :ppc b           0x4

>BKM>go 0
>RUN>dr pc
PC      = 00000004
>RUN>dr pc
PC      = 00000010
>RUN>sb 8

>RUN>

!BREAK! - [msg12000] Software breakpoint; PC = 0x00000008 [EVENT Taken]
>BKM>
>BKM>rb
>BKM>
```

20.5 Flashing the Bootloader

Before you begin:

- Copy the **u-boot.bin** file to your host computer.
- Make sure that a **>BKM>** prompt is visible in the **OCD Command Shell** in Workbench.

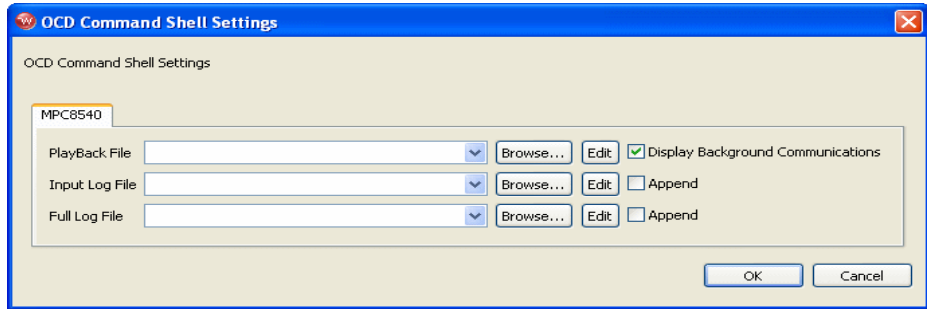
20.5.1 Playing a Register File

Play back the register file for your target board. The register file sets up access to memory and sets configuration options correctly (this can affect the setting of breakpoints.) In some cases it also sets up the default flash algorithm.

In the **OCD Command Shell**, click **Playback**.

The **OCD Command Shell Settings** dialog appears, as shown in [Figure 20-1](#).

Figure 20-1 OCD Command Shell Settings



Next to the **PlayBack File** field, click **Browse**.

Browse for the appropriate register file for your target and click **OK**.

20.5.2 Setting Up Chip Select 0 and Programming the Reset Configuration Word

Depending on the target board you are using, you may need to program the reset configuration word and set up chip select 0 for your target. For example, most 82xx targets need to have the reset configuration word set correctly. Check your **target.ref.linux** file for information about the reset configuration word.

If you are not using a Wind River-supported target, you must determine the correct reset configuration word for the bootloader. The reset configuration word is determined by the way you configure the bootloader. Use a hex editor to open the **u-boot.bin** file that you generated, and read the reset configuration word from that file. Your Motorola processor documentation provides information about how to read the reset configuration word from that file.

20

20.5.3 Unlocking Flash

On some target boards, you may need to unlock Sector 0 of the flash memory before you can program flash. Some targets do this with a series of commands; some use jumpers or on-board switches; and some do not need to be unlocked. The data sheet for the flash device on your target board should tell you whether unlocking is necessary, and also supply the necessary settings or commands.

To run commands, select **Window > Show View > OCD Command Shell**. Make sure a **>BKM>** prompt is visible in the **OCD Command Shell**.

For example, on a Sandpoint 8245 target, you would type the following commands:

```
>BKM>sct picr1 fee00000 ff041a88
>BKM>sct errdr1 feee00003 00
>BKM>mml fec00000 4c580080
>BKM>mml fee00000 00130400
>BKM>mml fec00000 4c580080
>BKM>dml fee00000 1
FEE00000: 00130400    ....
>BKM>
```

Once you enter these commands, Sector 0 is unlocked and you can program the flash memory.

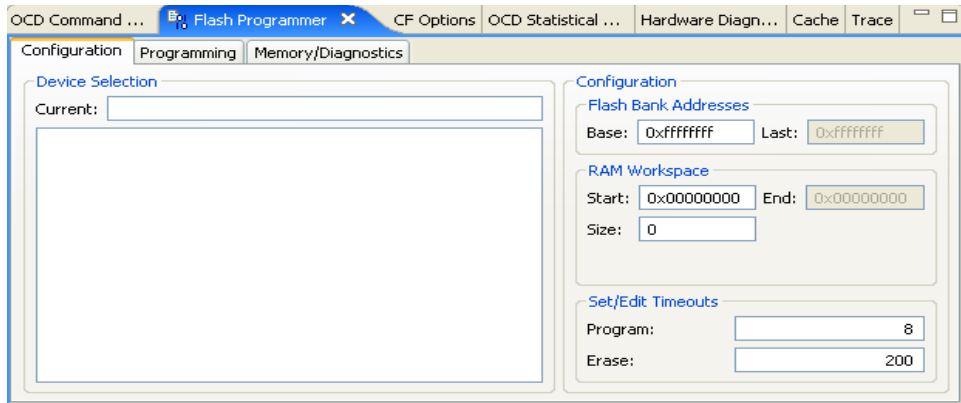
20.5.4 Programming Flash

Once your target is set up, you can program the flash memory on your target with the bootloader.

1. In the toolbar, select **Window > Show View > Flash Programmer**.

The **Flash Programmer** view appears.

Figure 20-2 **Flash Programmer View**



2. Select the **Configuration** tab.

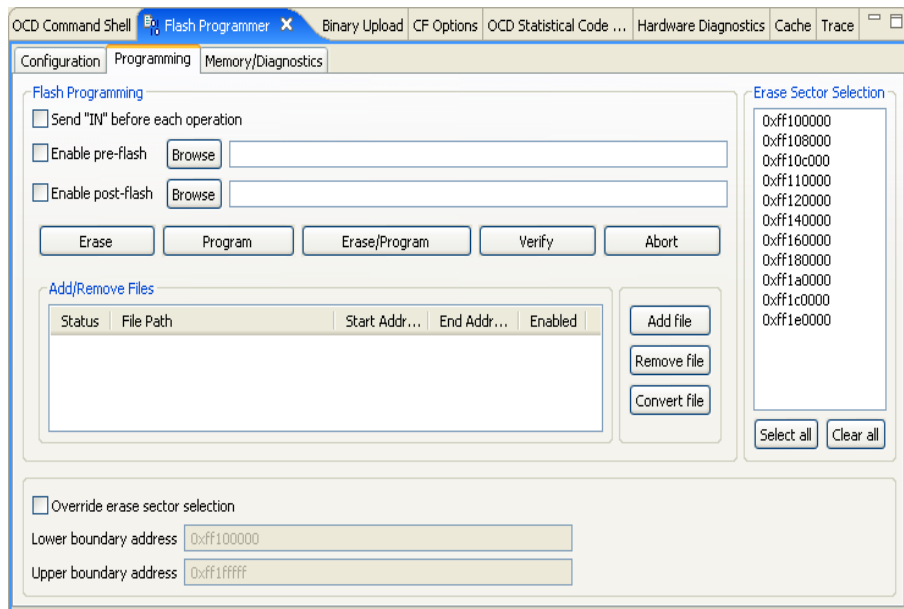
Use the **Configuration** tab to set the base address and workspace for your target board.

3. In the **Device Selection** area, select the flash device that is on your target.

4. In the **Configuration** area, enter the base register for your target.
5. In the **RAM Workspace** area, set the **Start** and **Size** fields to the correct value for your target, as described in 9. *Programming Flash Memory*.
6. Click on the **Programming** tab.

The **Programming** tab appears, as shown in Figure 20-3.

Figure 20-3 **Programming Tab**



7. Click **Add Files**.
A browser window appears. Browse for the **u-boot.bin** file and click **Open**. The file appears in the **Add/Remove Files** area of the **Programming** tab.
8. Click under the **Enable** heading to enable the file for downloading.
9. Verify that the **Flash Driver**, **Flash Bank**, and **Workspace** fields in the **Flash Settings** area of the **Programming** tab are set to the correct values.
10. Click **Erase/Program**.
Workbench erases the flash memory on your target and programs the flash with your **u-boot** image. When it completes, the status bar at the bottom of the **Programming** view states that the download is complete.

11. Click the **Memory/Diagnostics** tab.
12. Click **Refresh**.

The contents of the flash memory with the U-Boot image loaded appear.

At this point, your **uboot** image is successfully programmed into the flash memory on your target.

21

Downloading a Kernel Image Using a JTAG Connection

21.1 Introduction 345

21.2 Bypassing the Boot Line Address -- VxWorks 347

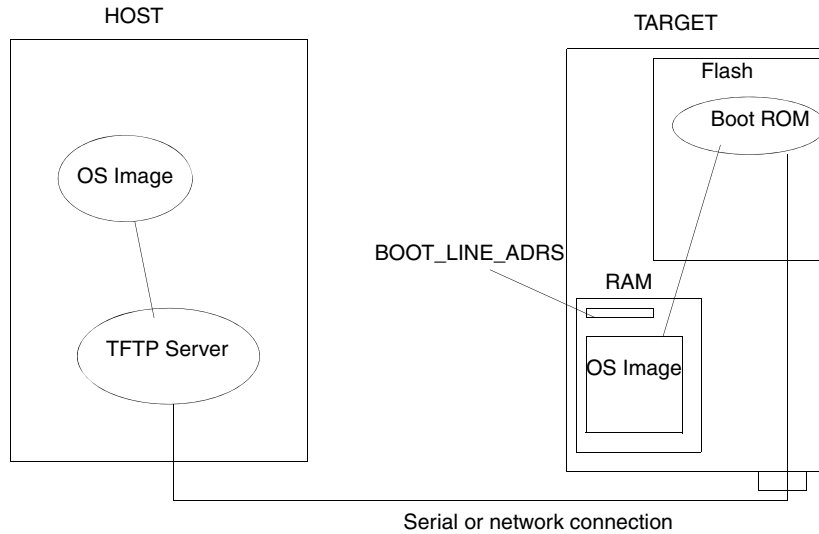
21.3 Bypassing the Boot Line Setup -- Linux 349

21.4 Downloading the Kernel Image 351

21.1 Introduction

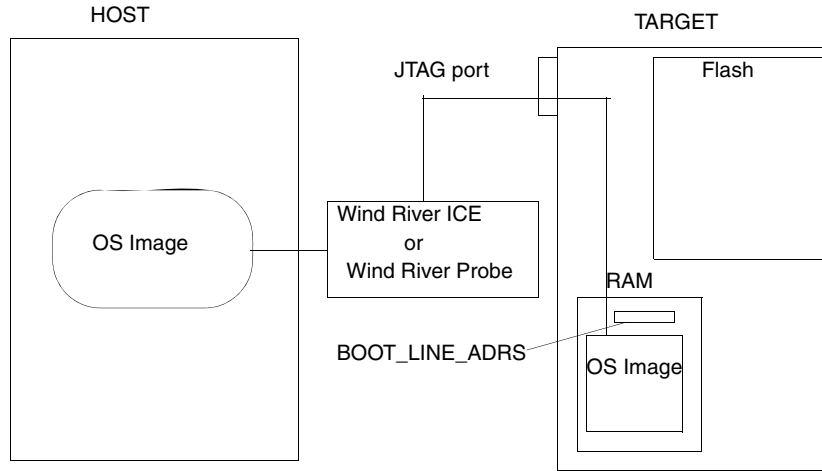
The function of a boot loader, whether a boot ROM for VxWorks or **uboot** or **grub** for Linux, is to initialize the hardware, set boot parameters, load a kernel image into RAM, and pass control to the loaded kernel image, as shown in [Figure 21-1](#).

Figure 21-1 Loading a Kernel Image with a Boot Loader



The Wind River ICE SX and Wind River Probe tools can also perform these functions through the JTAG port, as shown in [Figure 21-2](#). This can be useful for BSP or driver developers who may not have a boot ROM available or may be in the process of developing one.

Figure 21-2 Loading a Kernel Image with an Emulator



21.2 Bypassing the Boot Line Address -- VxWorks

Ordinarily, the boot ROM passes the boot parameters to the kernel image using an agreed-upon **BOOT_LINE_ADRS** memory location. For example, most PowerPC targets use the address 0x4200.

When the boot ROM retrieves and starts the kernel image through a TFTP server, the loaded image runs and uses the memory string at the **BOOT_LINE_ADRS** memory location as its boot parameters.

However, when an emulator loads the kernel image instead of a boot ROM, it does not set the boot parameters, and the **BOOT_LINE_ADRS** memory location probably contains uninitialized memory. The cracking of the bootline reads the random values until it finds a **0x00/null** string termination. This usually results in a message about an invalid boot line.

There are two ways of solving this problem:

- Manually setting the `BOOT_LINE_ADRS` location.
- Forcing the `DEFAULT_BOOT_LINE`.

21.2.1 Manually Setting the `BOOT_LINE_ADRS` Location

When setting up the boot parameters, the board looks at the beginning of the boot line memory location to crack the boot line, that is, to decode the boot parameter string into its individual fields: hostname, IP address, and so on.

When the value at the beginning of the boot line memory location is zero, the boot line cracking function will use the `DEFAULT_BOOT_LINE` value specified in the `config.h` file in the BSP.

Using a Wind River ICE SX or Wind River Probe, you can manually set the boot line memory location to zero before running the image, so the boot line cracking function will use the `DEFAULT_BOOT_LINE` value.

To manually set the boot line address to zero, do the following:

1. Make sure you have a `>BKM>` prompt in the **OCD Command Shell**.
2. At the `>BKM>` prompt, type `SM 4200 0` and press `ENTER`.



NOTE: `0x4200` is only one example; it is the standard address for PowerPC targets. To find the correct `BOOT_LINE_ADRS` memory location for your target, consult the manufacturer's documentation for your target processor.

21.2.2 Forcing the `DEFAULT_BOOT_LINE`

The other method is to force the default boot line by conditionally compiling in the `FORCE_DEFAULT_BOOT_LINE` construct. This construct is only available in certain Wind River-specific BSPs.

If the `FORCE_DEFAULT_BOOT_LINE` construct is available on your BSP, the `config.h` file will contain the following:

```
/*
 * If the FORCE_DEFAULT_BOOT_LINE is defined then the
 * DEFAULT_BOOT_LINE parameters are always used regardless of NVRAM
 * values specified at bootrom time. See target.nr for details.
 * This is usually used to debug downloaded images without a bootrom present.
 */

#define FORCE_DEFAULT_BOOT_LINE
```

Defining the `FORCE_DEFAULT_BOOT_LINE` construct informs the loaded kernel image that the `DEFAULT_BOOT_LINE` from the `config.h` file in the BSP should be used, regardless of the value at the `BOOT_LINE_ADRS` memory location.



CAUTION: Make sure you undefine the `FORCE_DEFAULT_BOOT_LINE` construct before you ship your product.

In the `sysHwInit()` `sysLib.c` file, the `FORCE_DEFAULT_BOOT_LINE` value is used to copy the default boot line information into the `BOOT_LINE_ADRS` location. The value of `DEFAULT_BOOT_LINE` is also specified in the `config.h` file.

```
#define FORCE_DEFAULT_BOOT_LINE
#ifdef FORCE_DEFAULT_BOOT_LINE
    strncpy (sysBootLine,DEFAULT_BOOT_LINE,strlen(DEFAULT_BOOT_LINE)+1);
#endif /* FORCE_DEFAULT_BOOT_LINE */
```

21.3 Bypassing the Boot Line Setup -- Linux

You can create, define, pre-set, and manage the Linux boot line (BL) parameters for your target board using low-level commands from the **OCD Command Shell**. This will allow you to define a new set of boot line parameters without re-compilation or rebooting from on-board ROM.

The boot line commands are **BL ADD**, **BL DELETE**, **BL DISPLAY**, **BL MODIFY**, and **BL UPLOAD**. For a detailed description of all these commands, see the *Wind River Workbench On-Chip Debugging Command Reference*.



NOTE: To use the **BL** commands you must set both the **CF BL** and **CF MMU** configuration options to **ENABLE**.

You can only use the **BL** commands in the **OCD Command Shell**. There is no BL-specific GUI for setting up boot parameters.

The following example displays a customized set of boot line parameters for a particular distribution of Linux that have been generated by the **BL** commands and stored in the emulator's NVRAM. On a **GO** command, these parameters will be loaded into the target memory and passed to Linux via a set of register pointers. This process is controlled by the emulator's run-time firmware DLL.

Dynamic Boot Table: structure configuration

Entry	Description	Value/String
00	MemStartAdd	0x00000000
01	MemSize	0x04000000
02	FlashStart	0x40000000
03	FlashSize	0x00400000
04	FlashOffset	0x00040000
05	SRAMStart	0x00000000
06	SRAMSize	0x00000000
07	IMMR_Base	0xf0000000
08	BOOTFlags	0x00000001
09	IP_ADDR	0x00000000
10	ENETADDR[6]	0x00a01ea87bcb
11	ETHSPEED	0x6c79
12	INTFREQ	0x0bcd3d80
13	vBUSFREQ	0x01f78a40
14	CPMFREQ	0x03ef1480
15	BRGFREQ	0x01f78a40
16	SCCFREQ	0x01f78a40
17	VCO	0x07de2900
18	BAUDRATE	0x00002580
19	bi_mon_fnc	0xffffffff
20	CmdStrg	->console=ttyS0,9600 root=/<- ->dev/ram0 rw

- The **Entry** field is a sequential reference for each line item.
- The **Description** field is an ASCII field only used for comment.
- The **Value/String** field can contain a char, unsigned long or unsigned short value.
- Unsigned long values are displayed in hexadecimal using 8 digits, as in Entry #1.
- Unsigned short values are displayed in hexadecimal using 4 digits, as in Entry #11.

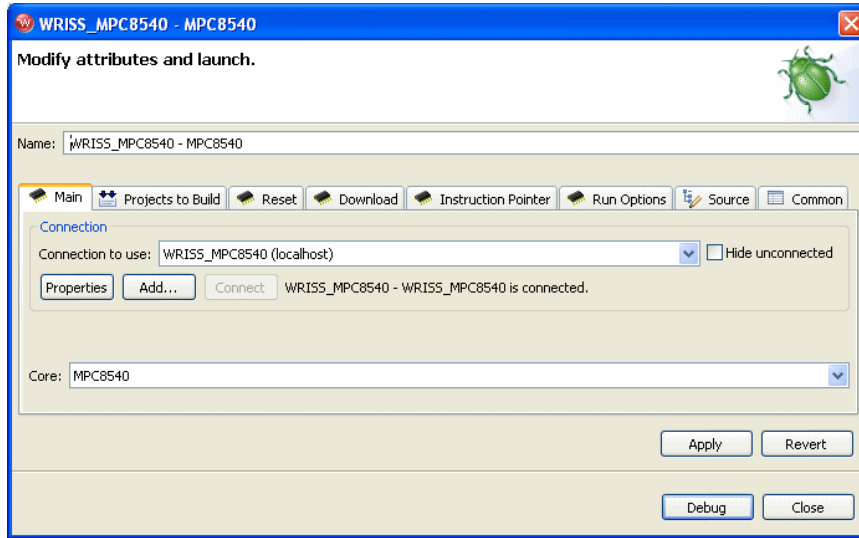
- Char values are displayed as in Entry #20.
- Char string greater than 20 characters will be displayed on several lines, using arrows.
- Byte values are displayed as in Entry #10.

21.4 Downloading the Kernel Image

Once you have bypassed the boot setup, initialize your target board and download your kernel image using the **Reset and Download** view.

1. Build your image using a VxWorks Image Project or a Linux Kernel Project.
For information on building projects, see the *Wind River Workbench User's Guide*.
2. Connect to your Wind River ICE SX or Wind River Probe.
For instructions on connecting to your emulator, see the *Wind River ICE SX for Wind River Workbench Hardware Reference* or the *Wind River Probe for Wind River Workbench Hardware Reference*.
3. In the **Target Manager** view, click the **OCD Reset and Download** icon to bring up the **Reset and Download** view, and click under the **Settings** heading to configure it.

Figure 21-3 **Reset and Download View**



4. In the **Reset** tab, choose your register file.
Check the **Play Register File** box and click **Browse**. Navigate to the register file for your target board.
For more information on register files, see [8. Configuring Target Registers](#).
5. Still in the **Reset** tab, choose the type of initialization you wish to perform, using the IN or INN command.
For more information on the IN and INN commands, see the *Wind River Workbench On-Chip Debugging Command Reference*.

Figure 21-4 Reset Tab

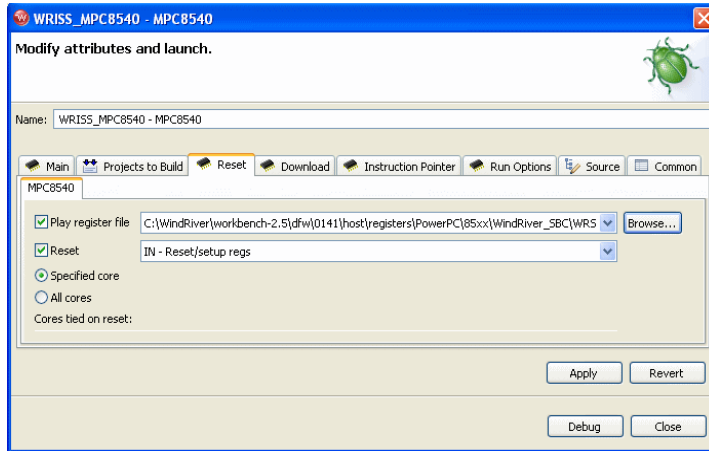
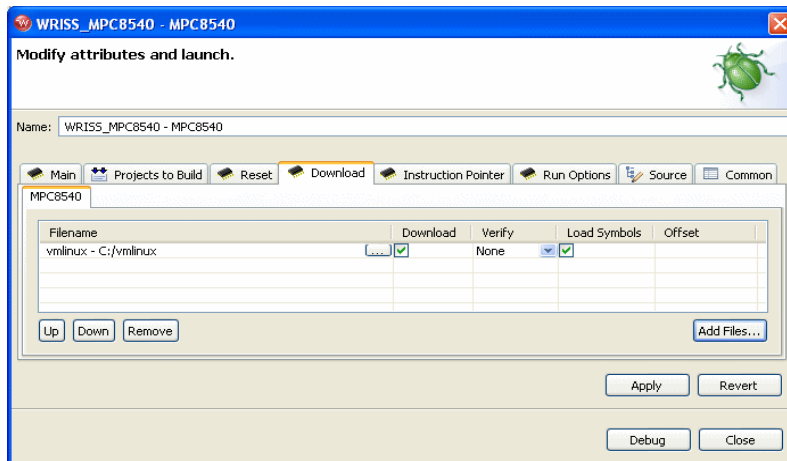


Figure 21-4 shows the **Reset** tab set to play the register file for a Wind River PowerQUICC II 8260 target and issue the **IN** initialization command.

6. In the **Download** tab, click **Add Files...** and navigate to the kernel image from your VxWorks Image Project or Linux Kernel Project.

Figure 21-5 shows the **Download** tab set to download the **vmlinux.elf** file from a Linux Kernel Project.

Figure 21-5 Download Tab

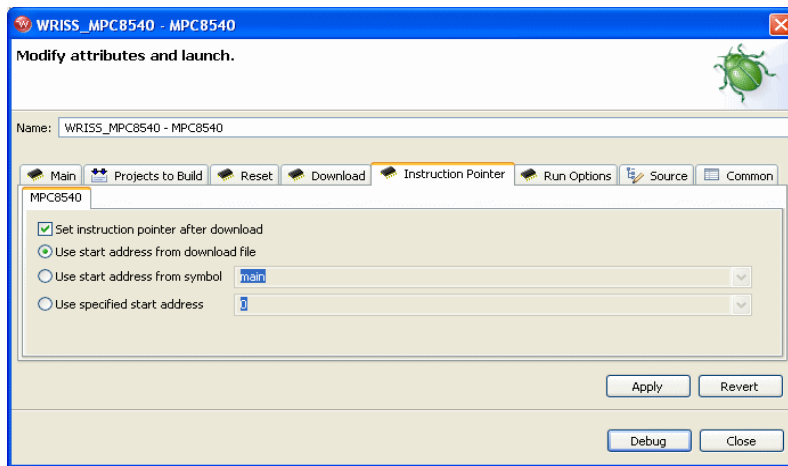


7. In the **Instruction Pointer** tab, set the start address for your file.

Figure 21-6 shows the **Instruction Pointer** tab set to use the start address from the download file. It also has the **Set Instruction Pointer After Download** box checked.

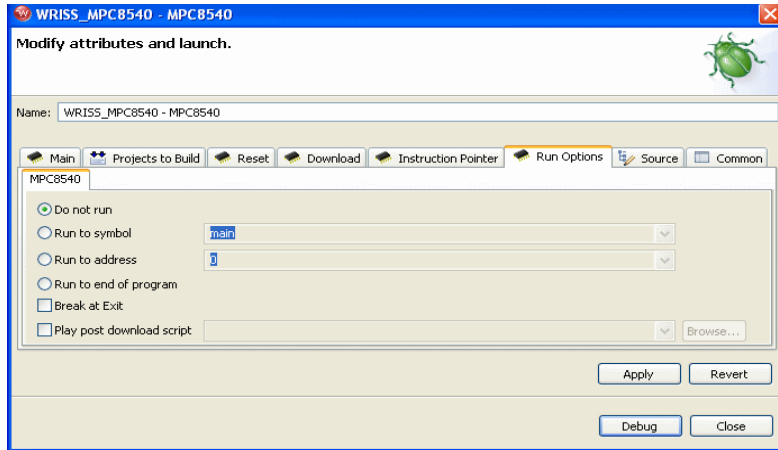
You can also select the **Use Start Address From Symbol** field and specify a symbol. For example, in VxWorks you would use the symbol **sysInit**. Or you can just specify a start address in hex.

Figure 21-6 **Instruction Pointer Tab**



8. In the **Run Options** tab, specify the memory location to where you want the download file to run. By default it is set not to run after download, as in Figure 21-7.

Figure 21-7 Run Options Tab

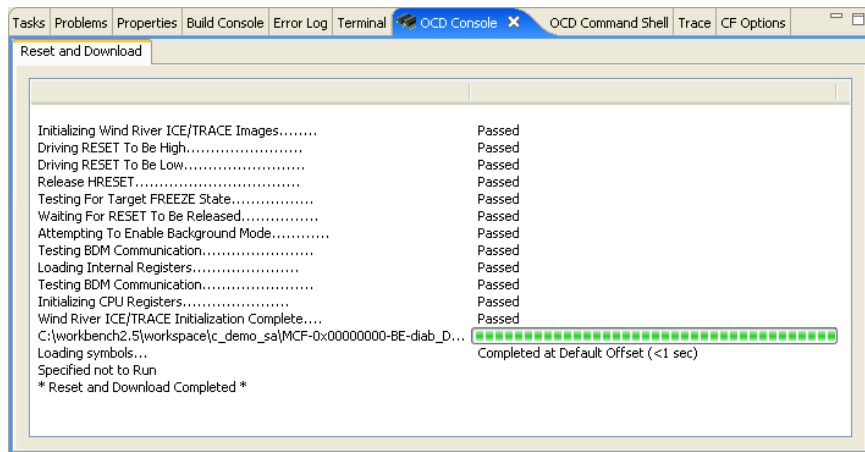


For more information on the **Reset and Download** view, see the *Establishing Communications* chapter of your emulator's Hardware Reference.

9. Once you have entered the values you want, click **Debug**.

The emulator initializes your target and downloads the kernel image. The **OCD Console** view opens to show the progress of the download, as shown in [Figure 21-8](#).

Figure 21-8 OCD Console View



The kernel image is now downloaded to your target. Run and debug it as you would any other file. For information on running and debugging files, see the *Wind River Workbench User's Guide*.

22

Kernel-Aware Debugging

22.1 Introduction

Wind River Workbench supports kernel-aware debugging for several operating systems other than VxWorks 6.x. Run control and data visibility for these operating systems are outlined in this chapter.

22.1.1 VxWorks 5.5

The following VxWorks 5.5 kernel objects are viewable in Workbench:

- Tasks

22.1.2 Linux

The following Linux kernel objects are viewable in Workbench:

- Processes
- Threads

The following Linux processor architectures are supported in Workbench:

- MIPS
- ARM9
- ARM11
- PowerPC

22.1.3 ThreadX

The following ThreadX kernel objects are viewable in Workbench:

- Threads
- Timers
- Mutexes
- Semaphores
- Memory Byte Pools
- Memory Block Pools
- Message Queues
- Event Flag Groups

The following ThreadX processor architectures are supported in Workbench:

- PowerPC 821
- PowerPC 823
- PowerPC 860
- PowerPC 8240
- PowerPC 8260
- PowerPC 60x
- PowerPC 44x
- PowerPC 405
- MIPS32 4kx

Additional processor support may be available from Express Logic at http://www.expresslogic.com/windriver_wb23.asp.

Index

A

- Adding a new group of registers to an existing .reg file 155
- Adding Files 170
- Adding registers to a register group 154
- Address Bus Test 253
- attach to user mode threads 199
- Attaching to a Process 199

B

- Basic Operation
 - Debugging with a Project 5
 - Debugging without a Project 33
- Bit-Level Detail 145
- Board Descriptor Files 88
- board files 89, 91
 - creating new 89
 - .XML 98
- boot line
 - address
 - bypassing with Linux 349
 - bypassing with VxWorks 347
 - forcing the default boot line 348
 - setting manually 347, 348
 - commands 349
 - configuration options 350

- Boot Line Commands 192
- boot parameters 347
- boot ROM 321
- Booting a Linux System with OCD 185
- bootloader 333, 345
 - U-Boot
 - downloading 334
 - generated files 337
 - installing 334
 - manually 335
- breakpoints
 - verifying with target 163, 324, 339
- Browsing Functions in Source 279
- Build Spec Wizard 67
- Build Tools 69
- Building a Downloadable U-Boot File 336
- Building an OCD Standalone Project 71
- Bus Tests 253
- Bypassing the Boot Line Address -- VxWorks 347
- Bypassing the Boot Line Setup -- Linux 349

C

- CF CMDRST 133
- CF HRESET 133
- changing CF options
 - with low-level commands 241
 - with the GUI 240

- Changing CF Options in the CF Options View 240
- Changing CF Options With Low-Level Commands 241
- Clear Trace Buffer 221
- Clear Trace Buffer on GO (TRCCLR) 227
- code profiling 255
- Collapsing and Expanding Fields 220
- commands
 - for multi-core debugging 134
- Commands for Multi-Core Debugging 134
- communications
 - with multiple devices 107
- Comparing Memory 303, 311
- Configuration Options 207
- configuration options
 - for multi-core debugging 131
- Configure Trace 225
- Configuring and Building the Bootloader 334
- Configuring and Building the Bootloader Manually 335
- configuring communication settings manually 110
- Configuring Communication Settings Manually 110
- Configuring Flash Memory Bounds 167
- Configuring Options for Multi-Core Debugging 131
- Configuring RAM Workspace 168
- Configuring Registers 164
- Configuring Registers Manually 143
- Configuring Target Registers 137
- Configuring target registers 137
- Configuring The Target 322
- Configuring the Target 338
- Configuring the Target Connection 238
- Configuring the Target Server 209
- Configuring Trace 225
- Configuring Wind River ICE SX 206
- Connecting Through the Transparent Mode Driver 204
- Connecting to a Target 158, 234
- Connecting to the Simulator 314
- Connecting to the Target 6, 34, 256, 282, 314
- Connecting to Your Target 244
- Connection Parameters 179

- Converting Files To .bin Format 171
- Converting Files To Wind River Flash Binary Format 171
- CRC Calculation 251
- creating
 - new board files 89
- Creating a Launch Configuration 76
- Creating a New Board Descriptor File 89
- Creating a New Set of Registers 147
- Creating a Project 18, 125, 266, 293
- Creating an OCD Standalone Project 64
- Creating New Registers With Low-Level Commands 149

D

- Data Bus Test 254
- Data Cache 301, 307
- debugger
 - disconnecting and terminating processes 32, 61
- Debugging a Process 199
- Debugging Code 25, 53
- Debugging Code in RAM 25, 53
- Debugging Linux Kernel Modules 196
- Debugging Multi-Core Targets 105
- Debugging the Beginning of a Process 200
- Debugging the init() Function of a Module 197
- Debugging the Linux Kernel 196
- Debugging User Space Applications with OCD 198
- Defining a Launch Configuration 75
- Disconnecting and Terminating Processes 32, 61
- Document Overview 1
- document overview 1
- downloading
 - register files 138
- Downloading a .REG File 138
- Downloading a Kernel Image Using a JTAG Connection 345
- Downloading a Register File 138
- downloading an OS image using a JTAG connection 345
- Downloading Code 46
- Downloading the Kernel Image 351

Downloading the Sample Code 23, 130, 271, 298

E

Editing Your Board Layout 97
 Emulator Configuration 183
 Enabling A File For Download 174
 Enabling and Disabling Register Groups 141
 Enabling and Disabling Register Groups with Low-Level Commands 142
 Erasing and Programming Flash 169
 Establishing Communications with Multiple Devices 107
 Examining Cache 299
 Examining the Cache 299
 Exporting Cache Information 305

F

file conversion utility 329
 files
 .REG 138
 Filter Visible Trace Events 224
 Flash Configuration Tab 166
 flash memory 321, 333
 programming
 requirements 321, 333
 Flash Memory/Diagnostics Tab 174
 Flash Programmer view 168
 Configuration tab 166
 getting started 165
 Memory/Diagnostics tab 174
 Flash programming
 erasing flash 169
 setting timeouts 168
 verifying flash contents 169
 flash programming 321, 324, 326, 333
 configuring the target 322
 reset configuration word 325
 setting chip selects 325
 testing breakpoints 324, 339
 unlocking flash 326

Flash Programming Tab 168
 Flash Programming view 326, 342
 Configuration view 327, 342
 file conversion utility 329
 Files view 327, 343
 Memory/Diagnostics view 331
 flashing a boot ROM 321, 324
 flashing a Linux bootloader 333
 Flashing the Boot ROM 324
 Flashing the Bootloader 340
 Forcing the DEFAULT_BOOT_LINE 348
 Full RAM Tests 251

G

Getting Started 165
 GO ALL command 134

H

HALT ALL command 134
 Hardware Diagnostic Tests 249
 Hardware Diagnostic view
 tests 249

I

Initializing the Targets 119
 Installing the Bootloader 334
 Instruction Cache 299, 306
 Instruction Set Simulator 313
 Internal Software Trace 219
 Interpreting the Cache View 301
 Introduction 1, 5, 33, 63, 75, 87, 105, 137, 157, 177, 203, 233, 243, 255, 281, 309, 313, 321, 333, 345, 357

J

JTAG 345

JTAG Editor

defining a core 94

defining a graphic layout in 91

editing a board layout 97

JTAG Editor view 89

selecting a processor type 93

toolbar 91

using the custom option 95

JTAG editor 89

JTAG Server 106

JTAG server 106

K

Kernel Configuration 198

Kernel Module Detection 196

Kernel-Aware Debugging 357

L

Launch Configuration

Add Source dialog 83

common tab 84

Download tab 80

Adding files 80

Configuring download 80

Download field 80

Load Symbol field 81

Offset field 81

Verify field 81

Setting the instruction pointer 81

source tab 83

Launch configuration ??–125

common tab 51, 124, 265, 292

source tab 50, 123, 264, 291

Limitations 200

Linux 357

linux

troubleshooting 201

user space applications 198

Linux Troubleshooting 201

Linux Virtual Memory Management

Architecture 178

M

Making Physical Connections 338

Making Physical Connections 322

Manually Configuring Flash Memory Erasure

Bounds 170

Manually Creating XML Board Files 101

Manually Setting the BOOT_LINE_ADRS

Location 348

MMUL Settings 183

Modifying an Existing Register File 154

Modifying the boardConfig.h File 335

Monitoring Processes 25, 54

Moving On 32, 61, 218

Multi-Core Debugging 107

multi-core debugging 107

commands for 134

configuration options 131

N

NV-RAM 137

O

OCD 1

OCD Boot 189

OCD Statistical Code Profiling 255

on-chip debugging 1

On-Chip Debugging for Linux 177

Open Trace Rules Dialog 222

Other Options 85

Other Resources 2

Overview 219

overview

document 1

P

- PA Semi Trace Configuration 228
- PFA profiling 255
- play a register file 47, 120, 138, 261, 288, 325
- Playing a Register File 325, 340
- PowerPC Trace Configuration Options 225
- processes
 - disconnecting debugger 32, 61
- Profiling Selected Functions 278
- Profiling Your Code 272
- Programming a Linux Bootloader into Flash Memory 333
- Programming a VxWorks Boot ROM into Flash Memory 321
- Programming Flash 326, 342
- programming flash 321, 333
- Programming Flash Memory 157

R

- Read From Location 253
- Reading and Writing Memory 163, 323, 339
- Reconfiguring the Cache 305
- Refresh Trace View 222
- .REG files 138
- register groups
 - disabling 141
 - enabling 141
- Registers view 144
- Removing Files 171
- Removing Functions 279
- requirements
 - for flash programming 321, 333
- reset configuration word 325
 - programming with Wind River Workbench u-boot.bin file 341
- Resetting CF Options 242
- Reverse-Engineering the Boot Line Parameters 195
- RST command 134
- RSTINN command 134
- Running a Pre- or Post-Flash Script 170
- Running a Program 28, 56
- Running Diagnostic Tests 175

S

- Save Output to File 224, 228
- Saving Register Settings from a Target 139
- SC Commands 155
- SCGA Options 151
- Scope Tests 253
- Selecting a Flash Driver 166
- Selecting Flash Sectors for Erasure 170
- Setting a Hardware Breakpoint 29, 58
- Setting a Software Breakpoint 27, 55
- Setting a Tracepoint 231
- Setting a Workspace 248
- Setting Breakpoints 200
- Setting CF Options in the CF Options View 207
- Setting CF Options with Low-Level Commands 207, 208
 - setting chip selects 325
- Setting Standalone Project Defaults 72
- Setting The Download Offset Of A File 173
- Setting Timeouts 168
- Setting Up Chip Select 0 and Programming the Reset Configuration Word 325, 341
- Simple RAM Test 249
- Source Mode 303
- Specifying Files 78
- Standalone Project Wizard 63
 - Default Settings 72
- Standard Boot 185
- Stepping Through a Program 29, 57
- Stepping Through Code 25, 54
- System Configuration (SC) Commands 155

T

- target
 - jumper settings 322, 338
 - physical connections to 322, 338
 - software breakpoints, verifying 163, 324, 339
 - switch settings 322, 338
- Target Console Redirection 208
- Testing Breakpoints 324, 339
- testing breakpoints 324, 339
- Testing Flash Workspace 163

Testing Memory and Breakpoints [323, 339](#)
The Trace View [220](#)
Thread-Qualified Breakpoints [200](#)
ThreadX [358](#)
TMD Mode [207](#)
Toggle Trace/Source view Auto-Sync [221](#)
Trace View Buttons [220](#)
Tracing Execution [231](#)
Transparent Mode Driver [203](#)
 configuring the target server [209](#)
 connecting [204](#)
 using with an emulator [206](#)
Trap Exception [208](#)

U

U-Boot
 building
 manually [335](#)
 configuring
 manually [335](#)
 downloading [334](#)
 generated files [337](#)
 installing [334](#)
 u-boot file [337](#)
Unlocking Flash [326, 341](#)
unlocking flash memory [326, 341](#)
Updating the Profile Data [279](#)
Uploading Memory [309](#)
Uploading Target Memory to a Binary File [309](#)
user mode [199](#)
 breakpoints [200](#)
 thread-qualified [200](#)
 processes [200](#)
user space applications [198](#)
Using Board Descriptor Files [87](#)
Using Hardware Diagnostics [243](#)
Using Processors Without Cache Lines [306](#)
Using the Cache View [281](#)
Using the CF Options View [233](#)
Using the Custom Option in the JTAG Editor
 View [95](#)
Using the Flash Programmer View [165](#)
Using the Instruction Set Simulator [313](#)

Using the OCD Standalone Project Wizard [63](#)
Using the Predefined Layouts in JTAG Editor [91](#)
Using the TMD With the Wind River ICE SX [206](#)
Using the WDB Transparent Mode Driver [203](#)
Using Your New Register File [153](#)

V

Verifying Flash Contents [169](#)
Viewing Cache Source [303](#)
Viewing Memory [175](#)
VxWorks 5.5 [357](#)

W

WDB [203](#)
Wind River DeBug [203](#)
Wind River ICE SX
 Configuring for Transparent Mode [206](#)
Working With Custom Register Groups [147](#)
Write and Complement [253](#)
Write Rotating Value [253](#)
Write Then Read [253](#)
Write To Location [253](#)

X

XML Board File Fields [100](#)
.XML board files [98](#)
XML Board Files [98](#)