Using Macraigor JTAG/BDM Devices with Eclipse 4.2 (Juno) and the Macraigor GNU Tools Suite on Windows and Linux32/64 Hosts

Contents

1. Introduction

The Macraigor GNU Tools Suite (MGTS) is an implementation and packaging of several of the open-source GNU tools and utilities along with a two programs called ocdremote and ocdremoteServer that provide an interface between the GDB debugger and a Macraigor On-Chip debug device. The GNU packages that are included with the MGTS are: binutils (version 2.21), gcc (version 4.6.0), and gdb (version 7.2). Macraigor provides two versions of the gdb debugger: *-elf-gdbtui which has a simple text windowed user interface, and *-elf-gdb with the standard gdb command line interface. Our pre-built Eclipse project examples show how to integrate *-elf-gdb into the free, open-source Eclipse 4.2 (Juno) IDE. Taken together, these tools and the Eclipse framework provide a complete environment for cross development targeted at several families of embedded processors. Macraigor distributes versions of the MGTS for the following processors:

- ARM (ARM7/9/11/Cortex)
- ColdFire
- MIPS (32 and 64 bit)
- PowerPC
- XScale
- X86

This document gives instructions for installing and configuring the MGTS and Eclipse Indig so that a Macraigor JTAG/BDM device can be used as the debug connection to the target processor. Under Windows, the following Macraigor devices are supported with this environment:

- Wiggler (windows only)
- Raven (windows only)
- usb2Demon
- usb2Sprite
- usbWiggler
- WifiDemon
- mpDemon

In addition to the tools and development environment, Macraigor also supplies pre-built Eclipse projects for many industry-standard evaluation boards. These projects provide a simple way to get the cross-development environment up and running on actual hardware. The user can then modify and expand these demos to work with other target hardware and more complex application development.

2. Required Components

There are several components that must be present or downloaded and installed on your host PC in order to get the Macraigor GNU Tools Suite working with Eclipse 4.2 (Juno). All of the following components (except the Java Runtime) are either available from the Macraigor web site (www.macraigor.com/eclipse) or from links found there. The versions of these components that Macraigor posts on their web site have been tested and are known to work together. If you want to try using a newer or alternate version of any of these components, Macraigor will be unable to assist you if something doesn't work properly.

WINDOWS/VISTA 32/Windows 7 32/Windows 7 64 HOST :

The Windows/Vista 32/Windows 7 32 host required components are as follows:

0. Cygwin environment
1. GNU C/C++ compiler, GDB and utilities for your target processor
2. Macraigor ocdremote/ocdremoteServer
3. SUN Java Runtime
4. Eclipse 4.2 (Juno) IDE for C/C++ Developers (bundled with CDT 8.1) + Plug-ins used by Macraigor JTAG Devices

Installation of each component is explained in detail in the following sections.

2-0 (Windows) Cygwin Environment

Cygwin is a free package that provides a Linux-like environment under Windows. This environment is required in order to run the gnu compiler, debugger and the Macraigor OCDRemote interface. Go to www.cygwin.com and select the "Install or update now! (using setup.exe)" link, download/run setup.exe and make the following choices in the "Cygwin Setup" dialogs:

1. Choose Installation Type : Select "Install From Internet"
2. Choose Installation Directory : Specify: Root Directory : "c:\cygwin", Install For : All Users, Default Text File Type : unix/binary
3. Select Local Package Directory: Specify : "c:\"
4. Select Connection Type : Specify your Internet connection type
5. Choose download sites : Select a site to download from

6. Select Packages :
   a) Expand the "Devel" category by clicking on the "+" sign then scroll to:

- "make : The gnu version of the make utility"

click the arrows icon ↻to replace "skip" with a version number

b) Expand the "Libs" category by clicking on the "+" sign, scroll to:

- "expat: XML parser library"
- "libexpat0: XML parser library written in C",
- "libgmp3 : Run time Library for GMP arbitrary precision arithmetic"
- "libmpfr4: A library for multiple precision floating point arithmetic with exact rounding"

Click the arrows icon ↻to replace "skip" with their version numbers.

c) At the "Python" category click the arrow icon ↻to replace "Default" with "Install"

Clicking the NEXT button in the "Select Packages" dialog box will start the Cygwin Installation.

7. Once Cygwin is installed, open a Cywin bash shell window (by clicking the Cywin icon on your desktop) and enter the following command:
   ```
   chmod –R a+w /usr/local
   ```
   This command makes the /usr/local directory (and it's subdirectories) write-able so we can install our software packages there.

2-1 (Windows) GNU C/C++ compiler, GDB and Utilities

Macraigor provides pre-built, installable builds of the gnu tools suite (gcc, binutils, and gdb) for cross-development with several families of target processor, including ARM (which also works for Intel XScale), i386 (for Intel Atom), MIPS, M68K (which works for Coldfire processors) and PowerPC. The tools for each architecture are packaged separately. Choose the appropriate package from the following links:

ARM:	http://www.macraigor.com/gnu/gnutools-arm-elf-4.7.0.exe
M68K:	http://www.macraigor.com/gnu/gnutools-m68k-elf-4.7.0.exe
MIPS:	http://www.macraigor.com/gnu/gnutools-mips-elf-4.7.0.exe
PowerPC:	http://www.macraigor.com/gnu/gnutools-powerpc-elf-4.7.0.exe
X86:	http://www.macraigor.com/gnu/gnutools-i386-elf-4.7.0.exe

and download and install the tools for your target processor. Note that all of these packages can coexist, so feel free to install all the versions of the tools that you might need for various projects.

In Windows 7, after you have downloaded the Install Shield executable. Use the Windows Explorer's: "Run as administrator" option (selected from the menu displayed after you have highlighted the gnu toolkit Install Shield executable and clicked the right mouse button) to insure you have all the "privs" needed for the install.

2-2 (Windows) Macraigor ocdremote and ocdremoteServer

In order to connect the gdb debugger to a target processor using a Macraigor JTAG or BDM interface device, a utility called ocdremote is required. This utility start ups and listens on a TCP/IP port for a connection from gdb and then translates gdb commands into JTAG/BDM actions for the target processor using the Macraigor hardware interface device. Two versions of this utility are provided:

ocdremote – which has to be started before each gdb session starts and terminates once the gdb session ends

ocdremoteServer – which only has to be started once. It runs continuously after it has been started and waits on the TCP/IP ports for the next gdb connection after the current gdb session ends.

Go to http://www.macraigor.com/gnu/hw_support_10.0.0.exe

Download and install the Hardware Support Package. This installation package also includes several Macraigor utilities that can be helpful in debugging and correcting connection issues when using Eclipse and the gnu tools. These utilities are:

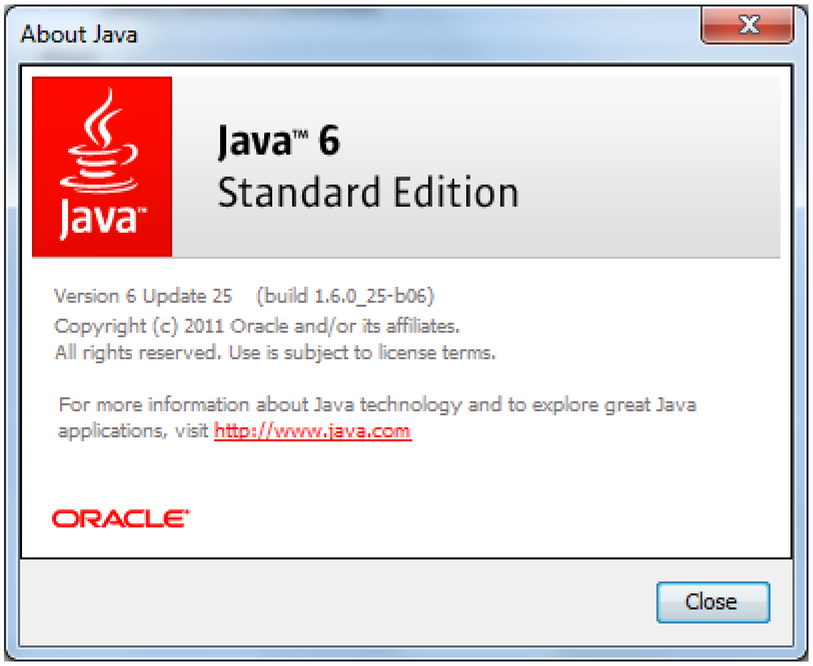
- OCDCommander – a low-level, assembly-only debugger that can be used to check proper connection of a Macraigor JTAG/BDM interface to the target processor.
- usbDemon Finder – a utility to detect and license Macraigor USB interface devices that are connected to the host PC.
- JTAG Scan Chain Analyzer – a utility that attempts to analyze a JTAG scan chain and determine the number of devices present and the identity of any processors that are found.
- AsmDasm – a single op-code RISC assembler/disassembler that supports the AM32, ARM, MIPS, and PowerPC processor families
- DccTerminal – a terminal emulator that when configured with the TCP/IP address/port of an ocdremote Debug Commutation Port lets the user send/receive ascii data to to/from and application running on the debug target

These utilities will be available to launch directly from within Eclipse once the environment has been correctly configured.

Installing this package will also create the demo Eclipse projects that will be used later. These files will be placed, by default, into C:\cyginw\usr\local\macraigor\EclipseDemo\Juno.

2-3 (Windows) SUN Java Runtime

The Eclipse IDE is written entirely in JAVA and requires that the Java Runtime Environment (JRE) be installed on the host computer. Many Windows hosts already have the JRE installed. You can check for an installed JRE by looking in the Windows Control Panel under Programs -> Java -> About. If the JRE is installed, there will be an entry here that looks like the following:



Version 6 Update 25 (build 1.6.0_25-b06)
Copyright (c) 2011 Oracle and/or its affiliates.
All rights reserved. Use is subject to license terms.

For more information about Java technology and to explore great Java applications, visit http://www.java.com

If you do not have the JRE installed, it can be obtained for free from www.java.com. Follow the directions to download and install the software and then check again in the Control Panel under : Programs -> Java -> About to be sure that the J2SE Runtime Environment is available.

LINUX 32/64 (Fedora Core 4–15/Ubuntu 10.04-12.04) HOST:

The Linux host required components are as follows:

1 GNU C/C++ compiler, GDB and utilities for your target processor
2. Macraigor ocdremote/ocdremoteServer
3. SUN Java Runtime
4. Eclipse 4.2 (Juno) IDE for C/C++ Developers (bundled with CDT 8.1) plug-ins used by Macraigor JTAG Devices

Installation of each component is explained in detail in the following sections.

2-1 (Linux) GNU C/C++ compiler, GDB and Utilities

Macraigor provides pre-built, installable builds of the gnu tools suite (gcc, binutils, and gdb) for cross-development with several families of target processor, including ARM (which also works for Intel XScale), i386 (for AMD embedded processors), MIPS, M68K (which works for Coldfire) and PowerPC. The tools for each architecture are packaged separately. Choose the appropriate package from the following links:

ARM:	http://www.macraigor.com/gnu/mcgr-arm-gnutools-4.7-0.i386.rpm http://www.macraigor.com/gnu/mcgr-m68k-gnutools_4.7.0-1_i386.deb
x86:	http://www.macraigor.com/gnu/mcgr-x86-gnutools-4.7-0.i386.rpm http://www.macraigor.com/gnu/mcgr-x86-gnutools_4.7.0-1_i386.deb
MIPS:	http://www.macraigor.com/gnu/mcgr-mips-gnutools-4.7-0.i386.rpm http://www.macraigor.com/gnu/mcgr-mips-gnutools_4.7.0-1_i386.deb
M68K:	http://www.macraigor.com/gnu/mcgr-m68k-gnutools-4.7-0.i386.rpm http://www.macraigor.com/gnu/mcgr-m68k-gnutools_4.7.0-1_i386.deb
PowerPC:	http://www.macraigor.com/gnu/mcgr-powerpc-gnutools-4.7-0.i386.rpm http://www.macraigor.com/gnu/mcgr-powerpc-gnutools_4.7.0-1_i386.deb

to download the tools for your target processor. Note that all of these packages can coexist, so feel free to install all the versions of the tools that you might need for various projects.

Users on Fedora systems typically download rpm files into the /usr/src/redhat/RPMS/i386 directory. They then cd to this directory and use the following command to install the tools contained in the rpm file:

```
rpm –i mcgr-<cpu type>-gnutools-4.6-0.i386.rpm
```

NOTES:
Fedora systems require the “compat-expat” package be installed prior to installing our gnu tools packages. Start up the Add/Remove Software system utility, search for compat-expat and install it if it is not already installed.

The command “rpm –e mcgr-<cpu type>-gnutools-4.6.0” uninstalls these packages.

These rpm files install the gnutools into /usr/local/bin.

Users on Ubuntu Systems typically download the .deb files into their home directories. They then cd to this directory and use the following command to install the tools contained in the .deb file:

```
sudo deb --install --force mcgr-<cpu type>-gnutools-4.6-1_i386.deb
```

These .deb files install the gnutools in /usr/local/bin

2-2 (Linux) Macraigor ocdremote/ocdremoteServer

In order to connect the gdb debugger to a target processor using a Macraigor JTAG or BDM interface device, a utility called ocdremote is required. This utility starts up and listens on a TCP/IP port for a connection from gdb and then translates gdb commands into JTAG/BDM actions for the target processor using the Macraigor hardware interface device. Two versions of this utility are provided:
ocdremote – which has to be started before each gdb session starts and terminates once the gdb session ends
ocdremoteServer – which only has to be started once. It runs continuously after it has been started and waits on the TCP/IP ports for the next gdb connection after the current gdb session ends.

These utilities can also be configured to set up ARM7/9/Xscale Data Communication Channel ports that let an external program such as DccTerminal send and receive data packets to/from target via the JTAG DCC channel

Download:

http://www.macraigor.com/gnu/mcgr-hwsupport-10.0-0.i386.rpm	(Fedora 32)
http://www.macraigor.com/gnu/mcgr-hwsupport-10.0-0_i386.deb	(Ubuntu 32)
http://www.macraigor.com/gnu/mcgr-hwsupport-10.0-0.x86_64.rpm	(Fedora 64)
http://www.macraigor.com/gnu/mcgr-hwsupport_10.0-0_amd64.deb	(Ubuntu 64)

into $HOME/Download. "cd" to this directory and install the mcgr-hwsupport<version>.rpm package with the following command:

```
rpm –i mcgr-hwsupport-<version>.rpm.
```

NOTE: Fedora systems require the "kernel-devel" or "kernel-PAE-devel" and "development-tools" packages be installed prior to installing our hwsupport package.
In a terminal window enter "uname –r". Note if your kernel version ends in ".PAE"
Start up the Add/Remove Software system utility:
1) Search for either "kernel-devel" (kernel version does not end in ".PAE") or "kernel-PAE-devel" (kernel version ends in PAE")
Install the kernel-devel/kernel-PAE-devel version that matches your kernel's version.
2) Search for development-tools and install them if they are not already installed.

Install the mcgr-hwsupport-<version>.deb with the following command:

```
sudo dpkg –install mcgr-hwsupport-<version>.deb
```

NOTE : The command "rpm –e mcgr-hwsupport-<version>" uninstalls all of the mcgr-hwsupport software and drivers in Red Hat/Fedora Linux systems
The command "sudo dpkg --remove mcgr-hwsupport" uninstalls all of the mcgr-hwsupport software and drivers on Ubuntu systems.

This installation package also includes several Macraigor utilities that can be helpful in debugging and correcting connection issues when using Eclipse and the gnu tools. These utilities are:

- OCDCommander – a low-level, assembly-only debugger that can be used to check proper connection of a Macraigor JTAG/BDM interface to the target processor

- OcdCommander.jar – a java (and therefore more Linux friendly) version of the OCD Commander.
- usbDemon Finder – a utility to detect and license Macraigor USB interface devices that are connected to the host PC.
- JTAG Scan Chain Analyzer – a utility that attempts to analyze a JTAG scan chain and determine the number of devices present and the identity of any processors that are found.
- AsmDasm – a single op-code RISC assembler/disassembler that supports the AM32, ARM, MIPS, and PowerPC processor families
- DccTerminal – a terminal emulator that when configured with the TCP/IP address/port of an ocdremote Debug Communation Port lets the user send/receive ASCII data to/from an application running on the debug target
- ErasePIC32 – utility used to erase the PIC32 CPU's on-board FLASH memory.

These utilities will be available to launch directly from within Eclipse once the environment has been correctly configured.

Installing this package will also create the demo Eclipse projects that will be used later. These files will be placed, by default, into \usr\local\macraigor\EclipseDemos\Juno.

2-3 (Linux) Java Runtime

The Eclipse IDE is written entirely in JAVA and requires that the Java Runtime Environment (JRE) be installed on the host computer. Many Windows hosts already have the JRE installed. You can check for an installed JRE by opening the Add/Remove Software and searching for "OpenJDK" If the JRE is installed, there will be an entry here that looks like the following:

Open jDK Runtime Environment
JAVA-<version>–openjdk- …..

If you do not have the JRE installed, it can be obtained for free from www.java.com. Follow the directions to download and install the software and then check again under Programs -> Java -> About to be sure that the J2SE Runtime Environment is available.

2-4 Eclipse 4.2 (Juno) IDE + Plug-ins used by Macraigor JTAG devices

The Eclipse IDE is an Integrated Development Environment that provides a graphical framework for running an editor, compiler, and debugger. Eclipse was originally developed by IBM, but has since been donated to the open-source community and is now maintained as a large Open-Source development project.

Macraigor Systems provides Eclipse 4.2 (Juno) preconfigured with the following Plug-ins:

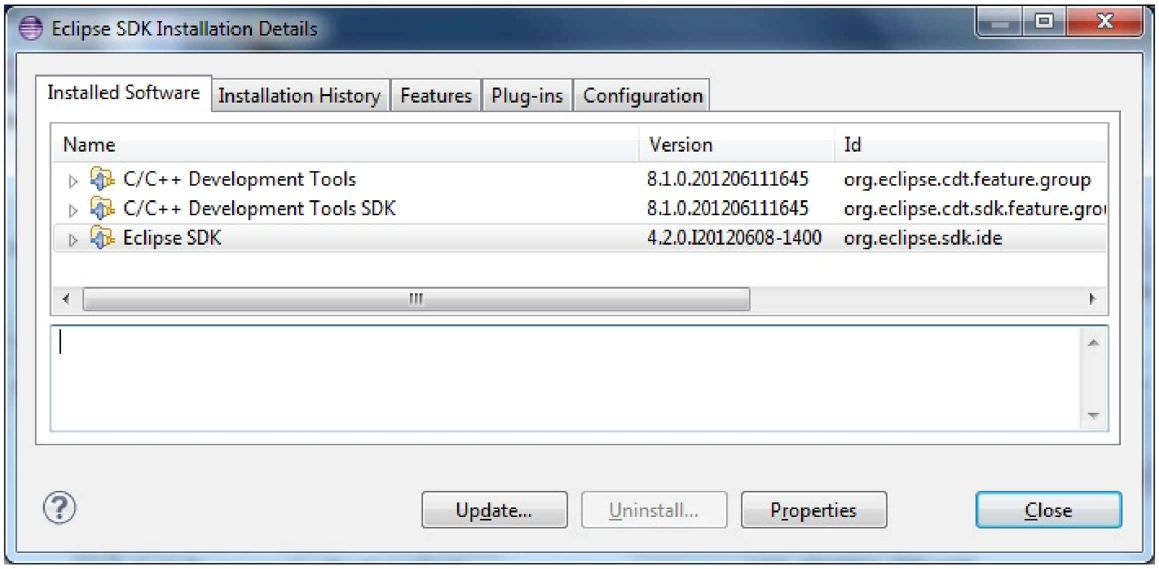
- Eclipse IDE for C/C++ Developers
- Eclipse IDE for Java Developers (which includes Junit Plug-In test tools)

This version of Eclipse can be used to write, compile, download, and debug embedded code on a remote target processor that is connected to the host machine via a Macraigor JTAG or BDM interface device. It can also be used to run Java Junit board tests on target hardware.

You can download our preconfigured version of Eclipse 4.2 Juno from:

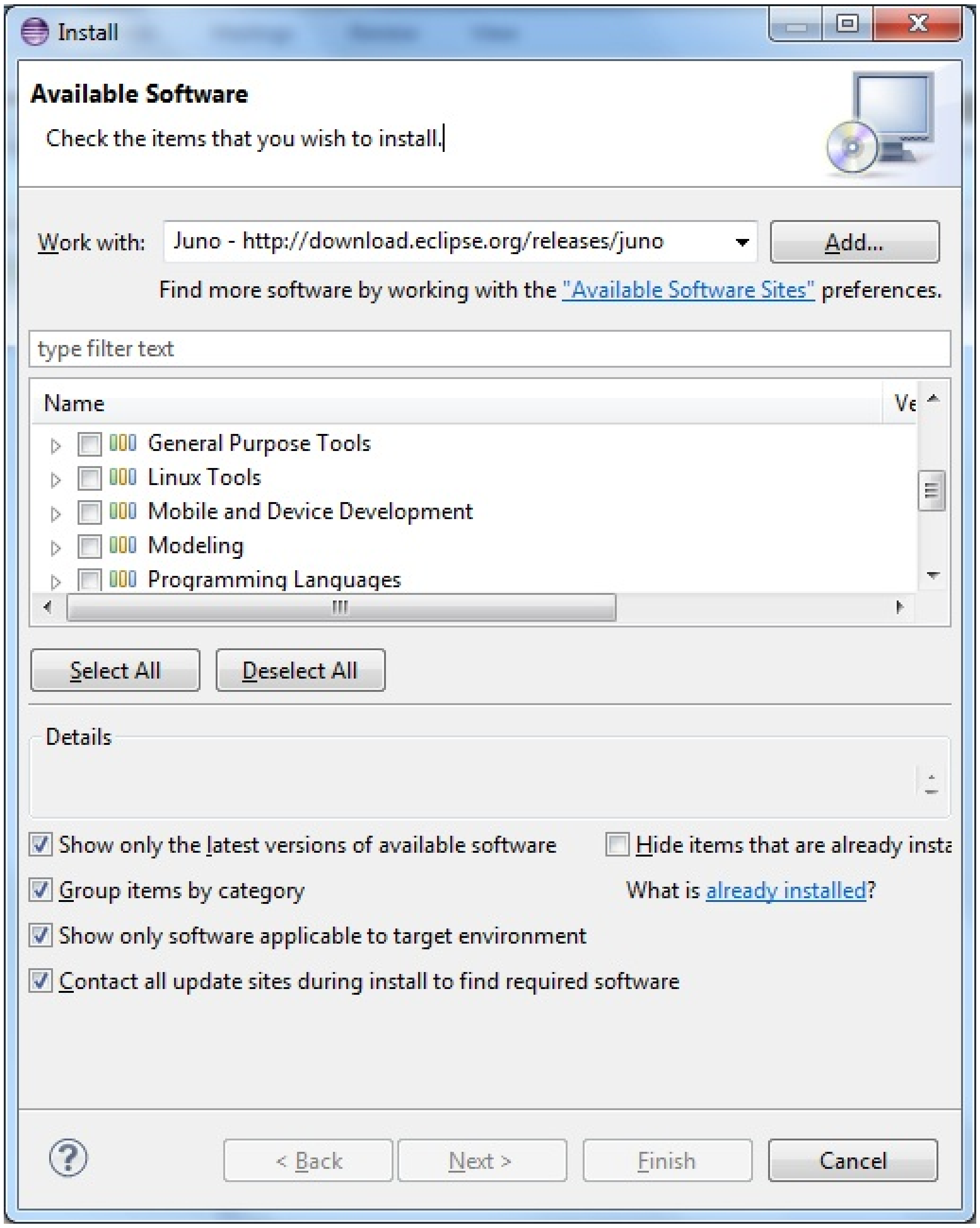
Eclipse 4.2.1 (Juno) + Plug-ins for Windows XP/2000/Vista
Eclipse 4.2.1 (Juno) + Plug-ins for Linux32
Eclipse 4.2.1 (Juno) + Plug-ins for Linux64

Alternatively you can download the latest version from the Eclipse web site:
http://www.eclipse.org/downloads -> Eclipse IDE for C/C++ Developers
(There are separate Windows, Linux 32 and Linux 64 downloads),
Start Eclipse, select Help->Install New Software, then click on the "What's already installed" hypertext at the bottom of the screen. Our version of Eclipse Juno has the following software packages installed:



Name	Version	Id
C/C++ Development Tools	8.1.0.201206111645	org.eclipse.cdt.feature.group
C/C++ Development Tools SDK	8.1.0.201206111645	org.eclipse.cdt.sdk.feature.gro
Eclipse SDK	4.2.0.I20120608-1400	org.eclipse.sdk.ide

You can install any packages your Eclipse is currently missing from the Install Dialog:


Install

Available Software

Check the items that you wish to install.

Work with: Juno - http://download.eclipse.org/releases/juno [Add...]

Find more software by working with the "Available Software Sites" preferences.

type filter text

Name	Ve
☐ General Purpose Tools	
☐ Linux Tools	
☐ Mobile and Device Development	
☐ Modeling	
☐ Programming Languages	

[Select All] [Deselect All]

Details

☑ Show only the latest versions of available software ☐ Hide items that are already insta

☑ Group items by category What is already installed?

☑ Show only software applicable to target environment

☑ Contact all update sites during install to find required software

[< Back] [Next >] [Finish] [Cancel]

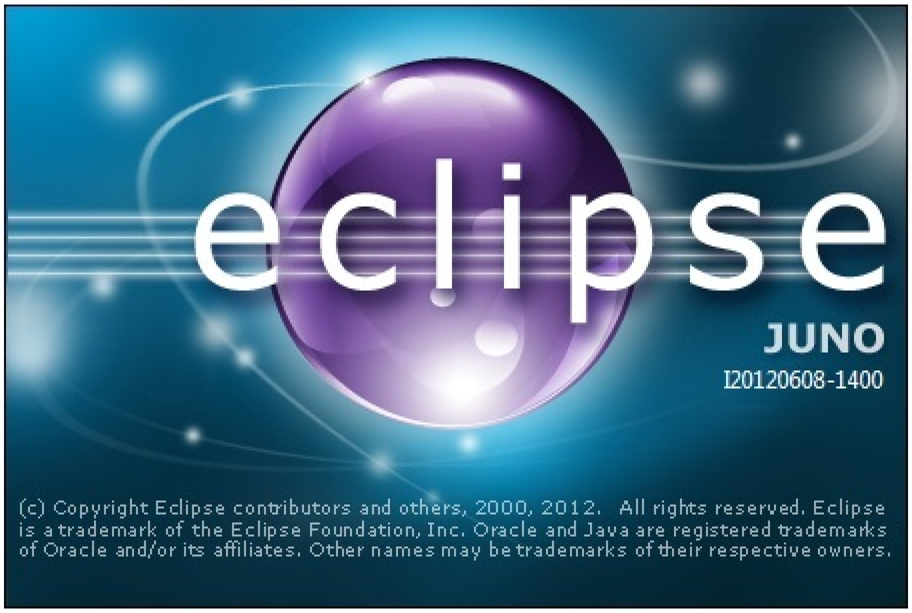
Eclipse is distributed as a single, large zip file.

WINDOWS:
Once you have it downloaded, extract the contents to the root of your hard drive (usually "C:") using Winzip or another unzipping utility..
LINUX 32/64:
Once you have downloaded extract the contents to /usr/local using the File Roller (by clicking on the Eclipse .tar.gz file in the File Browser).

This is the only installation that is required. A directory named "eclipse" should have been created on your drive.

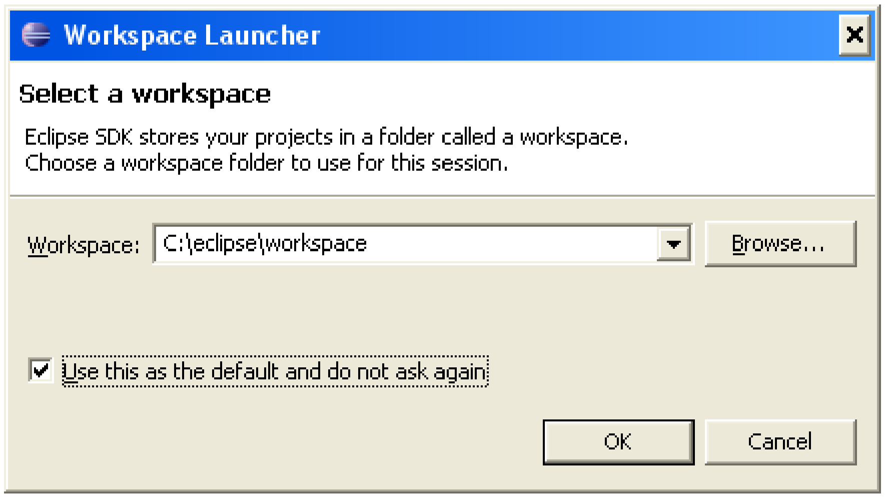
At this point, you should have a working Eclipse installation. This can be tested by attempting to run Eclipse. The Eclipse installation doesn't install a Desktop shortcut. You can create a short cut to the executable file …\eclipse\eclipse.exe (windows) or /usr/local/eclipse/eclipse (linux) for later ease of use.

To run Eclipse, either double-click your newly-created shortcut from the Desktop or double-click on "eclipse.exe" in Windows Explorer or "eclipse" in your Linux File browser. If everything is working properly, you should see the Eclipse 4.2 Juno splash screen, something like this:


eclipse
JUNO
I20120608-1400
(c) Copyright Eclipse contributors and others, 2000, 2012. All rights reserved. Eclipse is a trademark of the Eclipse Foundation, Inc. Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

If you don't see the splash screen, double-check that you have the JAVA Runtime Environment correctly installed (see section 2-4) and that the Eclipse zip file was properly unzipped.

After a short delay, you should see the "Workspace Launcher" dialog pop up. It looks like this:


Workspace Launcher

Select a workspace

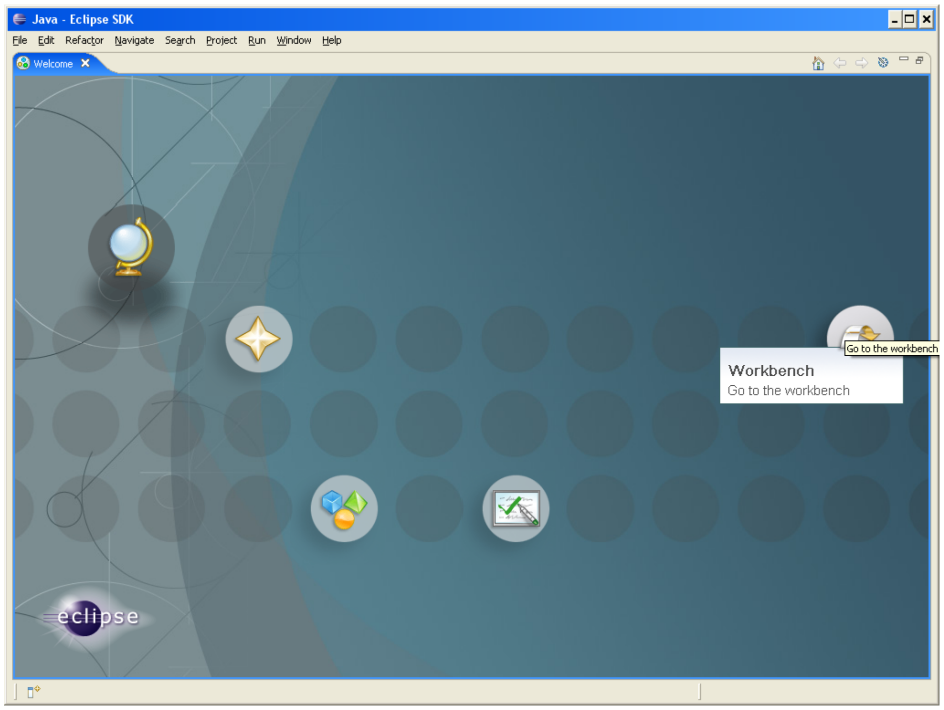
Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\eclipse\workspace

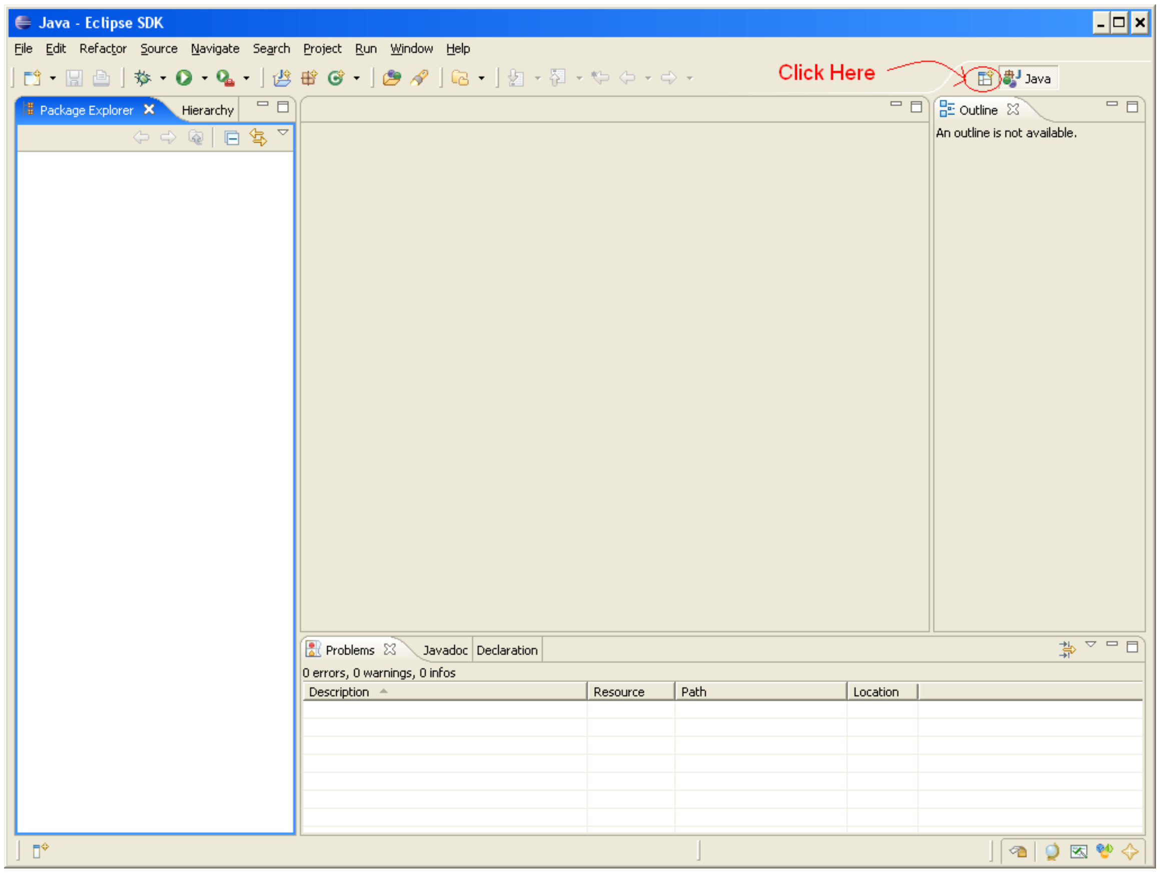
☑ Use this as the default and do not ask again

The Workspace location defaults to the current user's Documents and Settings directory. We prefer to keep all of the Eclipse files located together under the \eclipse directory and suggest that you create the workspace directory there as shown in the picture. This is not, however, critical and you can locate the workspace anywhere that you'd like.

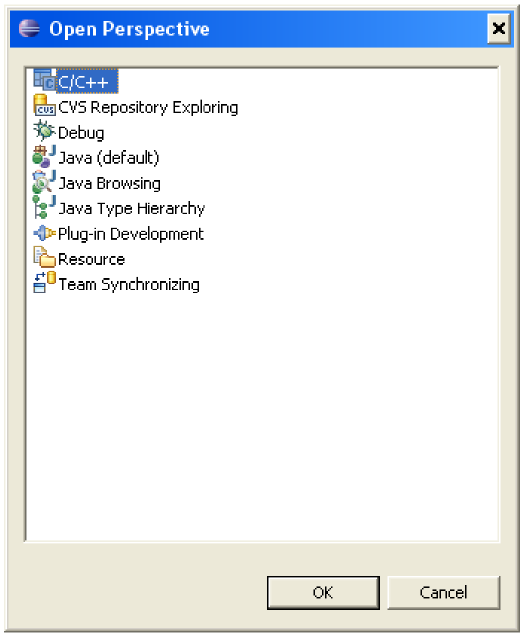
After selecting “OK” you should see a progress bar as Eclipse loads and you should eventually see the Eclipse Welcome Screen that looks like this:



Click on the Workbench icon as shown in the picture to bring up the main Eclipse Java Perspective. This is the default mode that Eclipse runs in. You may need to select it's C/C++ (instead of the "Java") perspective prior to importing one of our demo projects. Click on the Open Perspective button on the top-right of the Eclipse window (see the following picture) and select "Other…"


Java - Eclipse SDK
File Edit Refactor Source Navigate Search Project Run Window Help
Click Here
Java
Package Explorer
Hierarchy
Outline
An outline is not available.
Problems
Javadoc
Declaration
0 errors, 0 warnings, 0 infos
Description
Resource
Path
Location

This will open the “Open Perspective” dialog box as shown here:



You should see the C/C++ Perspective listed as an option at the top of the list. Select this and click “OK.”

Assuming that you selected the C/C++ Perspective and it worked, Eclipse is now properly installed and configured to work with a Macraigor JTAG/BDM interface device for developing embedded applications.

The next section will describe how to use a pre-built demo project to get all the pieces that were just installed working together to provide a fully integrated cross-development environment.

3. Using the Demo Projects

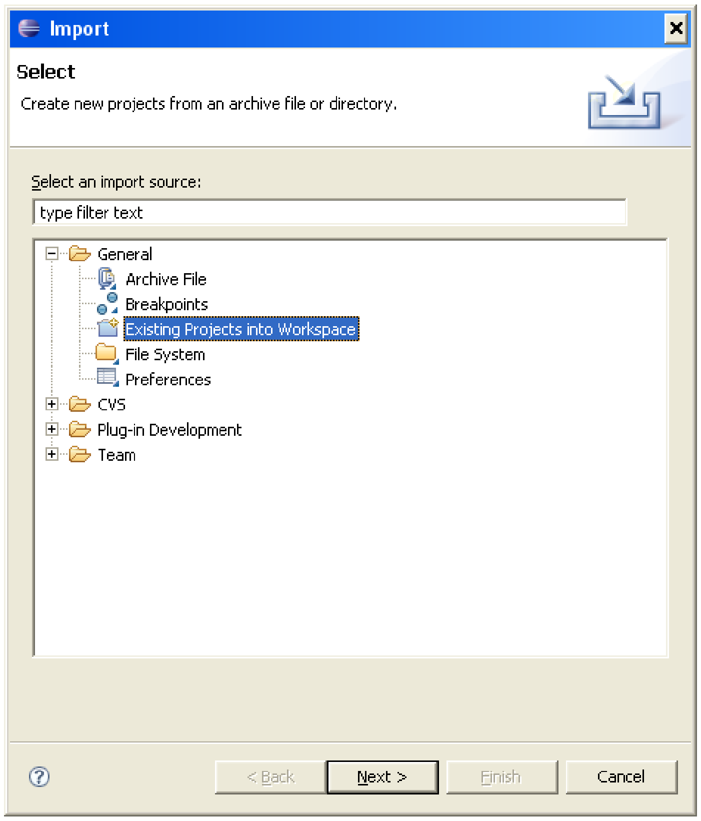
This section will explain how to import one of the installed Macraigor demo projects and will go through a simple example of using Eclipse to build code, connect to the target processor through a Macraigor JTAG/BDM device, download the code, and perform debugging tasks.

3-1 Importing a Project

The first step in using a demo project is to import the project into Eclipse. This essentially opens the project and all of its files and makes them available in the Eclipse windows for editing.

Start Eclipse, make sure that the C/C++ Perspective is open (see 2-6 if necessary) and then select File->Import…

You should see the Import dialog window. Click on the "+" next to General in the main window to expand the options, and you should see an item named "Existing Projects into Workspace." This is shown in the following picture:



Click on: "Existing Projects into Workspace" to select it and then click "Next >". In the following Import Projects dialog, select Browse… to open a navigation window. The default location of the Macraigor demos is:

Windows : C:\cygwin\usr\local\macraigor\EclipseDemos\Juno
Linux : /usr/local/macraigor/EclipseDemos/Juno .
Browse to this location or to the directory where you chose to install the Macraigor Eclipse demos. You should see several subdirectories, including:

ARM
Coldfire
MIPS
PowerPC
x86
XScale

Select the directory that corresponds to your target processor (such as ARM) and click on "OK."

You should now see a list of the available demos for your type of processor. These demos are pre-configured to work with various standard evaluation boards that are available from the processor vendor and other third-party board manufacturers. If you have one of the listed boards, you can follow through the rest of these instructions and actually get a small demo program running on your hardware. If you have a board that isn't listed, you will probably have to select a demo project that is similar to your board and then modify the demo to work on your hardware. See Appendix B for a discussion of how to modify an existing demo to work with a different board.

The rest of these instructions are going to use the Freescale_iMX21ADS demo project as an example. The instructions are the same regardless of which board/demo you use, so just substitute your project name where appropriate in the following text and screen shots.

You should be looking at a list of demo projects for several different target boards/processors that has a check box next to each demo and all of the boxes should be checked. We only want to import the demo for the board we are working with, so click on "Deselect All" to the right of the list and then click on the check box next to the demo that you wish to use.

Note that there is an option to "Copy projects into workspace." If you don't intend to modify the project, there is no need to select this. Eclipse will just work with the files where they are currently located. If, however, you plan to change the demo to work with a different board or you think that you might want to experiment with modifying the code, you can check this box and all of the demo files will be copied into the Eclipse workspace, giving you your own copy to play with without changing the files that are distributed by Macraigor.

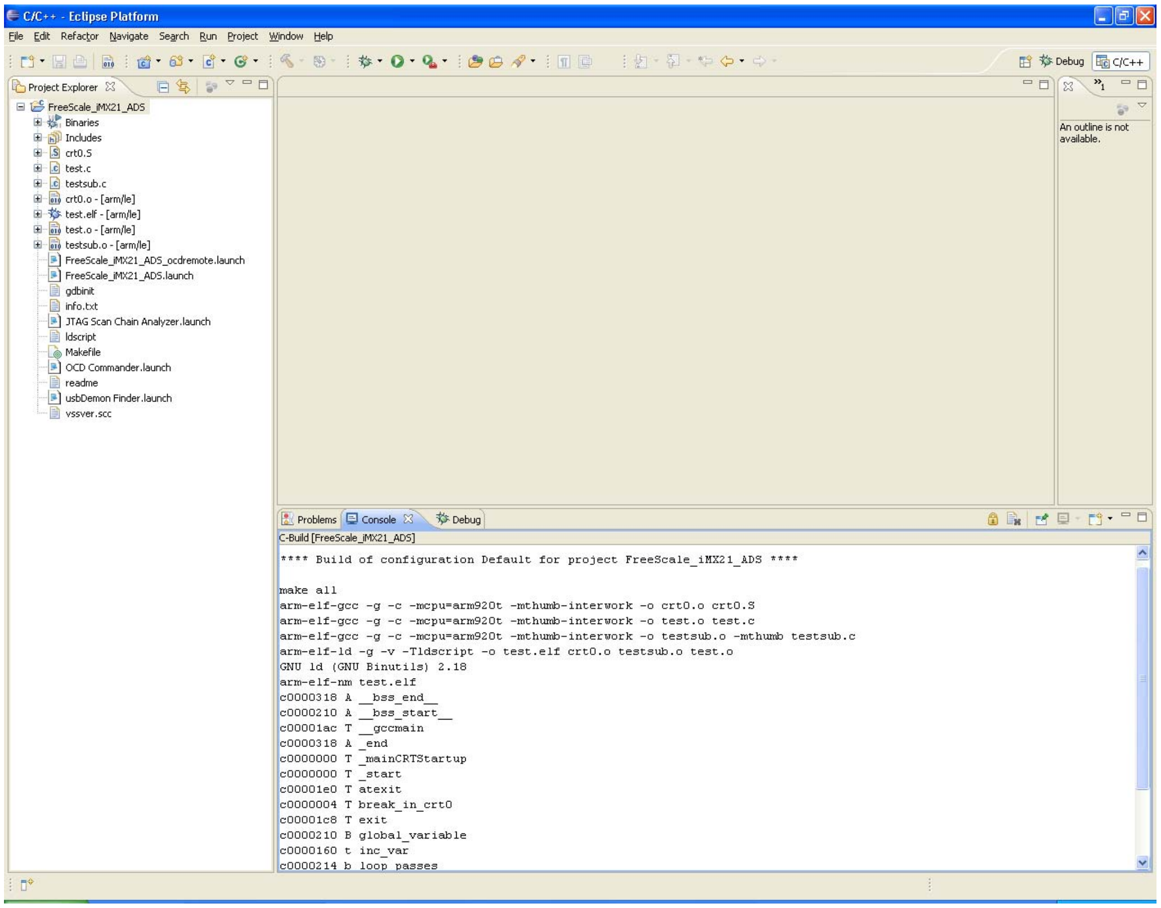
When you are ready click "Finish" to load the project into Eclipse.

3-2 Building the Code

You should now be back at the, mostly blank, C/C++ Perspective screen, but your chosen project should be displayed in the project pane at the upper left of the screen. Click on the "+" next to the project name to display the files associated with this project. You should see some source-code files like test.c and crt0.S, along with a README file (which you are encouraged to open and actually read… just double click the file name to do so), a makefile, and several files whose names end in .launch. The .launch files will be explained below.

The first thing that we need to do is compile, assemble and link the test program. To do this:
First click on the "Console" tab near the bottom of the screen. This window is where the output of the build process will be displayed.
Then in the "Project Explorer" tab window, select the project name by clicking on it, then right-click. You should see a list of several actions that you can take on a project. Near the middle of the list should be: "Build Project" followed by "Clean Project". Selecting "Build Project" (by left clicking on it) will run the Makefile to assemble, compile and link the project. "Clean Project" will clean the project directory and then (optionally) rebuild it.
Note: With the project name highlighted you can also click on the "Build All" tool bar icon (text with 010) to do a build if one or more of your source files have changed.

The screen shot below shows what you should see after building the project. The make file for the project dumps out a list of symbols in the program after building it. This is what is shown in the console window. You can scroll the Console window up to see the output from the various steps in the build process. If everything worked, you should see a new item listed on the left under the project called "test" with a picture of a green bug next to it. This is the code that was built and will be downloaded to the target processor later.



```
**** Build of configuration Default for project FreeScale_iMX21_ADS ****

make all
arm-elf-gcc -g -c -mcpu=arm920t -mthumb-interwork -o crt0.o crt0.S
arm-elf-gcc -g -c -mcpu=arm920t -mthumb-interwork -o test.o test.c
arm-elf-gcc -g -c -mcpu=arm920t -mthumb-interwork -o testsub.o -mthumb testsub.c
arm-elf-ld -g -v -Tldscript -o test.elf crt0.o testsub.o test.o
GNU ld (GNU Binutils) 2.18
arm-elf-nm test.elf
c0000318 A __bss_end__
c0000210 A __bss_start__
c00001ac T __gccmain
c0000318 A _end
c0000000 T _mainCRTStartup
c0000000 T _start
c00001e0 T atexit
c0000004 T break_in_crt0
c00001c8 T exit
c0000210 B global_variable
c0000160 t inc_var
c0000214 b loop_passes
```

Before starting up the debugger, you need to verify that both the project's gdbinit file and the debug launch file will send commands to the same debug target. The gdbserver debugger interface connects twice to the target during initialization, once when running gdbinit and once when gdbinit finishes. It's important the both connections go to the same place.
Click on the arrow next to the bug on Eclipse's toolbar

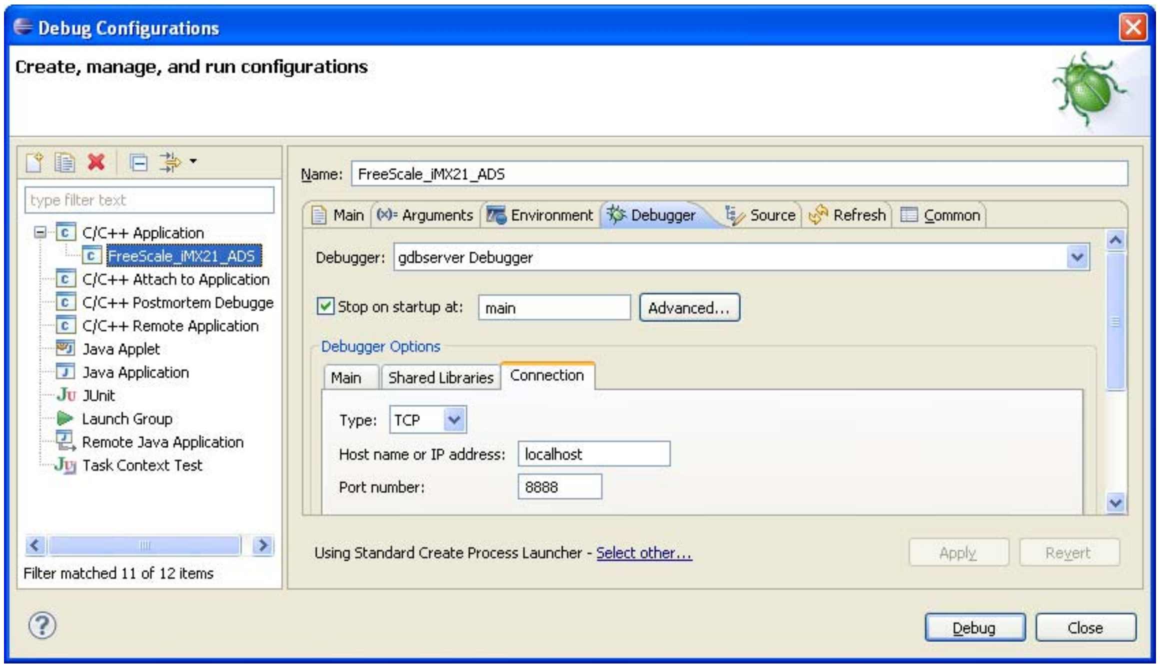
and select "Debug Configurations …". In the Debug Configuration dialog under C/C++ Application select <your project name>. Your project's debug configuration will appear. Select the Debugger tab, and in it's screen's Debugger Options, click the Connection tab. Verify that :

Type: TCP

And that

Host or IP address:
Port Number:

match the "target remote" address in your gdbinit file.



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
  - FreeScale_iMX21_ADS
- C/C++ Attach to Application
- C/C++ Postmortem Debugge
- C/C++ Remote Application
- Java Applet
- Java Application
- JUnit
- Launch Group
- Remote Java Application
- Task Context Test

Filter matched 11 of 12 items

Name: FreeScale_iMX21_ADS

Main | Arguments | Environment | Debugger | Source | Refresh | Common

Debugger: gdbserver Debugger

☑ Stop on startup at: main   Advanced...

Debugger Options

Main | Shared Libraries | Connection

Type: TCP
Host name or IP address: localhost
Port number: 8888

Using Standard Create Process Launcher - Select other...

Apply   Revert

Debug   Close

```
# CONNECT TO TARGET:
#
# Tell GDB where send it's "gdb monitor" commands:

# Tell GDB to send them to the ocdremote/ocdremoteServer monitoring
# port 8888 that is running on your PC

target remote localhost:8888
```

Your project may connect to ocdremote via a different ethernet address and/or TCP/IP port for example:

gdbinit :

target remote **127.0.0.1:8889**

Eclipse's Debug Configuration, Connection tab parameters:

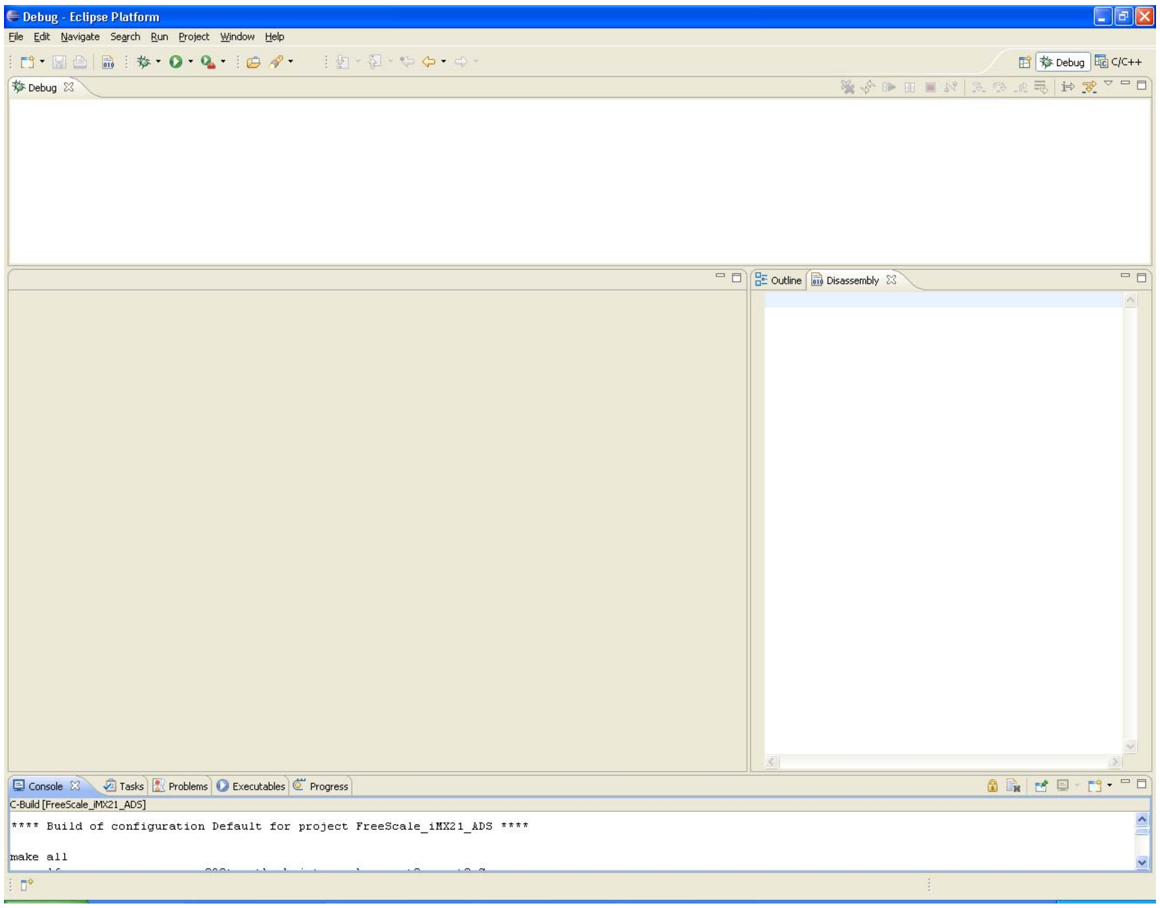
Type: TCP
Host or IP address: 127.0.0.1
Port Number: 8889

3-3 Debugging the Code

The next step is to download the code to the target processor and debug it. To do this, we need to use another perspective called Debug. Find either the Debug icon

and/or the Open Perspective icon :

on the tab in the upper-right of the Eclipse window. If the debug icon is present simply click it to enter the debug perspective. If not click the Open Perspective icon and select Debug from the menu (note that you can also open a perspective using the Window-> Open Perspective menu).
After clicking on Debug, the Eclipse windows should change and look like the following screenshot.


Debug - Eclipse Platform
File Edit Navigate Search Run Project Window Help
Debug C/C++
Debug
Outline Disassembly
Console Tasks Problems Executables Progress

C-Build [FreeScale_iMX21_ADS]

```
**** Build of configuration Default for project FreeScale_iMX21_ADS ****

make all
```

We now need to actually get connected to the target processor through the Macraigor JTAG/BDM interface device. The demo projects configure Eclipse to use the gdb debugger in the background, but gdb needs a way to connect to the processor via the Macraigor device. This task is handled by the Macraigor ocdremote or ocdremoteServer programs. These utilities act as a translator for gdb debugging commands. It starts up, initializes the Macraigor device, establishes a connection to the target processor via JTAG or BDM, opens a TCP/IP port and then waits for gdb to make a connection to the port. Once gdb connects, ocdremote or ocdremoteServer accept debugging commands from the TCP/IP port and translates these commands into the appropriate JTAG/BDM actions for the target processor.

ocdremote has to be started before each gdb starts and terminates once the gdb session ends

ocdremoteServer only has to be started once. It runs continuously after it has been started and waits on the TCP/IP ports for the next gdb connection after the current gdb session ends.
If started as an Eclipse "external tool" ocdremoteServer will exit when you exit Eclipse.
To stop ocdremoteServer started from a Linux or Cygwin shell enter the CTRL-c key combination in its parent shell window.
If ocdremoteServer is running in the background it can be stopped using the kill command in a shell window by:
1) Getting its process id :

```
        ps -e | grep ocdremoteServer
```

2) Issuing a kill command on that process:

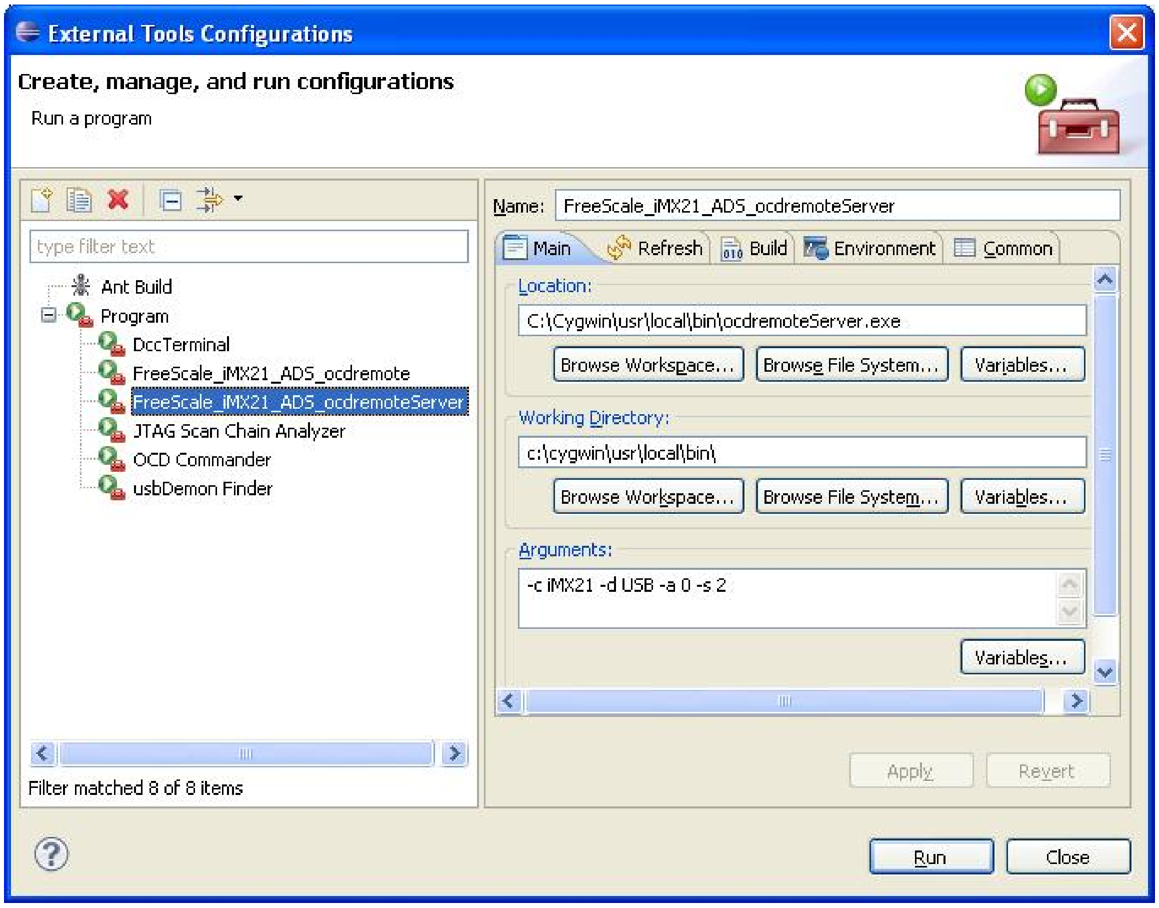
```
 kill -9 <ocdremoteServer PPID>
```

Eclipse doesn’t know anything about ocdremote/ocdremoteServer or the need for a target connection, but it does provide a mechanism for running external programs. The demo projects use this mechanism to start ocdremote/ocdremoteServer with the correct arguments for the target processor that we are using and the specific Macraigor device that is being used. This setup information is saved in .launch files that are stored as part of the project.

Find the External Tools button on the Eclipse button bar just under the menu at the top of the screen. It looks like this:

Clicking on the down-arrow on the right side of the button will open a Favorites menu and provide a quick way to launch OCDRemote (or other programs) in the future. For now, we want to look through the details of the OCDRemote launch configuration, so click on the left side of the button and select “External Tools

Configurations …” to open the External Tools interface. The window should look like the following:


External Tools Configurations

Create, manage, and run configurations

Run a program

type filter text

- Ant Build
- Program
  - DccTerminal
  - FreeScale_iMX21_ADS_ocdremote
  - FreeScale_iMX21_ADS_ocdremoteServer
  - JTAG Scan Chain Analyzer
  - OCD Commander
  - usbDemon Finder

Filter matched 8 of 8 items

Name: FreeScale_iMX21_ADS_ocdremoteServer

Main | Refresh | Build | Environment | Common

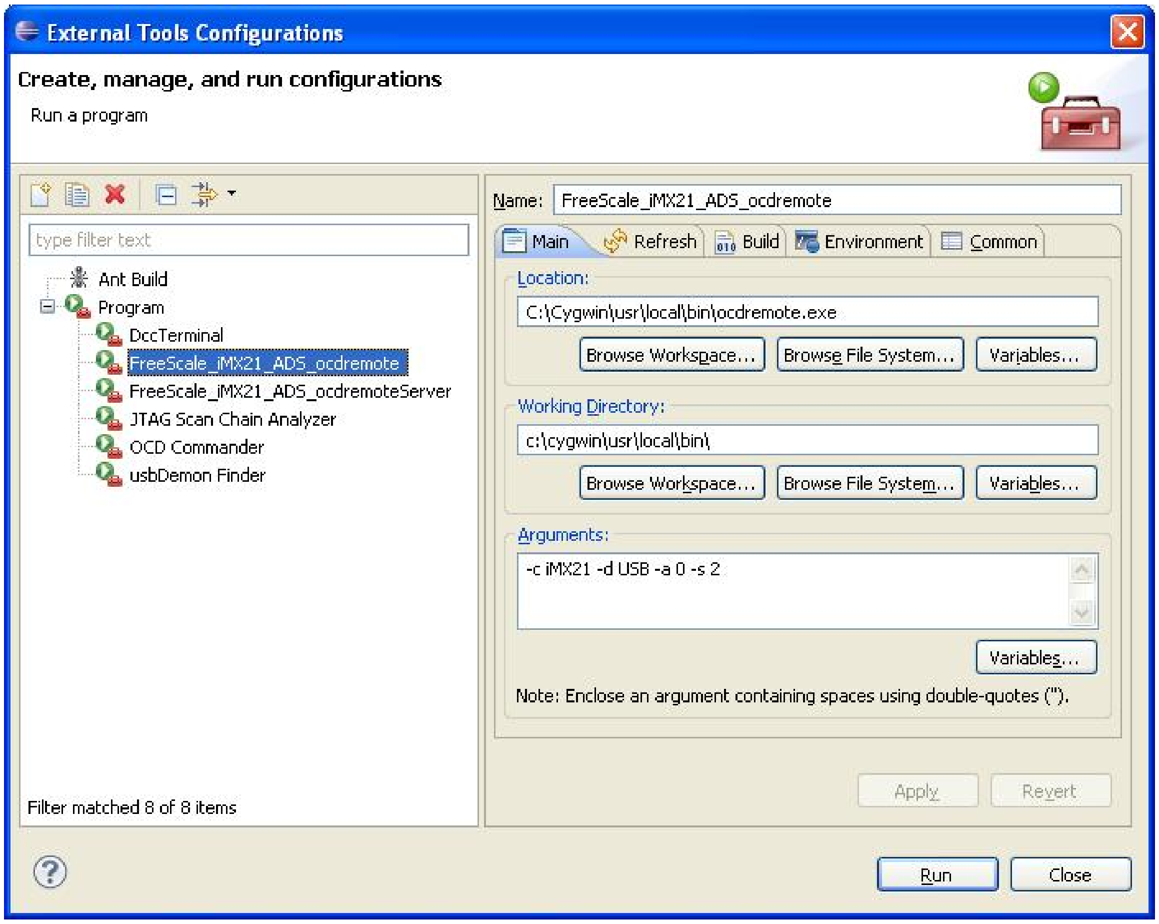
Location:
C:\Cygwin\usr\local\bin\ocdremoteServer.exe

Working Directory:
c:\cygwin\usr\local\bin\

Arguments:
-c iMX21 -d USB -a 0 -s 2

Note that you may have to click on the “+” next to Program in order to expand the list. As in the above screenshot, you should see several existing launch configurations. There should be two with the same name as your project but with one with “_ocdremote” and one with _ocdremoteServer appended to it. These are the entries that we need to work with. The other entries provide an easy way to launch Macraigor utility programs that can be useful for debugging problems with the environment. Appendix D discusses how these programs can be used to help find and correct connection and debugging issues.

Click on the entry that ends in "_ocdremote" and you should see the details of this particular launch configuration, as shown in the following picture:



Name: FreeScale_iMX21_ADS_ocdremote

Location: C:\Cygwin\usr\local\bin\ocdremote.exe

Working Directory: c:\cygwin\usr\local\bin\

Arguments: -c iMX21 -d USB -a 0 -s 2

Note: Enclose an argument containing spaces using double-quotes (").

All of the important details about this configuration are shown on the Main tab. The Location field should be set to the full path to the OCDRemote executable. This is typically

Windows: C:\Cygwin\usr\local\bin\ocdremote.exe
Linux : /usr/local/bin/ocdremote

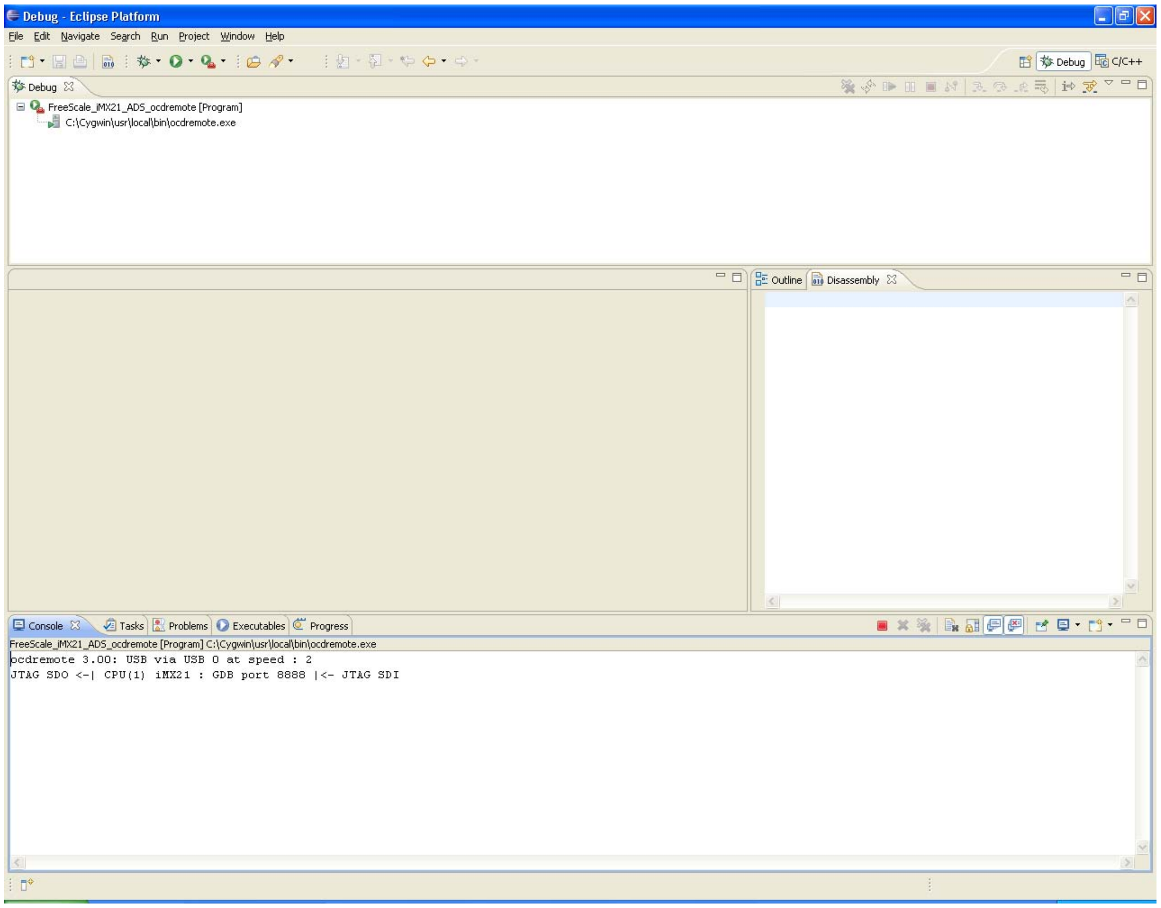
If your Cygwin installation is on a drive other than C: or is installed in a different directory, you can click on the Browse File System button, find ocdremote.exe and set the correct location. The working directory can be left as is.

The Arguments window is where program arguments are passed to ocdremote, just as if it were being run from a command line. These arguments need to correctly specify the type of target processor to connect to and the type of Macraigor interface device that is being used. The available options for ocdremote / ocdremoteServer are described and listed in Appendix A. If you are working with a standard demo board and project, the –c argument is probably already correct for your processor. The –d argument, which selects the type of Macraigor device to use, will generally be pre-set to USB. If you are using a single usb2Demon, usb2Sprite or usbWiggler, you can leave this argument as it is. If you are using a different Macraigor device or have multiple Macraigor USB devices connected to your host computer, you will need to change the argument to match your hardware interface. Refer to Appendix A for the available options and examples and change the text as necessary.

The –s option specifies the JTAG/BDM clock speed. This setting is a divisor on the raw clock speed of the interface device, so lower speed settings correspond to higher clock rates. Speed 1 is the fastest speed and 8 the slowest. The demo projects are generally set to use speed 2 as a precaution. If everything is working for you and you would like faster downloads and debugging, you can experiment with setting the speed to 1 and then rechecking that everything still works. Conversely, if you are having trouble getting connected to your board and/or downloading code, you can try changing the speed to a slower setting.

If you changed any of the arguments, make sure to click on the Apply button to save the changes. When everything looks correct, turn on the target board, make sure that the Macraigor device is properly connected to the JTAG/BDM header on the board and to the appropriate host interface cable (USB, Parallel or Ethernet) and then click on the Run button to start OCDRemote. You should again see the Debug perspective, but there should now be an entry in the upper-left Debug window showing that OCDRemote is running. OCDRemote will start up and attempt to initialize a connection to the target processor. This process takes a few seconds. If everything works correctly, OCDRemote will output a startup message to the Console window at the bottom of the screen.

A successful launch of OCDRemote should look like the following screen shot:


Debug - Eclipse Platform
File Edit Navigate Search Run Project Window Help
Debug
C/C++
Debug
FreeScale_iMX21_ADS_ocdremote [Program]
C:\Cygwin\usr\local\bin\ocdremote.exe
Outline
Disassembly
Console
Tasks
Problems
Executables
Progress

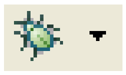
FreeScale_iMX21_ADS_ocdremote [Program] C:\Cygwin\usr\local\bin\ocdremote.exe

```
ocdremote 3.00: USB via USB 0 at speed : 2
JTAG SDO <-| CPU(1) iMX21 : GDB port 8888 |<- JTAG SDI
```

If you get an error message in the console window, double check that your target board is turned on, that the Macraigor device is connected properly (with Pin 1 in the correct location) and that the arguments to OCDRemote match your setup and then try to start OCDRemote again. If you continue to have problems getting ocdremote or ocdremoteServer to start, please refer to the Troubleshooting section below.

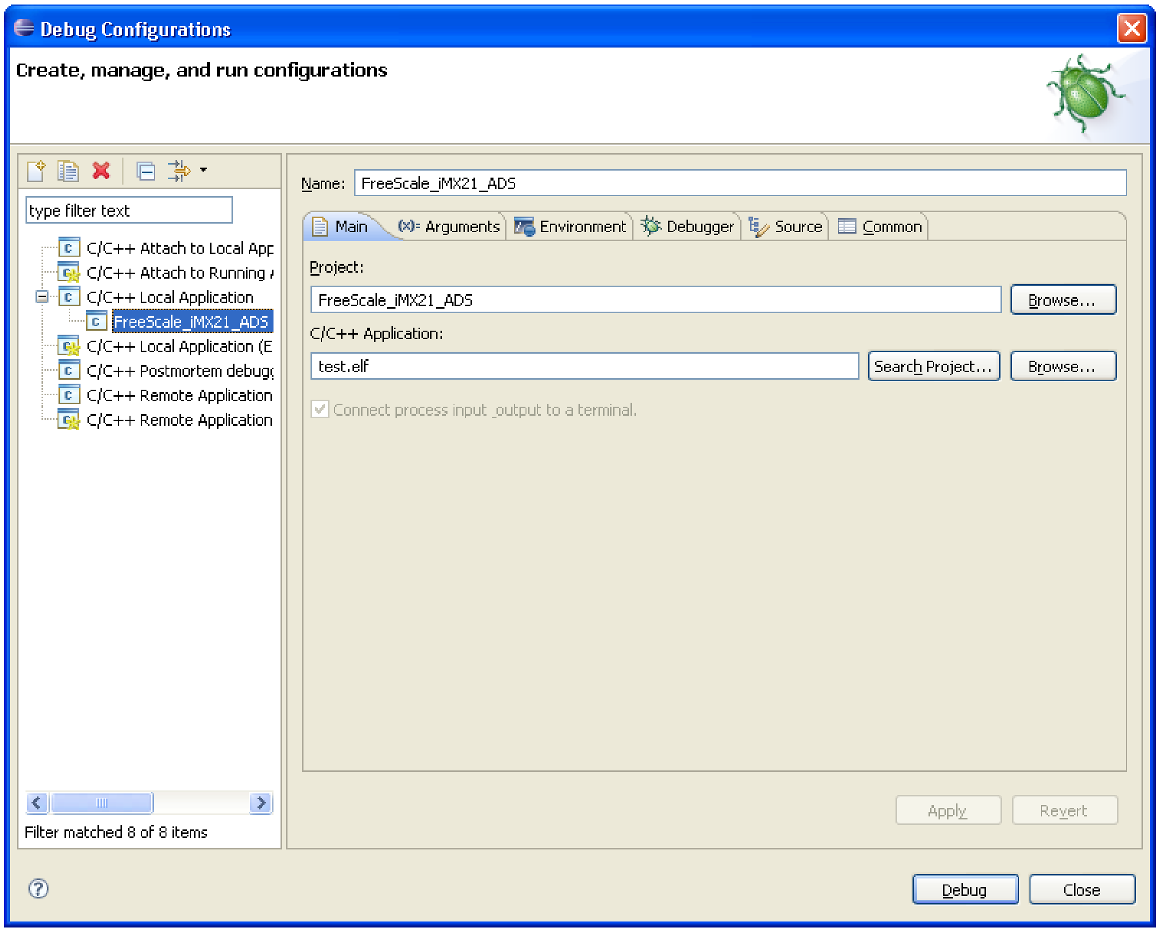
Now that ocdremote or ocdremoteServer is running and waiting for a connection from gdb, we need to start the debugger. Eclipse and the plug-ins that were installed provide a seamless integration between the gdb debugger and the Eclipse environment. All the details of downloading code, setting breakpoints, reading and writing variables and registers and memory, etc. are handled by gdb in the background, but the results are displayed by Eclipse in the windows of the Debug perspective.

In order to run gdb from within Eclipse, we will use another launch configuration similar to the one used above for OCDRemote. Find the Debug button on the Eclipse toolbar under the menu options. It looks like this:



Similar to the External Tools button, the down-arrow on the right side of the button provides access to a favorites menu that can be used in the future to quickly launch the debugger. For now, we want to see what will be done by the Debug launch configuration, so click on the left side of the button.

You should now see the following Debug configuration menu:



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Attach to Local App
- C/C++ Attach to Running
- C/C++ Local Application
  - FreeScale_iMX21_ADS
- C/C++ Local Application (E
- C/C++ Postmortem debug
- C/C++ Remote Application
- C/C++ Remote Application

Filter matched 8 of 8 items

Name: FreeScale_iMX21_ADS

Main | Arguments | Environment | Debugger | Source | Common

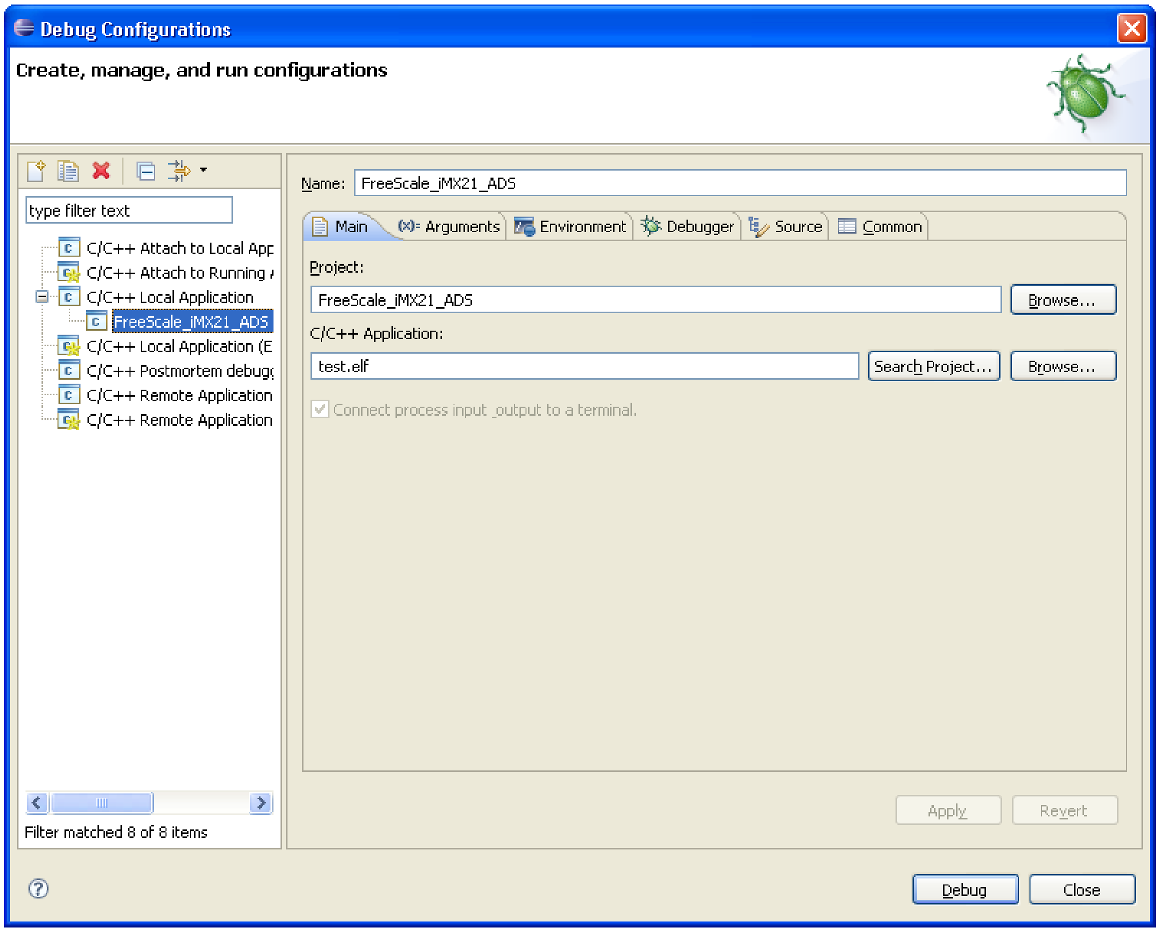
Project: FreeScale_iMX21_ADS

C/C++ Application: test.elf

☑ Connect process input _output to a terminal.

There should be a "+" next to the entry on the left that says: "C/C++ Local Application."

Click on the “+” to expand the entry and you should see a sub-entry with the same name as your project. Click on this name to select it and you should then see the options for this Debug launch configuration. The screen should now look like the following screen shot:



Name: FreeScale_iMX21_ADS

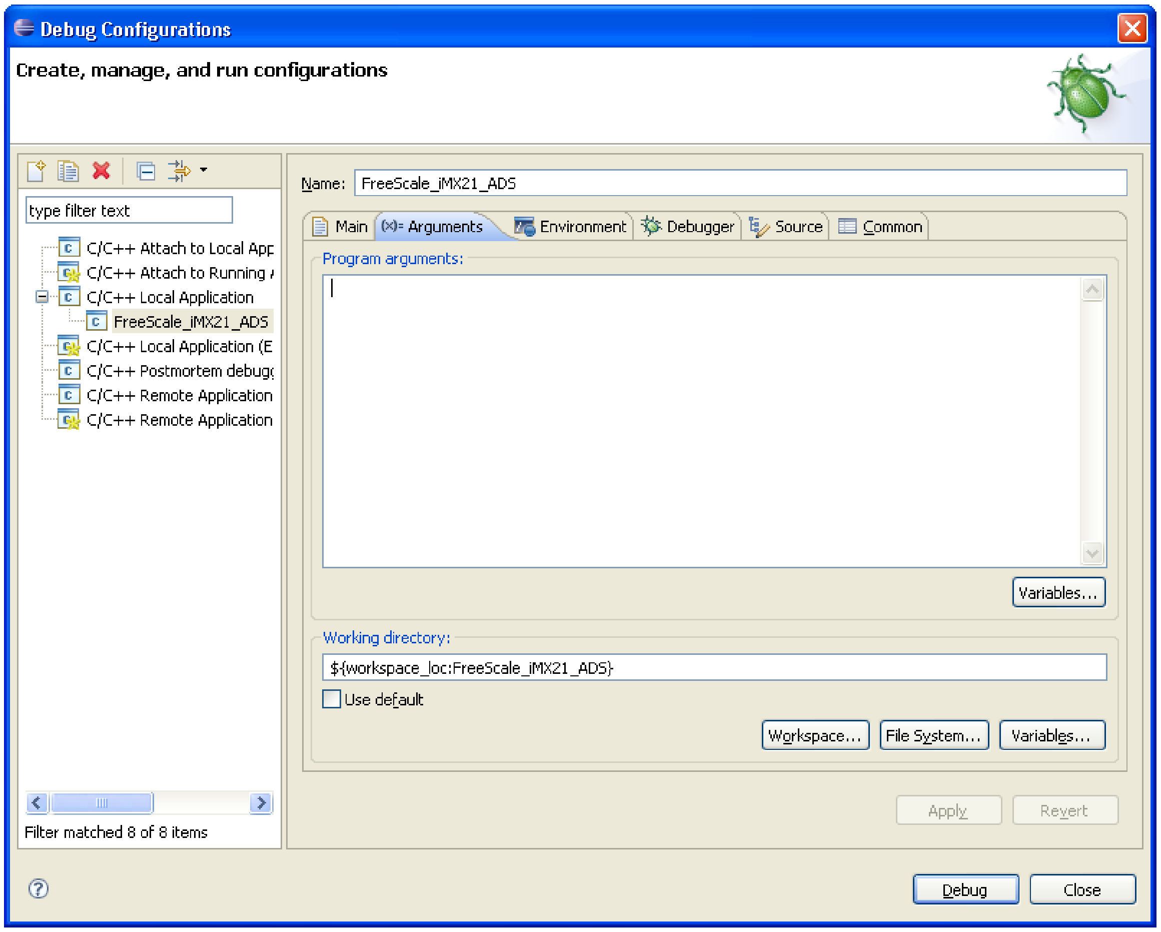
Project: FreeScale_iMX21_ADS

C/C++ Application: test.elf

☑ Connect process input _output to a terminal.

You should see your project name in the Project entry and “test.elf” under the “C/C++ Application” entry. This is the name of the application that will be downloaded to the target and debugged. The “test.elf” file in our case is an ELF file which contains both the object code that will be downloaded and debug information such as file names and line numbers of entry points for use by gdb.

Click on the Arguments tab. It should be blank except that the “Working Directory” is specified as ${workspace_loc:<your project name>} and the “use default” checkbox is unchecked as in this screen shot:


Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Attach to Local App
- C/C++ Attach to Running A
- C/C++ Local Application
  - FreeScale_iMX21_ADS
- C/C++ Local Application (E
- C/C++ Postmortem debugg
- C/C++ Remote Application
- C/C++ Remote Application

Filter matched 8 of 8 items

Name: FreeScale_iMX21_ADS

Main | Arguments | Environment | Debugger | Source | Common

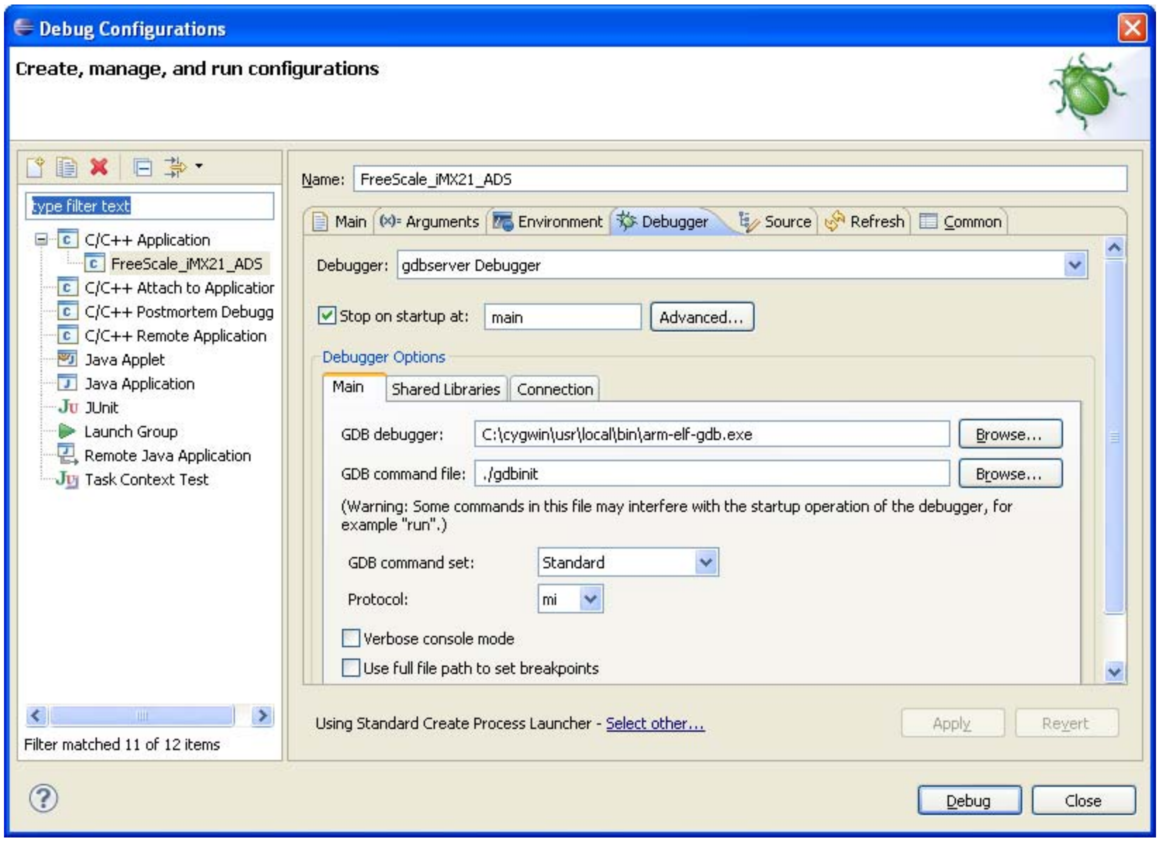
Program arguments:

Working directory:

${workspace_loc:FreeScale_iMX21_ADS}

☐ Use default

Click on the Debugger tab and you should see that the demo is calling the appropriate target-version of the gdb debugger for your processor. In this example, it is set to “arm-elf-gdb”. Your particular demo might be using “powerpc-elf-gdb”, “mips-elf-gdb”, “m68k-elf-gdb” or “i386-elf-gdb”. Also shown are the type of debug session (remote gdb/mi) and where gdb will initially halt once the load image is downloaded to the target and run in the “Stop on startup at” text box. Our examples usually stop at main (the first line of example C function). A label “break_in_crt0” has been put in our example crt0.S file and by entering “break_in_crt0” in this field you can have gdb stop in the assembly code instead.
A “GDB command file” is specified. This file is used to setup gdb and perform any target setup operations (writing to configuration registers, initializing RAM) required. Here is the Freescale_iMX21’s screen shot for this dialog:



Create, manage, and run configurations

Name: FreeScale_iMX21_ADS

Debugger: gdbserver Debugger

[✓] Stop on startup at: main

GDB debugger: C:\cygwin\usr\local\bin\arm-elf-gdb.exe

GDB command file: ./gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

GDB command set: Standard

Protocol: mi

[ ] Verbose console mode

[ ] Use full file path to set breakpoints

Using Standard Create Process Launcher - Select other...

Filter matched 11 of 12 items

The specific gdbinit file commands for each target board vary widely and are beyond the scope of this document. You can look at the gdbinit file for your project by returning to the C/C++ perspective and double-clicking on “gdbinit” on the left to edit the file. The file is simply text with one gdb command per line and should be well-commented.

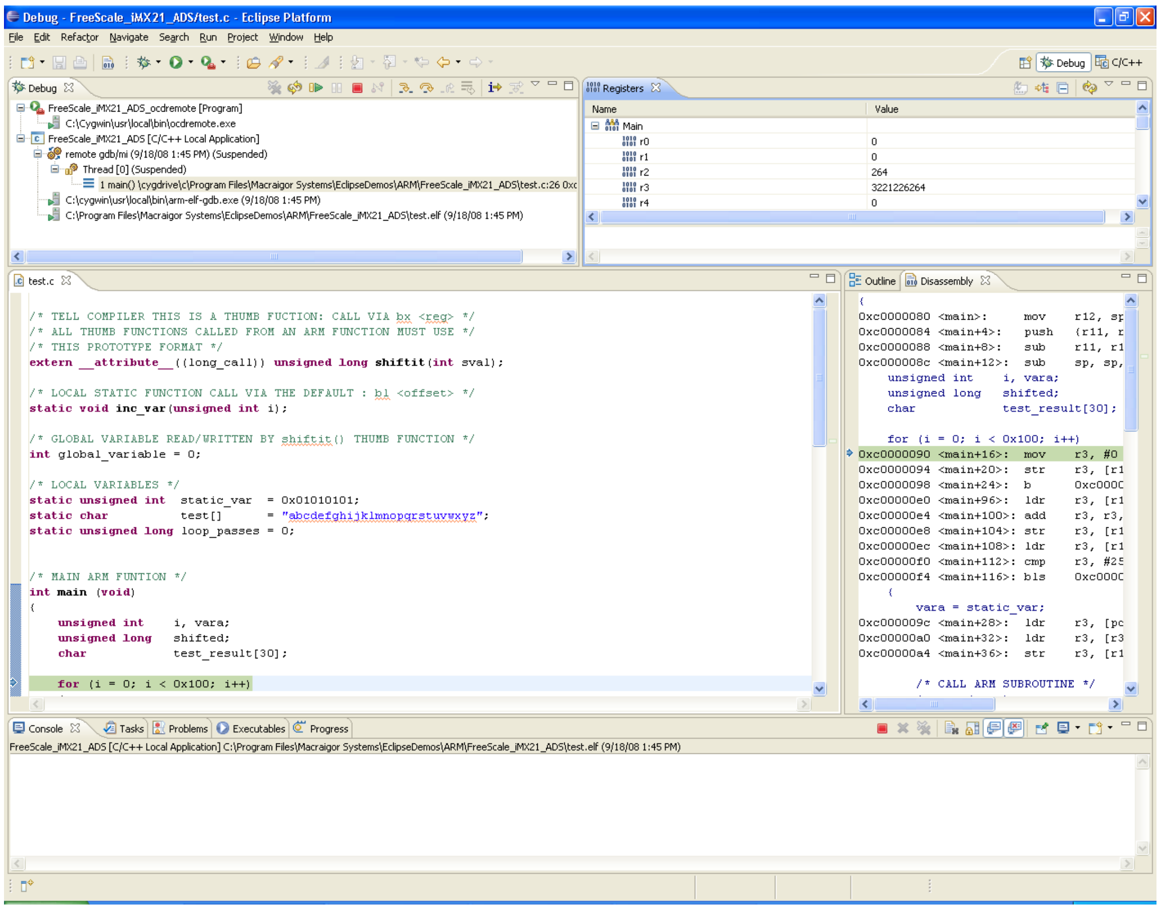
NOTE : Linux users will need to check the following line :

```
target remote 127.0.0.1:8888
```

in their gdbinit file. DSDP does not recognize localhost on Linux systems as a valid target address. So instead the default Linux local host ethernet address 127.0.0.1 is used. Check your /etc/hosts file to verify that this address is your system's localhost address.

The remaining three tabs, Environment, Source and Common, don't contain much of interest to us at the moment. When you are ready, click on the Debug button at the bottom-right of the window to start gdb, download the code, and begin to debug the application.

As gdb starts up and runs the commands from the gdbinit file, you should see messages in the console window at the bottom of the screen. Once the file has been downloaded, the processor is run and the breakpoint at main() should be hit. Eclipse will then display the source code at the location of the breakpoint (which will be in the C module test.c) and will also display and update the local variables window. The Debug window should now contain another entry for the debug process. A sub-entry of this will say something like "Thread[0] (Suspended)". This "thread" is the application that was downloaded and run and it is currently suspended because execution was halted by the breakpoint at main(). A stack trace is displayed as sub-entries of the thread and should show that we are currently stopped at main(). A small, blue arrow on the left of the source window shows the current location of the Program Counter and various other windows, such as a register display, are available on tabs in the upper-right window.


Debug - FreeScale_iMX21_ADS/test.c - Eclipse Platform
File Edit Refactor Navigate Search Run Project Window Help
Debug
C/C++
Registers
Outline
Disassembly
Console
Tasks
Problems
Executables
Progress

```
FreeScale_iMX21_ADS_ocdremote [Program]
    C:\Cygwin\usr\local\bin\ocdremote.exe
FreeScale_iMX21_ADS [C/C++ Local Application]
    remote gdb/mi (9/18/08 1:45 PM) (Suspended)
        Thread [0] (Suspended)
            1 main() \cygdrive\c\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.c:26 0xc
    C:\cygwin\usr\local\bin\arm-elf-gdb.exe (9/18/08 1:45 PM)
    C:\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.elf (9/18/08 1:45 PM)
```

Name	Value
Main	
r0	0
r1	0
r2	264
r3	3221226264
r4	0

test.c

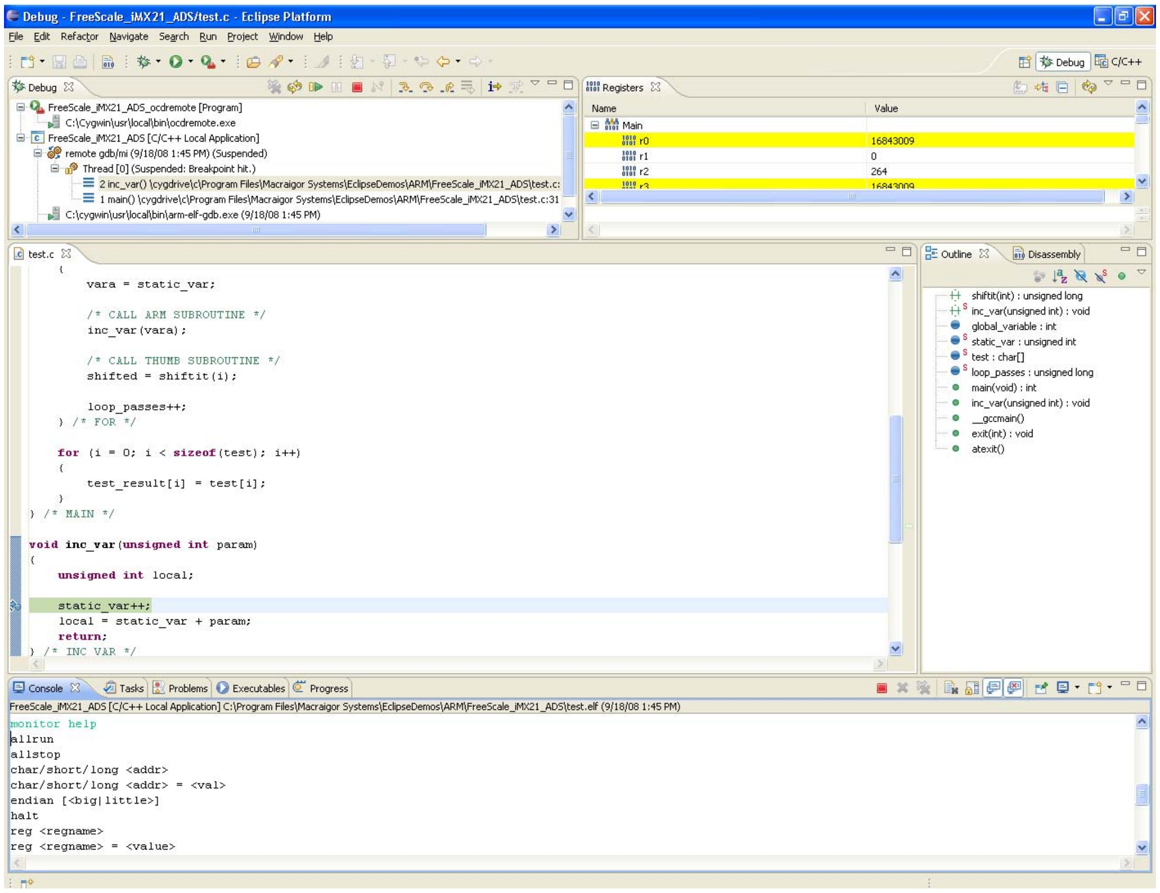
```
/* TELL COMPILER THIS IS A THUMB FUCTION: CALL VIA bx <reg> */
/* ALL THUMB FUNCTIONS CALLED FROM AN ARM FUNCTION MUST USE */
/* THIS PROTOTYPE FORMAT */
extern __attribute__((long_call)) unsigned long shiftit(int sval);

/* LOCAL STATIC FUNCTION CALL VIA THE DEFAULT : bl <offset> */
static void inc_var(unsigned int i);

/* GLOBAL VARIABLE READ/WRITTEN BY shiftit() THUMB FUNCTION */
int global_variable = 0;

/* LOCAL VARIABLES */
static unsigned int  static_var  = 0x01010101;
static char          test[]      = "abcdefghijklmnopqrstuvwxyz";
static unsigned long loop_passes = 0;


/* MAIN ARM FUNTION */
int main (void)
{
    unsigned int    i, vara;
    unsigned long   shifted;
    char            test_result[30];

    for (i = 0; i < 0x100; i++)
```

Disassembly

```
{
0xc0000080 <main>:      mov    r12, sp
0xc0000084 <main+4>:    push   {r11, r
0xc0000088 <main+8>:    sub    r11, r1
0xc000008c <main+12>:   sub    sp, sp,
    unsigned int    i, vara;
    unsigned long   shifted;
    char            test_result[30];

    for (i = 0; i < 0x100; i++)
0xc0000090 <main+16>:   mov    r3, #0
0xc0000094 <main+20>:   str    r3, [r1
0xc0000098 <main+24>:   b      0xc000C
0xc00000e0 <main+96>:   ldr    r3, [r1
0xc00000e4 <main+100>:  add    r3, r3,
0xc00000e8 <main+104>:  str    r3, [r1
0xc00000ec <main+108>:  ldr    r3, [r1
0xc00000f0 <main+112>:  cmp    r3, #25
0xc00000f4 <main+116>:  bls    0xc000C
    {
        vara = static_var;
0xc000009c <main+28>:   ldr    r3, [pc
0xc00000a0 <main+32>:   ldr    r3, [r3
0xc00000a4 <main+36>:   str    r3, [r1

        /* CALL ARM SUBROUTINE */
```

```
FreeScale_iMX21_ADS [C/C++ Local Application] C:\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.elf (9/18/08 1:45 PM)
```

You can set software breakpoints in the code by simply double-clicking in the gray bar on the left of the source window next to the source line where you want the breakpoint. The following screen shot shows how Eclipse should look after successfully starting gdb, loading the code, hitting the breakpoint at main() and then setting another breakpoint further down in the source file.



Debug - FreeScale_iMX21_ADS/test.c - Eclipse Platform
File Edit Refactor Navigate Search Run Project Window Help
Debug
C/C++
Registers
Outline
Disassembly
Tasks
Problems
Executables
Progress

```
FreeScale_iMX21_ADS_ocdremote [Program]
    C:\Cygwin\usr\local\bin\ocdremote.exe
FreeScale_iMX21_ADS [C/C++ Local Application]
    remote gdb/mi (9/18/08 1:45 PM) (Suspended)
        Thread [0] (Suspended: Breakpoint hit.)
            2 inc_var() \cygdrive\c\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.c:
            1 main() \cygdrive\c\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.c:31
    C:\cygwin\usr\local\bin\arm-elf-gdb.exe (9/18/08 1:45 PM)
```

Name	Value
Main	
r0	16843009
r1	0
r2	264
r3	16843009

```
    {
        vara = static_var;

        /* CALL ARM SUBROUTINE */
        inc_var(vara);

        /* CALL THUMB SUBROUTINE */
        shifted = shiftit(i);

        loop_passes++;
    } /* FOR */

    for (i = 0; i < sizeof(test); i++)
    {
        test_result[i] = test[i];
    }
} /* MAIN */

void inc_var(unsigned int param)
{
    unsigned int local;

    static_var++;
    local = static_var + param;
    return;
} /* INC VAR */
```

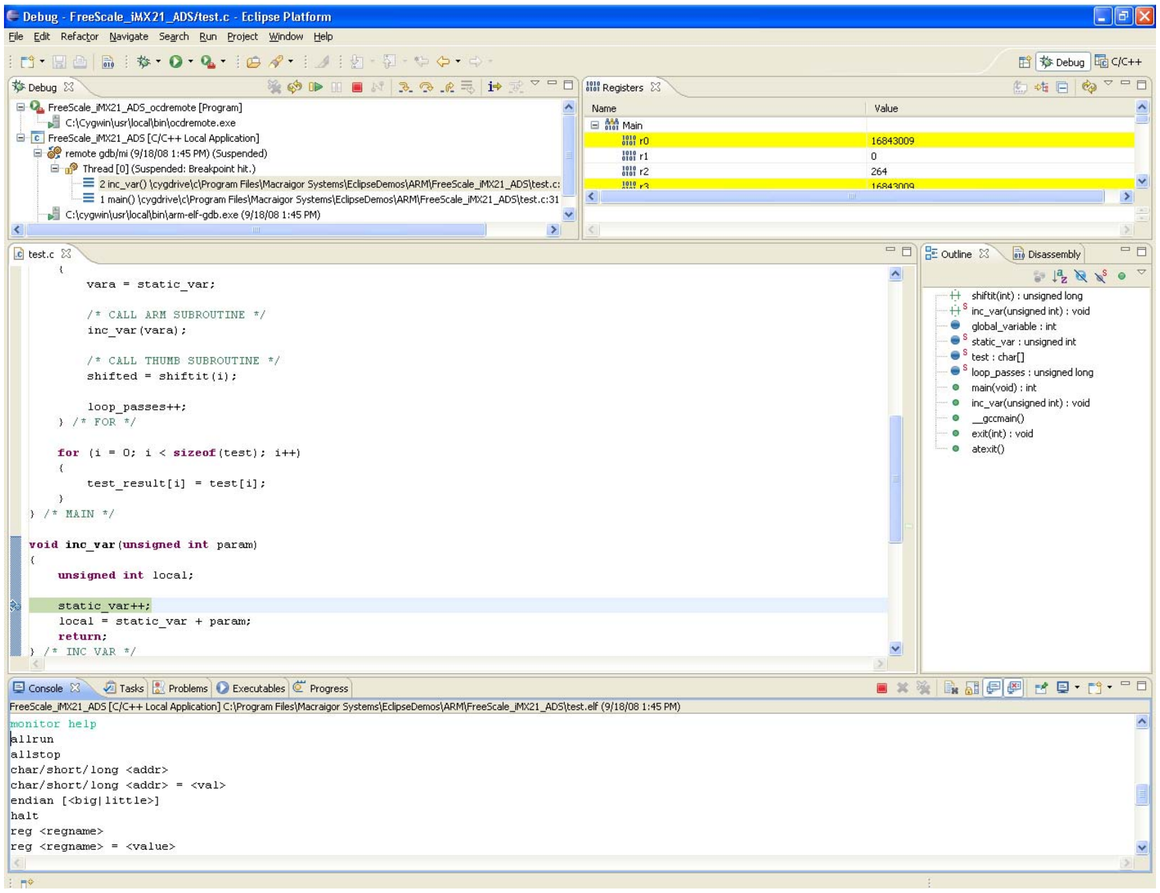
```
shiftit(int) : unsigned long
inc_var(unsigned int) : void
global_variable : int
static_var : unsigned int
test : char[]
loop_passes : unsigned long
main(void) : int
inc_var(unsigned int) : void
__gccmain()
exit(int) : void
atexit()
```

```
FreeScale_iMX21_ADS [C/C++ Local Application] C:\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.elf (9/18/08 1:45 PM)
monitor help
allrun
allstop
char/short/long <addr>
char/short/long <addr> = <val>
endian [<big|little>]
halt
reg <regname>
reg <regname> = <value>
```

Once you have set another breakpoint, you can resume execution of the program by clicking on the Resume button above the Debug pane. The button looks like this:

When you click this button, the processor is allowed to continue running until it hits another breakpoint or is otherwise stopped or interrupted.

The following screenshot shows Eclipse after the newly-set breakpoint is hit.


Debug - FreeScale_iMX21_ADS/test.c - Eclipse Platform
File Edit Refactor Navigate Search Run Project Window Help
Debug
C/C++
Registers
Outline
Disassembly
Console
Tasks
Problems
Executables
Progress

```
FreeScale_iMX21_ADS_ocdremote [Program]
    C:\Cygwin\usr\local\bin\ocdremote.exe
FreeScale_iMX21_ADS [C/C++ Local Application]
    remote gdb/mi (9/18/08 1:45 PM) (Suspended)
        Thread [0] (Suspended: Breakpoint hit.)
            2 inc_var() \cygdrive\c\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.c:
            1 main() \cygdrive\c\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.c:31
    C:\cygwin\usr\local\bin\arm-elf-gdb.exe (9/18/08 1:45 PM)
```

Name	Value
Main	
r0	16843009
r1	0
r2	264
r3	16843009

```
    {
        vara = static_var;

        /* CALL ARM SUBROUTINE */
        inc_var(vara);

        /* CALL THUMB SUBROUTINE */
        shifted = shiftit(i);

        loop_passes++;
    } /* FOR */

    for (i = 0; i < sizeof(test); i++)
    {
        test_result[i] = test[i];
    }
} /* MAIN */

void inc_var(unsigned int param)
{
    unsigned int local;

    static_var++;
    local = static_var + param;
    return;
} /* INC VAR */
```

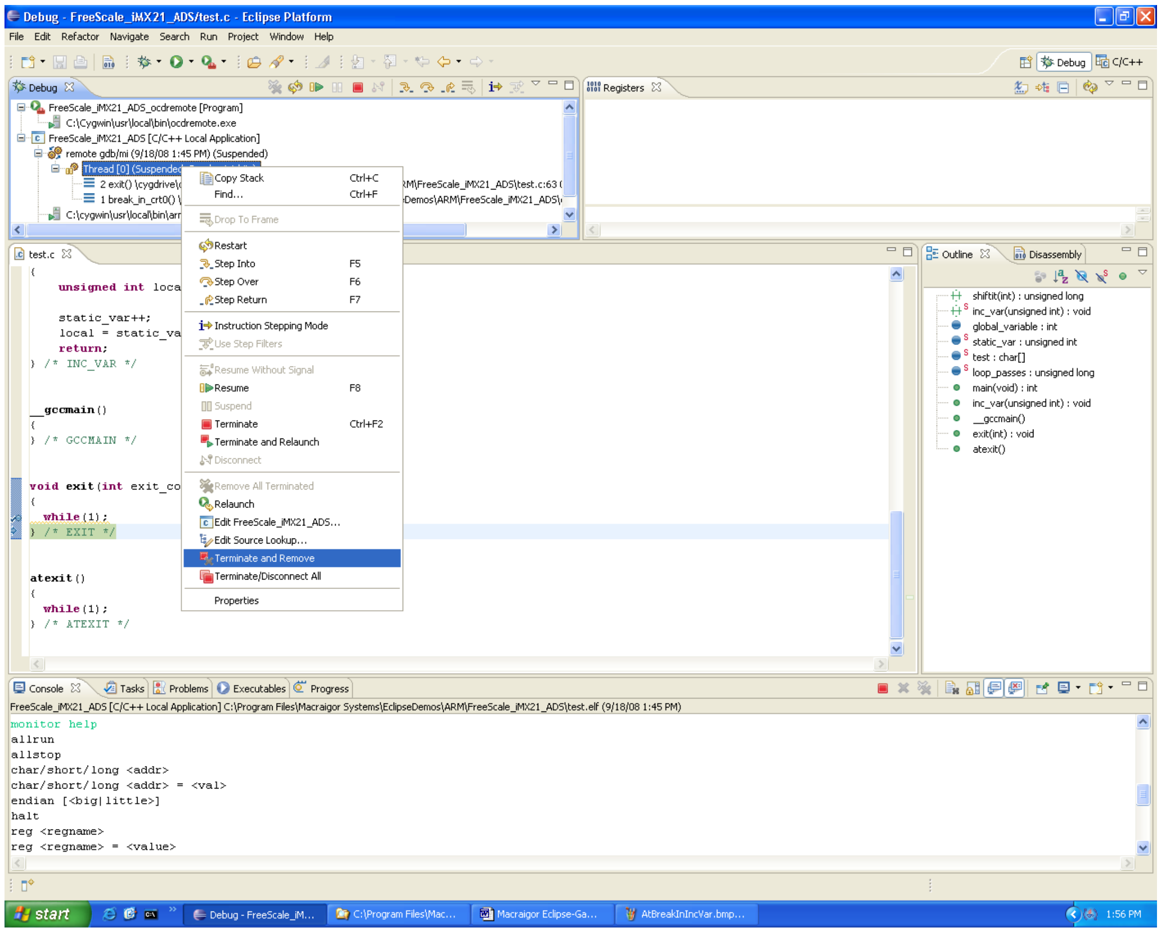
```
shiftit(int) : unsigned long
inc_var(unsigned int) : void
global_variable : int
static_var : unsigned int
test : char[]
loop_passes : unsigned long
main(void) : int
inc_var(unsigned int) : void
__gccmain()
exit(int) : void
atexit()
```

```
FreeScale_iMX21_ADS [C/C++ Local Application] C:\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.elf (9/18/08 1:45 PM)
monitor help
allrun
allstop
char/short/long <addr>
char/short/long <addr> = <val>
endian [<big|little>]
halt
reg <regname>
reg <regname> = <value>
```

Eclipse provides many other features that would be found in any typical debugger, including the ability to modify variables, modify registers, step through source code, instruction step, display assembly code (with or without inline source statements), etc. If you have trouble finding a debug feature or action that you need, please refer to the built-in help system within Eclipse.

Most of the typical debugging tasks that you will need to perform can be handled within the Eclipse GUI. However, certain target processors may have features or capabilities that Eclipse can’t access. For instance, there may be registers that are specific to your particular PowerPC processor that aren’t displayed in the normal Register window. This issue is addressed partly by gdb and partly by OCDRemote. The Console window at the bottom of the screen can also be used for user input to gdb. If you place the cursor in the Console window and hit Enter, you will be given a gdb prompt. Here you can type any gdb command that you could normally enter when using gdb from a command line (see Appendix C for a list of some useful gdb commands). One of the commands that gdb implements is called “monitor”. This command provides a mechanism for passing arbitrary information to a “debug monitor”. In our case, the debug monitor is either ocdremote or ocdremoteServer. These utilities implement a number of monitor commands for each processor that extend the functionality of gdb (and, therefore, Eclipse) to cover the non-standard features of various embedded processors. For example, ocdremote and ocdremoteServer are generally capable of displaying ALL the registers that are available on a given processor. To see what monitor commands are supported for your processor, place the cursor in the console window, hit Enter, then type “monitor help”. You will be presented with a list of monitor commands that can be called for your particular processor.

We have now successfully built, downloaded, and debugged the example program running live on the evaluation hardware. The final thing that we might typically want to do is quit the debugger and go back to the C/C++ perspective in order to fix something in the source code or add more code. The easiest way to stop execution of the debugger is to right-click on the thread in the debug window and select “Terminate and Remove” from the menu that is displayed. This is shown in the following screenshot.



```
FreeScale_iMX21_ADS_ocdremote [Program]
    C:\Cygwin\usr\local\bin\ocdremote.exe
FreeScale_iMX21_ADS [C/C++ Local Application]
    remote gdb/mi (9/18/08 1:45 PM) (Suspended)
        Thread [0] (Suspended
            2 exit() \cygdrive\ ...RM\FreeScale_iMX21_ADS\test.c:63
            1 break_in_crt0() \ ...Demos\ARM\FreeScale_iMX21_ADS\
    C:\cygwin\usr\local\bin\arr
```

test.c

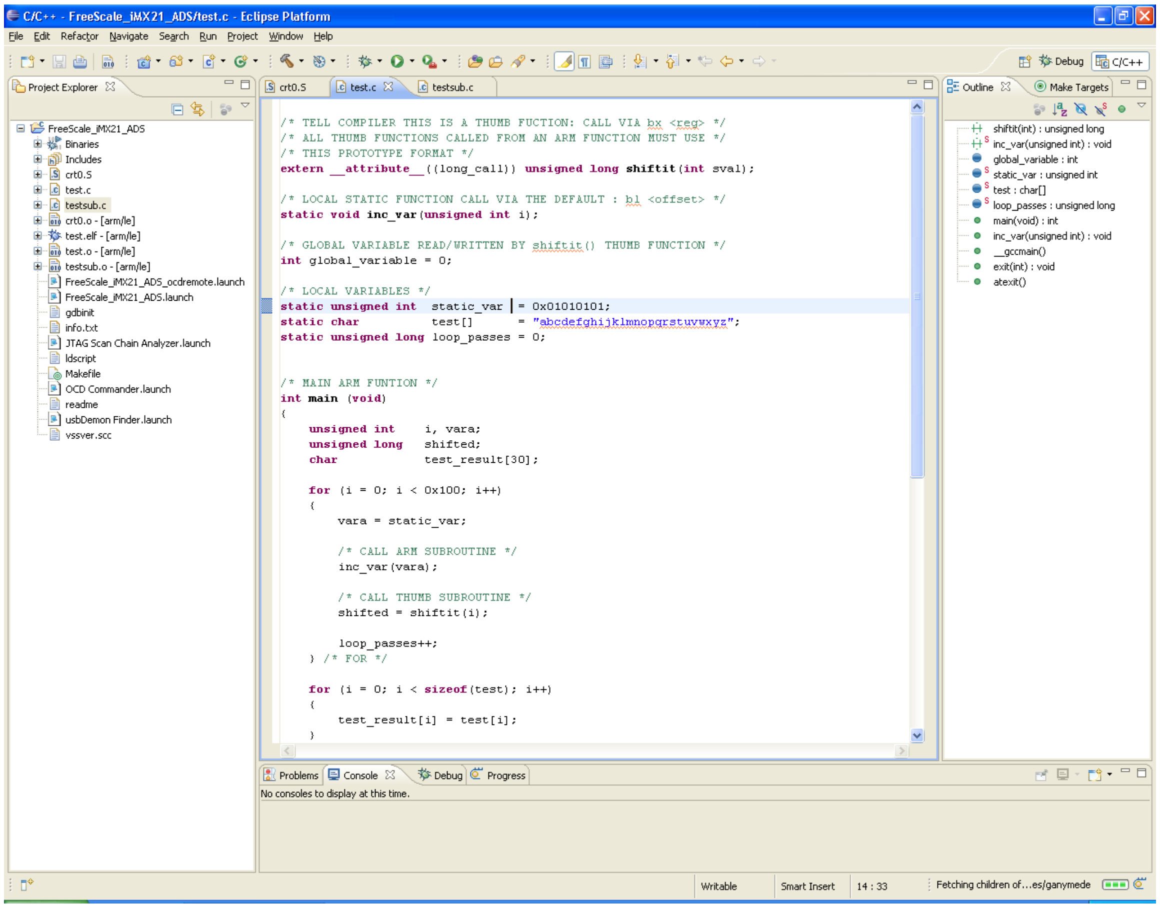
```
{
    unsigned int loca

    static_var++;
    local = static_va
    return;
} /* INC_VAR */


__gccmain()
{
} /* GCCMAIN */


void exit(int exit_co
{
  while(1);
} /* EXIT */


atexit()
{
  while(1);
} /* ATEXIT */
```

```
shiftit(int) : unsigned long
inc_var(unsigned int) : void
global_variable : int
static_var : unsigned int
test : char[]
loop_passes : unsigned long
main(void) : int
inc_var(unsigned int) : void
__gccmain()
exit(int) : void
atexit()
```

FreeScale_iMX21_ADS [C/C++ Local Application] C:\Program Files\Macraigor Systems\EclipseDemos\ARM\FreeScale_iMX21_ADS\test.elf (9/18/08 1:45 PM)

```
monitor help
allrun
allstop
char/short/long <addr>
char/short/long <addr> = <val>
endian [<big|little>]
halt
reg <regname>
reg <regname> = <value>
```

This command stops the debugger and then cleans up the debug window by removing all the entries for the gdb debug launch. Killing the debugger will also cause ocdremote (but not ocdremoteServer) to exit automatically, but the entry will still be shown in the Debug pane. To remove it, right-click on it and select “Remove All Terminated” as shown in the following screen shot. You should then be back to a blank debug pane and you can change the perspective back to “C/C++” and continue to work with your code.



```c
/* TELL COMPILER THIS IS A THUMB FUCTION: CALL VIA bx <reg> */
/* ALL THUMB FUNCTIONS CALLED FROM AN ARM FUNCTION MUST USE */
/* THIS PROTOTYPE FORMAT */
extern __attribute__((long_call)) unsigned long shiftit(int sval);

/* LOCAL STATIC FUNCTION CALL VIA THE DEFAULT : bl <offset> */
static void inc_var(unsigned int i);

/* GLOBAL VARIABLE READ/WRITTEN BY shiftit() THUMB FUNCTION */
int global_variable = 0;

/* LOCAL VARIABLES */
static unsigned int  static_var  = 0x01010101;
static char          test[]      = "abcdefghijklmnopqrstuvwxyz";
static unsigned long loop_passes = 0;


/* MAIN ARM FUNTION */
int main (void)
{
    unsigned int    i, vara;
    unsigned long   shifted;
    char            test_result[30];

    for (i = 0; i < 0x100; i++)
    {
        vara = static_var;

        /* CALL ARM SUBROUTINE */
        inc_var(vara);

        /* CALL THUMB SUBROUTINE */
        shifted = shiftit(i);

        loop_passes++;
    } /* FOR */

    for (i = 0; i < sizeof(test); i++)
    {
        test_result[i] = test[i];
    }
```

No consoles to display at this time.

Appendix A. ocdremote / ocdremoteServer Options

Ocdremote and ocdremoteServer can be configured to run with any Macraigor hardware device and support all target processors that Macraigor Systems has JTAG/BDM support for. They can also be configured to support target boards with multiple processors on a single scan chain. The available options for use with these utilities are shown below:

Usage:
ocdremote -c <CPU,CPU,...> [-p <port number>] [-d <device>] [-a <device address>] [-s <speed>]

ocdremoteServer -c <CPU,CPU,...> [-p <port number>] [-d <device>] [-a <device address>] [-s <speed>]

The –c/--cpu CPU argument takes a list of one or more processor types and is the only required parameter (the others will default). The processors must be listed in the order in which they are arranged on the scan chain. Available target CPU options are:

ARM 7/9/11/Cortex Families:

ARM7EJ-S | ARM7TDMI | ARM7TDMI-S | ARM9E-S | ARM9EJ-S | ARM9TDMI | ARM11 | ARM11MP | ARM720T | ARM920T | ARM922T | ARM926EJ-S | ARM940T | ARM946E-S | CortexA8 | CortexA9 | iMX21 | iMX23,iMX25x,,iMX27,iMX28,iMX31,iMX35,iMX50,iMX51,iMX53 | LH7A40X | LH7952X | NET+15 | NET+20M | NET+40 | NET+50 | NS7520 | NS9360 | NS9750 | NS9775 | NSARM9 | OMAP310 | OMAP35x | OMAP35xTC | OMAP710 | OMAP 1510 | OMAP5910 |SharpARM7 | SharpARM9 | TI925T |

Cavium 64 bit Family:

301x | CN300x | CN3020 | CN31xx | CN36xx | CN38xx | CN502x | CN52xx CN56xx | CN58xx | CN63xx | CN66xx | CN68xx | N3 | O1P | O2P | ONP | O52

Freescale Coldfire Family

MFC520x | MFC521x | MFC522x | MFC523x | MFC524x | MFC525x | MFC525x | MFC527x | MFC540x | MFC547x

ARM Cortex M3 Family

CortexM3 | CortexM3_2wire

MIPS 32-bit Family:

Alchemy_AU1x00 | BCM7115 | MIPS32_4Kc | MIPS32_4Ke | Vitesse_V3000 |

MIPS 64-bit Family:

MIPS64 | TX49 |

PowerPC Family:

AMCC440EP | AMCC440GP | AMCC440GX | AMCC460GT | MPC55X | MPC56X | MPC603 | MPC740 | MPC745 | MPC750 | MPC755 | MPC8XX | MPC5200B | MPC5554 | MPC8240 | MPC8245 | MPC8247 | MPC8248 | MPC825X | MPC826X | MPC8270 | MPC8271 | MPC8272 | MPC8275 | MPC8280 | MPC85X0 | PPC403 | PPC405 | PPC603 | PPC740 | PPC750 | PPC750FX | PPC750GX | APM821x1

DSP563XX Family:

dsp563xx |

AMD x86 Family

Duron | GeodeNX | SC520

Intel Atom x86 Family

Atom32| gmchTC | AtomSOC

Intel Xscale Family:

80200 | 80219 | 80321 | 81341 | 81342 | IOP303 | IOP315 | IOP321 | IOP331 | IOP332 | IOP333 | IXC1100 | IXP42x | IXP46x | IXP23XX | IXP2400 | IXP2800 | IXP2850 | PXA210 | PXA25x | PXA26x | PXA27x | XSCALE-5IR | XSCALE-5IRSLAVE | XSCALE-7IR | XSCALE-CORE3 |

Unknown or Non-Processor Devices

U:<ir bits>:<bypass bits>

The –p/--port argument is used to specify the TCP/IP port number that will be opened for communication with a remote debugger. This argument defaults to port 8888

The –d/--device argument specifies which Macraigor interface device is being used. This argument defaults to RAVEN. Available options are:

RAVEN | USB | MPDEMON_ETHERNET | WIGGLER | MPDEMON_PARALLEL | MPDEMON_SERIAL

The –a/--address argument indicates the address of the Macraigor Hardware device. This means different things for different devices. For Parallel devices (Raven, Wiggler, or mpDemon in Parallel mode), it is the LPT port number that the device is attached to, allowed values are 1 – 4, and it will default to LPT1. For the mpDemon in Serial mode, it is the COM port number that it is attached to, allowed values are 1 – 8, and it will default to COM1. For USB devices (usb2Demon, usb2Sprite, usbWiggler) it is the assigned address for the device, allowed values are 0 – 15, and it will default to 0. If a single USB device is attached to the host computer, it will always be at address 0. If multiple Macraigor USB devices are attached, the assigned address for each device can be found by running the "usbDemon Finder" utility application. For the mpDemon in Ethernet mode, it is the TCP/IP address of the mpDemon, it will not default (i.e. it must be specified) and must be in the format xxx.xxx.xxx.xxx. Examples of valid configurations are given at the end of this section.

The –s/--speed argument specifies the JTAG/BDM clock rate to use. This argument is used as a divisor on the raw clock rate of the Macraigor device, so lower numbers are faster. Allowed values are 1 – 8. This argument defaults to 1.

The –b/--baud argument is used only for the mpDemon in serial mode and specifies the baud rate for the serial connection. Allowed values are 1200 – 115200 baud and this will default to 115200.

Examples:

Wiggler on LPT2 at JTAG speed 1:

```
ocdremote       -c AMCC440EP -d WIGGLER -a 2 -s 1
ocdremoteServer -c AMCC440EP -d WIGGLER -a 2 -s 1
```

Raven on LPT1 at JTAG speed 1 :

```
ocdremote       -c AMCC440EP -d RAVEN -a 1 -s 1
ocdremoteServer -c AMCC440EP -d RAVEN -a 1 -s 1
```

mpDemon : ethernet - 192.168.1.30 at JTAG speed 2 :

```
ocdremote -c AMCC440EP -d MPDEMON_ETHERNET -a 192.168.1.30
-s 2
ocdremoteServer -c AMCC440EP -d MPDEMON_ETHERNET -a
192.168.1.30 -s 2
```

mpDemon: parallel on LPT1 at JTAG speed 3:

```
ocdremote       -c AMCC440EP -d MPDEMON_PARALLEL -a 1 -s 3
ocdremoteServer -c AMCC440EP -d MPDEMON_PARALLEL -a 1 -s 3
```

usb2Demon as USB device 0 at JTAG speed 1:

```
ocdremote       -c AMCC440EP -d USB -a 0 -s 1
ocdremoteServer -c AMCC440EP -d USB -a 0 -s 1
```

UsbSprite as USB device 2 at JTAG speed 3:

```
        ocdremote       -c AMCC440EP -d USB -a 2 -s 3
        ocdremoteServer -c AMCC440EP -d USB -a 2 -s 3
```

UsbWiggler as USB device 0 at JTAG speed 1:

```
        ocdremote       -c AMCC440EP -d USB -a 0 -s 1
        ocdremoteServer -c AMCC440EP -d USB -a 0 -s 1
```

Appendix B. Modifying Demos

If you will be working with either a custom board or an eval board that we don't have a pre-configured Eclipse project for, you will need to create an Eclipse project for your board and configure the environment to work with your hardware. This section will discuss the general tasks that will typically need to be done in order to get the Eclipse environment, gdb, and ocdremote / ocdremoteServer set up to work with your target hardware. This section might also be useful if/when you need to expand one of the demo projects in order to start building an actual, custom application.

The easiest way to begin working with a new or custom target board is to start with one of our provided demo projects that is as close as possible to the target hardware that you will be using. This will usually involve selecting a project that uses either the same target CPU type or at least one from the same family of CPUs. Browse through the existing demo projects, select one that uses a CPU type that is as close as possible to the one you have and import that project. You can right-click on the project name in Eclipse and rename the project to something that makes sense for your board. Note that you will have to also change the project name reference in the External Tools launch configuration and the Debug launch configuration. Just open these, change the names of the configurations and set the project name references to the new project name.

The first things that may need to be changed in order to work with a target board that we don't yet support are the arguments to ocdremote / ocdremoteServer. If your target processor is not exactly the same as is used in the existing demo, refer to Appendix A and find the CPU argument that matches your processor, then open the External Tools interface (discussed in section 3-3) and change the command-line arguments that are passed to OCDRemote. If you aren't sure of which processor type to select, you can run OCDCommander (discussed in Appendix D) and try various options until you find ones that work.

Once you think you have the arguments to ocdremote / ocdremoteServer correctly set up, try launching it from Eclipse and check that it starts without any errors. If you have problems at this stage, refer to Appendix D for some tips on troubleshooting connection issues.

In order to actually run code on a target CPU, your build must correctly specify the address of RAM on the target processor when the code is linked and the gdbinit file must be set up to configure your board so that, when gdb starts, the board is properly configured to run code and has RAM available at the address to which the code will be loaded. We suggest that you first get your board working with the simple test program that we provide in our demos. Once this works, you can begin adding your own code modules and developing your own applications. Each of the required steps is discussed below:

Setting the start address of the program:

The build process must be made aware of the address on the target to which the code will be loaded in order to correctly link the source modules. Our demos use the gcc C/C++ compiler and other gnu utilities to assemble, compile and link the source code. The standard way of specifying the start address of the code when using gcc is to provide a linker script that is passed to the build process by the makefile. You can look at what our make file does by editing it within Eclipse. You should see an entry for "Makefile" in the Eclipse project pane. You can double-click on the file name to edit it and read through the build steps. In our demos, the linker script is named "ldscript". You should see this file listed in the Eclipse project pane. You can edit it by double-clicking on it. Near the top of this file, you should see a line that looks something like this:

```
        . = <address>
```

This line sets the starting address of the code and must correspond to a RAM location on the target hardware where there will be enough room to hold the downloaded code. Edit this line and set the address to a location where your board will have RAM available and then rebuild the code.

Configuring the target hardware:

Now that the build process knows where to place the code, the target hardware must be configured so that code can be run and so that there is RAM available at the address specified in the ldscript file. In our demos, board configuration is handled by a gdb script file called "gdbinit". When Eclipse starts gdb, this file is passed to the debugger to be run. The gdbinit file must contain whatever steps are necessary to set up RAM and prepare the target CPU for running code.

In general, our demos use the simplest possible method of setting up RAM and configuring the board. On processors that have on-chip RAM, we use that. If a particular eval board has boot code already in Flash, we often just reset the processor and let the boot code run for a few seconds to get the board configured. For a custom board design, or an application which will be large and needs to run from the primary DRAM (as opposed to a small area of on-chip RAM), these methods will probably be insufficient. Typically, a significant amount of setup may need to be done on a board, after a hard reset, in order to get a RAM controller configured and to set up the processor environment so that code can be run. An example of a more involved gdbinit file can be seen in the Freescale_MPC8560_ADS demo project. You can import this project or just open the gdbinit file in this directory to see the steps that are taken. This example shows that many of the typical steps that need to be performed for board setup are best done using monitor commands to directly read/write registers, memory locations and/or memory-mapped registers. You can find a reference for available monitor commands in Appendix C.

The process of developing the steps necessary to configure a board when a debugger starts is somewhat cumbersome from within Eclipse. We strongly suggest that you use our OCDCommander debugger to develop the configuration process and, once it is working, translate the necessary commands into the syntax required by gdb. You can read more about OCDCommander in Appendix D. It is a very simple debugger with simple syntax and can run “macro” files that are just like startup scripts. These macros are just text files that contain a single command per line. A few example startup macros are included with the OCDCommander installation and can usually be found in \Program Files\Macraigor Systems\OCD Commander. You should be able to pick up the required command syntax from these files. Commander gives you a simple, quick way to test and modify your startup commands until you get everything working. Commander provides a built-in memory test command that can be used to test your memory configuration. Try typing “memtest 2 1024 0x0f000000 4”. This will run a memory test (writing a known pattern to memory and reading it back) using two iterations with a block size of 1k Bytes. The address of the test will begin at 0x0f000000 and each write/read will be done as a 4-byte quantity (i.e. a 32-bit word). The block size can be as large as 16K and allowable size parameters (the last argument) are 1,2,4,8 or 0, where 0 automatically selects the largest transfer size supported by the processor.

Once you have a working setup macro using OCDCommander, it is a simple task to translate the commands into the syntax required by gdb. Open and look at the gdbinit file in the Freescale_MPC8560_ADS project directory and you should be able to see the required syntax.

In general, you won’t want to change the configuration settings at the top of gdbinit or the macros that we define. Add your customized setup steps to the gdbinit file after these initial sections. Also, you will probably want to keep the last few commands in the gdbinit file that load the program, load the symbol table, set a breakpoint at main() and then run the processor.

Appendix C. Useful GDB and Monitor Commands

Some useful text mode commands to use on the GDB command line or in the Eclipse console:

s	Step, single step a C source code line.
si	Step Instruction, single step a machine code instruction.
c	Continue, run the processor after a step or breakpoint.
b	Breakpoint, set a breakpoint at specified location.
hbreak	Hardware Breakpoint - set a hardware breakpoint at specified location
d	Delete all breakpoints
bt	Print backtrace of all stack frames
up	Select and print stack frame that called this one
^C	Control C, stop execution from the keyboard.
l	List, show the source code being executed.
x	Examine, show the contents of memory.
i r	Info registers, show the contents of all registers.
set	Set, change the contents of ram or a register.
print	display the value of a register or variable
monitor	Send the following commands to the debug monitor. See below for monitor commands that the Macraigor ocdremote and ocdremoteServer utilities understand.

Monitor commands implement various functions that are not available using GDB directly and vary from CPU type to CPU type. You can get the full list of "monitor" commands by issuing the command "monitor help" in the console window after GDB has attached to your ocdremote/ocdremoteServer session. The following monitor commands are common to all CPU types and can be executed from the gdb command line, from the Eclipse console window after attaching to OCDRemote or from with a gdb command file, such as gdbinit.

Common ocdremote/ocdremoteServer monitor commands:

monitor help – Show all available monitor commands for the current CPU
monitor allrun – Synchronized run command for all processors
monitor allstop – Synchronized stop command for all processors
monitor char/short/long <addr> - Read a char/short/long value from <addr>
monitor char/short/long <addr> = <val> - Write <val> to <addr>
monitor endian [<big|little>] – Set endian mode of OCDRemote
monitor halt – Stop the CPU
monitor reg <regname> - Read the value of the <regname> register
monitor reg <regname> = <value> - Write <value> to <regname> register
monitor help regname - Display all available register names

monitor reset – Reset the target CPU and stop at the reset vector
monitor resetrun – Reset the target CPU and run from the reset vector
monitor runfrom <addr> - Start the CPU running at <addr>
monitor set memspace <virtual|physical|#> - Set the memory space for CPU
monitor set cpu <cpu number> - Set which CPU on the scan chain to send commands to.
monitor set/clear hbreak [<addr>] – Set or clear a hardware breakpoint at <addr>
monitor set regbufaddr <addr> - Set location of the register buffer to <addr>
monitor sleep <seconds> - Do nothing for <seconds> seconds
monitor softbkpts <on/off> - off = use only hardware breakpoints (useful when debugging flash) during running and stepping.
on = use software breakpoints by default unless user calls for a hardware breakpoint to be used (via gdb hbreak command)
monitor status – Report the status of the target CPU (running or halted)
monitor sync cpus <on/off> - Turn syncing on or off. Syncing on will attempt to run/stop all processors simultaneously.

Appendix D. Creating a new Eclipse-Juno Project from Scratch

1) Create a new directory; give this directory the same name you are to give your new Eclipse Juno project. (NewProject in this example). Copy your source, makefile, ldscript, and gdbinit files into a new directory.
2) In the …/Macraigor Systems/EclipseDemos/<CPU Type>/<Eval Board> directories find an example project that uses your CPU type. Copy the sample project's "<Project Name>.launch" file into your new directory. Rename it to same name as your directory (in this example …/NewProject/NewProject.launch)
3) Start up Eclipse Juno
4) Select the File->New->C Project menu option.
5) Set Project Name and Directory : In the C Project dialog that appears: enter you project's name, (in this example: NewProject) uncheck the "use default location" checkbox and use the browse button to select the directory location you saved your source, makefile, … files in. Then click the FINISH button. If your makefile is correct, Eclipse should compile your files and show them in the Project Explorer window and the results of your build in the Console window. If the build fails, fix your makefile until it builds successfully before continuing on to step 6. (Highlighting your project name and then clicking the "Build All" (piece of paper with 010 in the right side of the tool bar) icon starts a build).
6) Select a Binary Parser: Highlight your project name. Select the Project -> Properties menu choice. Expand the "C/C++ Build" list item in the Properties for <your project name> dialog that appears. Click on "Settings" to select the settings dialog. In the "Binary Parses" tab list check the "GNU ELF Parser" check box and then click the OK button
7) Set up ocdremote or ocdremoteServer As An External Tool :
   - Click on the arrow next to the External Tools icon (green arrow with suitcase) and select "External Tools Configurations…."
   - In the New Configuration Dialog that appears highlight "Program" and click the New Launch Configuration icon (piece of paper with + sign). A New_configuration dialog will appear. In this dialog's Main tab:
      a. Rename New_configuration to <ProjectName>_ocdremote[Server] (in this example NewProject_ocdremote[Server].launch)
      b. Location: Use the Browse File System button to select ocdremote[Server].exe (windows) or ocdremote[Server] (Linux). In the standard windows/cygwin environment this would be c:\cygwin\usr\local\bin\ocdremote[Server].exe.
      c. Working Directory: Use the Browse File System button to select the path to the directory that contains ocdremote.exe/ocdremote or ocdremoteServer.exe/ocdremoteServer. In the standard windows/cygwin environment this would be c:\cygwin\usr\local\bin.
      d. Arguments: Enter the parameters needed by ocdremote[Server] to connect to your target CPU. Typically this is "-c <cpu type> -d USB –s <speed>". See the example project readme files for more

information. Select the readme from an example project that is most similar to your target system.

- Click the New_configuration's Common tab:
    a. Save As: change the radio button from Local File to Shared file. Use the Browse button to select your project name as the base directory to save <ProjectName>_ocdremote[Server]   in.
    b. Display in Favorites Menu: check the External Tools checkbox.
- Click the APPLY button. This saves your changes.
- With your target powered up and connected to your host via a Macraigor JTAG device, click the RUN button. This should start ocdremote or ocdremoteServer and connect to your target. If so the console will display :
    1. ocdremote[Server] version>: USB via USB 0 at speed : <speed>

    JTAG SDO <-| CPU(1) <cpu type> : GDB port 8888 |<- JTAG SDI

  If not you can use the other external tools we provide: the JTAG ScanChainAnalyzer, UsbDemonFinder, and OcdCommander to debug your host to target interface.

8) Set up GDB Debug Configuration:

- In the C/C++ perspective: Project Explorer view, double click on the <ProjectName>.launch file to bring up the text editor. In the text editor modify the following lines :
    - CHANGE :
      <stringAttribute key="org.eclipse.cdt.launch.PROJECT_ATTR" value="**OldProjectName**"/>
      TO:
      <stringAttribute key="org.eclipse.cdt.launch.PROJECT_ATTR" value="**ProjectName**"/>
    - CHANGE :
      <stringAttribute key="org.eclipse.cdt.launch.WORKING_DIRECTORY" value="${workspace_loc:**OldProjectName** }"/>
      TO: <stringAttribute key="org.eclipse.cdt.launch.WORKING_DIRECTORY" value="${workspace_loc:**ProjectName** }"/>
    - CHANGE:
      <listEntry value="/**OldProjectName**"/>
      TO:
      <listEntry value="/**ProjectName**"/
      In our example these lines would now read :
      … PROJECT_ATTR" value="**NewProject**"/>
      … WORKING_DIRECTORY" value="${workspace_loc:**NewProject** }"/> … <listEntry value="/**NewProject**"/
    - Save these changes.

    - In the Project Explorer, highlight the project name, left click and select "Refresh" to update the project with these changes to the gdb debug configuration's launch file
- Click on the arrow next to the Debug (bug) icon in the middle of the toolbar and select Debug Configurations …
- In the Debug Configurations Dialog, click the plus sign next to "C/C++ Local Application" there should be an entry under C/C++ Local Application for <ProjectName>. If not make sure your modifications to your <ProjectName>.launch file are correct. Click on <ProjectName> to bring up its Debug Configuration parameters.
- Review/verify the parameters in the Debug Configuration Dialog tabs:
    a) Main tab : Project : <ProjectName>
    b) Main tab: C/C++ Application: use the Search Project button to select your load image file (in our examples this is typically test.elf).
    c) Arguments tab: Program Arguments: should be blank.
    d) Arguments tab: Use default : uncheck this checkbox
    e) Arguments tab. Set Working Directory : ${workspace_loc:<ProjectName>}
    f) Environment tab: Environment variables to set : should be blank
    g) Debugger tab: : Debugger : gdbserver debugger
    h) Debugger tab: Stop Startup At : (leave checked) enter function name or label that gdb will halt on at startup, typically "main"
    i) Debugger tab: Debugger Options : Main tab : Gdb Debugger : Use Browse button to select your debugger either "arm-elf-gdb", m68k-elf-gdb","mips-elf-gdb", x86-elf-gdb" or "powerpc-elf-gdb" in either c:\cygwin\usr\local\bin (WINDOWS) or /usr/local/bin (LINUX)
    j) Debugger tab : Debugger Options : Main tab : : Gdb Command File : .\<name of your gdbinit file> typically .\gdbinit
    k) Debugger tab : Debugger Options: Main tab : Protocol : mi
    l) Debugger tab : Debugger Options : Shared Libraries : leave blank.
    m) Debugger Tab : Debugger Options : Connection Tab : Type: TCP, Host Name or IP Address : <ocdremote ethernet address>, Port Number : <ocdremote port> (see page 18)
    n) Source tab : (use default values)
    o) Common tab : Save As : "Shared file"(checked) : \<ProjectName>
    p) Common tab : Display in Favorites menu : Debug (checked)
    q) Common tab : "Default Inherited", "Allocate Console", and "Launch in background" (all checked)
- Click the APPLY button. This saves your changes

9) Review/update your gdbinit file. When you start a debug session via the GDB debug configuration you set up in step 8, DSDP does the following:
    a. Runs the gdbinit file specified in step 8j. This file does three things:
        i. Sets up gdb for the target type it will be debugging (including creating any macros needed).
        ii. Connects to the target (via target remote <addr>:<port>) and does any board setup (using monitor char/short/long <address> =

<value>, monitor reset, and/or monitor reg <regname> = <value> commands) needed before the load image can be loaded into target RAM. Use our EclipseDemos/<CPU TYPE>/<EVAL BOARD> gdbinit files as a starting point for your gdbinit setup.

   iii. Writes the contents of the load image file (in our examples: test.elf) into target RAM, and loads the load image's symbols into gdb.

b. Disconnects from the target
c. Reconnects, sets a breakpoint at the symbol address specified in step 8-h and runs the load image written to the target by the gdbinit file until the breakpoint is reached.

This means that your gdbinit file must look like the ones in our examples to work correctly with Eclipse Juno and DSDP. Verify that your gdbinit file contains the line:

```
target remote <ocdremote address>:<ocdremote port number>
```

(Typically target remote localhost:8888 (Windows) or target remote 127.0.0.1:8888 (Linux))

And that your the Debug Configuration Dialog : Debugger Tab : Debugger Options : Connection Tab parameters match those on the gdbinit's "target remote" line

10) Starting the Debug Session.

- With your target powered up and connected to your host via a Macraigor JTAG device start up ocdremote or ocdremoteServer as per step 7.
- Click on the arrow next to the Debug (bug) icon in the middle of the toolbar and select Debug Configurations …
- In the Debug Configurations Dialog, click the plus sign next to "C/C++ Local Application" there should be an entry under C/C++ Local Application for <ProjectName>. Click the DEBUG button to start gdb. If all is as it should be, DSDP will execute the operations talked about in step 9 and Eclipse will switch to the debug perspective, halted at main (or the label you set in the Debugger tab: Stop Start at).
- More information on trouble shooting and Eclipse/gdb can be found in "Appendix E. Troubleshooting" in this document and at http://www.gnu.org/software/gdb/gdb.html
  Eclipse Help -> Help Contents -> C/C++ Development User Guide

Appendix E. Troubleshooting

If you are having trouble getting your Eclipse environment to work with a Macraigor device, please refer to the text below to see the most likely causes of some of the more common issues. Many of these issues can be resolved, or at least better characterized, by using the additional Macraigor utility programs that are provided as part of the OCDRemote installation package. These programs and their general uses are described below. Each of these utilities can be launched from within Eclipse from the External Tools interface. To access this interface, click on the button that looks like this:

and then select the tool that you wish to run.

<u>OCDCommander:</u>

OCDCommander is a very simple, low-level debugger that was written by Macraigor and is used extensively internally for our own testing. In general, if you select the correct connection device at the correct address and the correct processor type from the Commander startup screen and everything is connected properly, you should be able to establish a connection to your processor and then check that you can do simple things like run and halt the processor, read registers, etc. Once you get a successful connection using OCDCommander, you can then use the same configuration settings within Eclipse when you start OCDRemote.

If you are sure that everything is connected correctly and you still can't get a connection using Commander, try slowing down the Speed setting (low speeds are faster, so set the number higher to go slower). A speed setting of 6 or 7 is usually slow enough to see if the clock rate might be your problem. If a slow speed works, you can experiment with increasing the speed until it doesn't work and then use the fastest setting that does work in Eclipse.

If you contact Macraigor with a connection problem, we will almost always ask you to try the connection with Commander and will often also ask for a log file. You can save yourself time by trying Commander first yourself. Checking the box on the opening dialog that says "Start with logging on" will cause a log file to be generated named "wigglers.log". You can email this file to us in order to assist in addressing your problem.

<u>usbDemon Finder:</u>

This utility is used for several things related to working with Macraigor USB interface devices. It will start and look for all Macraigor USB devices that are connected to the host machine and will then display the serial numbers of each device along with the assigned device number. This device number is the "address" that must be used when calling OCDRemote or when setting up OCDCommander. To distinguish between multiple USB devices, there is a button in the Finder that will cause the green LED on the device to flash for a visual indication of which device is currently selected. Even with a single connected USB device, the Finder can be helpful to ensure that the device is properly connected and functioning and that the Macraigor USB drivers are installed correctly. The usbDemon Finder is also used to program licenses into the USB device. If you are using a Macraigor application that requires licensing, such as the Flash Programmer, you will need to use the Finder to set the license code in the USB device.

JTAG Scan Chain Analyzer:

This utility analyzes the JTAG scan chain that the selected device is connected to and will attempt to identify any processors that it finds on the chain along with any other JTAG devices that are connected. Running this utility can be helpful to check that, at the most basic level, the JTAG connection is working properly. It can also be used to help identify the type of processor that you are working with and to detect if there are other, non-processor, devices on the scan chain. We have often found that some hardware designs will put devices such as FPGAs on the same scan chain with the processor. This type of configuration will cause the connection to the processor to fail initially. Our drivers and OCDRemote are capable of handling designs of this type (or designs with multiple CPUs), but we have to know what the configuration of the scan chain looks like. If you are having unexplained connection problems, try running this utility and make sure that the reported scan chain and the type of processor(s) are what you expect. If you find that there are other devices or CPUs on the scan chain, we can help you to configure OCDRemote so that it can properly communicate with your processor.

Note that this utility only works for JTAG scan chains. It cannot be used with CPUs that use a BDM type of interface such as the Freescale MPC8xx/5xx families of processors.

General tips for finding connection problems:

If you can't get ocdremote or ocdremoteServer to start up without errors, there are several things you can try to verify that the Macraigor hardware is working, that all of your configuration settings are correct, and that you are correctly connected to the target processor.

The first thing to check is that your Macraigor device is connected to the host, the proper drivers are installed and that the device is communicating properly. If you are using a USB interface device, run the usbDemon Finder as described above. If you have a Wiggler or a Raven, you can download a utility called the Raven

Tester from the Macraigor web site. This utility will exercise a Raven to be sure that it is working properly. For a Wiggler, it can usually determine if the device is connected, but cannot test it. If you are using an mpDemon (most often used in Ethernet mode), you can try pinging the address of the mpDemon from the host machine.

Assuming that your device is responding to these basic connection tests, the next level of checking is to test that the JTAG interface and scan chain are working properly. If you are working with a BDM device, skip to the next paragraph. With a JTAG-based debug interface, you can run the JTAG Scan Chain Analyzer. This utility, as described above, will analyze the scan chain and attempt to identify any processors that it finds. Make sure that the processor(s) reported are what you expect and that no other, unexpected devices are found on the chain. If the Scan Chain Analyzer reports an error, the most likely causes are either a bad connection to the target header (make sure that the ribbon cable is positioned correctly for Pin 1) or a problem with your board's JTAG interface design. We often see custom CPU boards that are designed in such a way that they won't work with a Macraigor interface device. If you are getting errors from the Scan Chain Analyzer on a custom-designed board, you can email the schematic for your JTAG interface to us and we can usually quickly spot any potential problems. See below for how to contact us by email.

If the Scan Chain Analyzer is reporting what you expect (or you are using a CPU with a BDM interface), the next step is try establishing a connection using OCDCommander. Start Commander, double-check all of the initial configuration settings and then try to connect. If Commander starts without reporting an error, try clicking on the "status" button. You may see either "running" or "In DEBUG". If the processor is reported as running, try clicking on the "reset" button. This should reset the processor and stop it at the reset vector. Once it stops, try clicking on the "cpu" button and/or the "regs" button to see if you can read registers. If this seems to work, type "test 2". This will run a write/read test on the registers. If any of the previous suggestions don't work in Commander, you can exit, find the wigglers.log file that should have been created (assuming that you checked "Start with logging on" on the startup dialog) and email it to us. This file can provide us with a lot of information about what might be happening.

If all of these things work in Commander, you should be able to use the same configuration parameters in Eclipse when you start OCDRemote. Refer back to section 3-3 and Appendix A if you don't remember how to configure OCDRemote's startup parameters.

If you still have problems after trying the above debugging steps, you can contact Macraigor for help. Our contact information is given below. We prefer that you contact us via email and <u>include as much information and detail as possible</u> in your initial email. This will help us to resolve your problem as quickly as possible. In your email, please include at least the following information:

- Type of target CPU
- Type of Macraigor device you are using
- Did you run the usbDemon Finder or Raven Tester? If so, what happened and what, if any, error messages did you get?
- Did you run the JTAG Scan Chain Analyzer? If so, what happened and what, if any, error messages did you get?
- Did you run OCDCommander? If so, what happened and what, if any, error messages did you get?
- If you were able to get connected with OCDCommander, but ran into problems trying to control the processor, what did you try, what happened, what error messages did you get? If this is the case, please also include or attach the wigglers.log file with your email.
- If OCDCommander works, but ocdremote /ocdremoteServer still won't start, what error message is it reporting?

Also include any other information that you think might be helpful. Is the problem consistent or intermittent? If it is intermittent, have you observed anything that might correspond to times that it works or doesn't work? The more information you can include, the more likely it is that we will be able to resolve the problem without having to contact you for further information.

Macraigor Systems Support Contacts:

Email support issues to: support@macraigor.com

We can be reached by phone during regular business hours M-F, 9 a.m. – 5 p.m. EST (GMT – 5:00) at (617) 739 – 8693.

Further information can also be found on our web site at www.macraigor.com.