



OVP Debugging Applications with Eclipse User Guide

Imperas Software Limited

Imperas Buildings, North Weston,
Thame, Oxfordshire, OX9 2HA, UK
docs@imperas.com



Author:	Imperas Software Limited
Version:	1.2
Filename:	OVPsim_Debugging_Applications_with_Eclipse_User_Guide.doc
Project:	OVP Debugging Applications with Eclipse User Guide
Last Saved:	Wednesday, 22 July 2015
Keywords:	

Copyright Notice

Copyright © 2015 Imperas Software Limited All rights reserved. This software and documentation contain information that is the property of Imperas Software Limited. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Imperas Software Limited, or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Imperas permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

IMPERAS SOFTWARE LIMITED, AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Table of Contents

1	Preface.....	4
1.1	Notation.....	4
1.2	Related OVP Documents	4
2	Introduction.....	5
2.1	Prerequisites	5
3	The Imperas Eclipse Helios Distribution	6
4	Imperas Eclipse Provided Debug Demos	7
4.1	Example Demonstration OVPSim_single_arm	7
4.1.1	Launching Eclipse.....	7
4.1.2	Starting Debugging	8
4.1.2.1	Start the simulated platform.....	8
4.1.2.2	Connect the Eclipse Debugger.....	9
4.1.3	Example Debug Session for Dhrystone Benchmark application.....	10
4.1.4	Re-building the Platform and Application.....	13
5	A New Debugging Example	15
5.1	Command Line arguments for debugging an ICM Platform	15
5.1.1	Specifying the debugger connection details	15
5.1.2	Nominating the debugged processor.....	15
5.2	Creating a Debuggable ICM Platform	15
5.2.1	Specifying the debugger connection details	16
5.2.2	Nominating the debugged processor.....	16
5.3	Preparing an Eclipse Project	17
5.3.1	Creating a New Project	17
5.3.2	Configuring Project Build Commands.....	18
5.3.2.1	Using MINGW ‘make’	18
5.3.2.2	Adding MinGW tools onto PATH.....	18
5.3.3	Deselect ‘Autobuild’ from Eclipse	20
5.4	Importing a Demo to the Project.....	21
5.5	Creating Launch Configurations.....	24
5.5.1	A Debug Configuration.....	24
5.5.2	Creating a Shortcut to Launch the OVPSim Platform.....	27
5.5.3	Testing the Debug Configuration.....	30
5.5.4	Switching Console Output	31
5.5.5	Terminating the Debug Session	33
5.5.6	Verbose GDB Console Output.....	34
5.6	An Example Debug Session.....	34

1 Preface

This document describes how to debug an application running on the OVP simulator using the Eclipse Helios 3.6.2 release Integrated Development Environment.

The example in this document demonstrates debugging of ARM and MIPS32 applications but the same approach is valid for applications running on processor models for other architectures supported by the OVP simulator.

1.1 *Notation*

Code

Code and command extracts

1.2 *Related OVP Documents*

- OVPsim Installation and Getting Started Guide
- OVPsim and CpuManager User Guide

2 Introduction

The *OVPsim and CpuManager User Guide* describes how platforms containing any number of processor models can be constructed. This document describes how to debug an application running on *one* processor in such a platform, while it is simulating using the OVPsim simulation environment. OVPsim supports single-processor debugging with Eclipse and the Gnu debugger (GDB) via the Remote Serial Protocol (RSP). Advanced multi-processor debug facilities are available in Imperas commercial products.

2.1 Prerequisites

Pre-built demos are provided, that will be described later, to allow the immediate debugging of an application using OVP and Eclipse Helios 3.6.2

To debug a simulated application with Eclipse you will need Eclipse with the C/C++ Development Tooling (CDT) plugins.

This walkthrough was produced with Eclipse Helios version 3.6.2.

It is recommended that Eclipse Helios version 3.6.2 has been installed using the Imperas installer, see section 3 The Imperas Eclipse Helios Distribution. Alternatively, Eclipse is available for free download from www.eclipse.org. The *Eclipse IDE for C/C++ developers* package should be used which includes the necessary plugins.

To build OVPsim platforms you require the OVP API files and libraries as well as a native compiler.

To build applications you need a cross-compiler and a version of the Gnu debugger for the chosen architecture.

The *OVPsim Installation and Getting Started Guide* provides a step-by-step guide to obtaining and installing the OVP files, a native compiler and cross compiler tools. It is strongly recommended that you follow that guide until you are able to build, compile and simulate a platform using OVPsim.

3 The Imperas Eclipse Helios Distribution

An installer *Imperas_Eclipse_Helios_3_6_2.<version>* is available from the OVP resources pages.

The Imperas Eclipse installation package provides an Eclipse installation that includes the required plugins for C/C++ application and OVPsim platform development and debugging.

The package installs, by default, into a directory *Imperas_Eclipse*.

An environment variable, *IMPERAS_ECLIPSE_HOME*, is used to point to the root of the installation. This is setup automatically on Windows and may be setup using the script, *setup.sh*, provided within the installation on Linux.

4 Imperas Eclipse Provided Debug Demos

A number of demonstrations are provided that include the required files to setup a defined project into a new Eclipse workspace.

The demos provided will include a script to start an Eclipse development and debug session. These scripts are identified with the starting prefix **DEBUG_ECLIPSE_**

The script expects that the Imperas Eclipse package has been installed and is located by the environment variable IMPERAS_ECLIPSE_HOME. If it is not available the script will fail.

The GDB is expected to be found in one of the Cross Compiler libraries. If it is not available the script will fail.

4.1 Example Demonstration OVPsim_single_arm

4.1.1 Launching Eclipse

The first time we run the script `DEBUG_ECLIPSE_dhrystone.bat` we will create a new Eclipse workspace and in it a new project `OVPsim_single_arm` before invoking Eclipse.

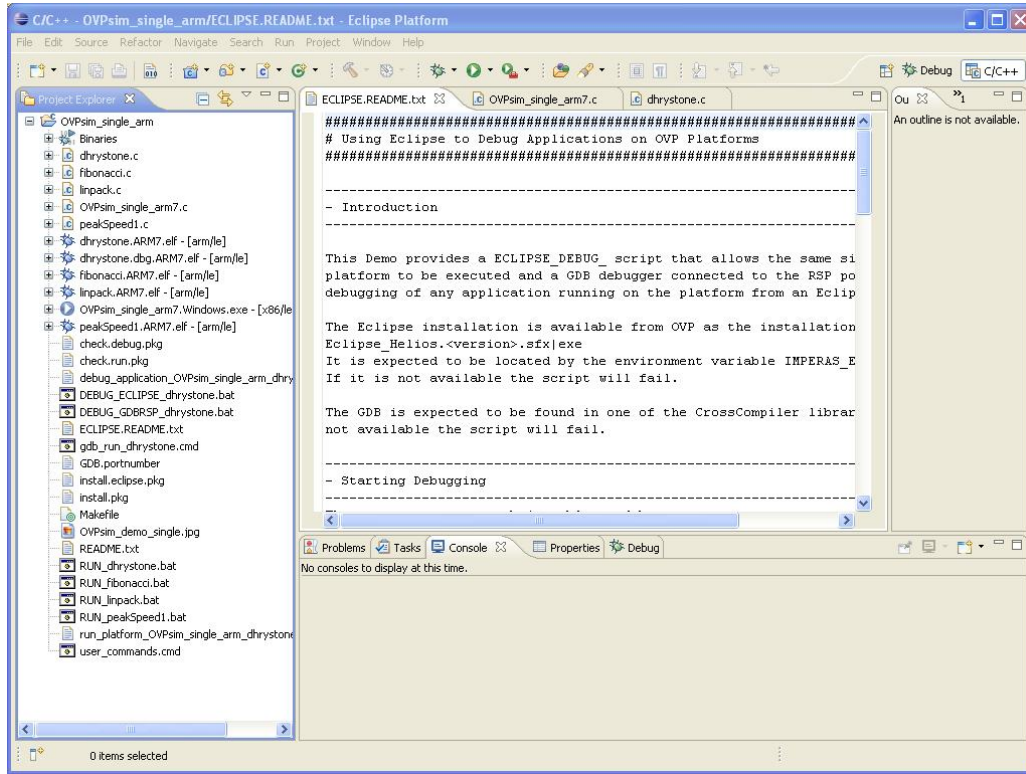
You will see that a project is Imported into a default workspace location for this demonstration before Eclipse is invoked using this workspace.



All subsequent runs of the script will only invoke Eclipse allowing the same workspace to remain.

When Eclipse has started a project is available with three files open

1. A README providing the same information provided in this section
2. The OVPSim platform definition
3. The Dhrystone benchmark application that will be used for this debugging demonstration.



4.1.2 Starting Debugging

There are two stages to being able to debug.

- 1) Starting the simulated platform.
- 2) Connecting the Eclipse Debugger

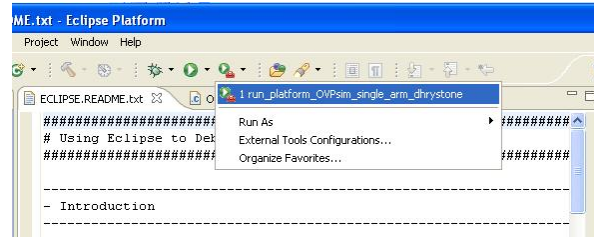
These are provided in the form of Eclipse 'launch's

4.1.2.1 Start the simulated platform

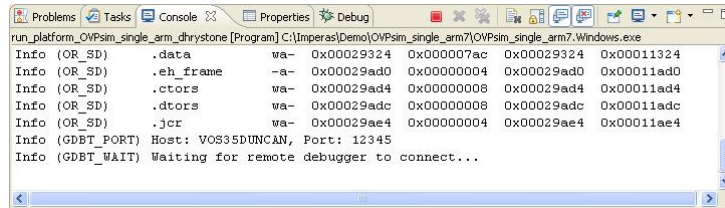
This is invoked as an 'External Tool' from the Menu "run->external tools" or the drop down menu button identified by the green run arrow with red toolbox icon

The 'launch' may already be visible in the quick launch section or will have to be selected from the "External Tools Configuration..." menu

The launch to invoke is "run_platform_OVPSim_single_arm_<application name>"



The platform will now wait for a connection to be made to the specified port

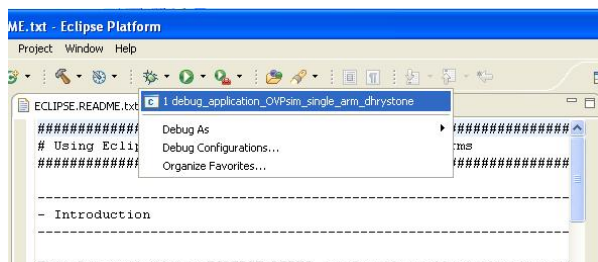


4.1.2.2 Connect the Eclipse Debugger

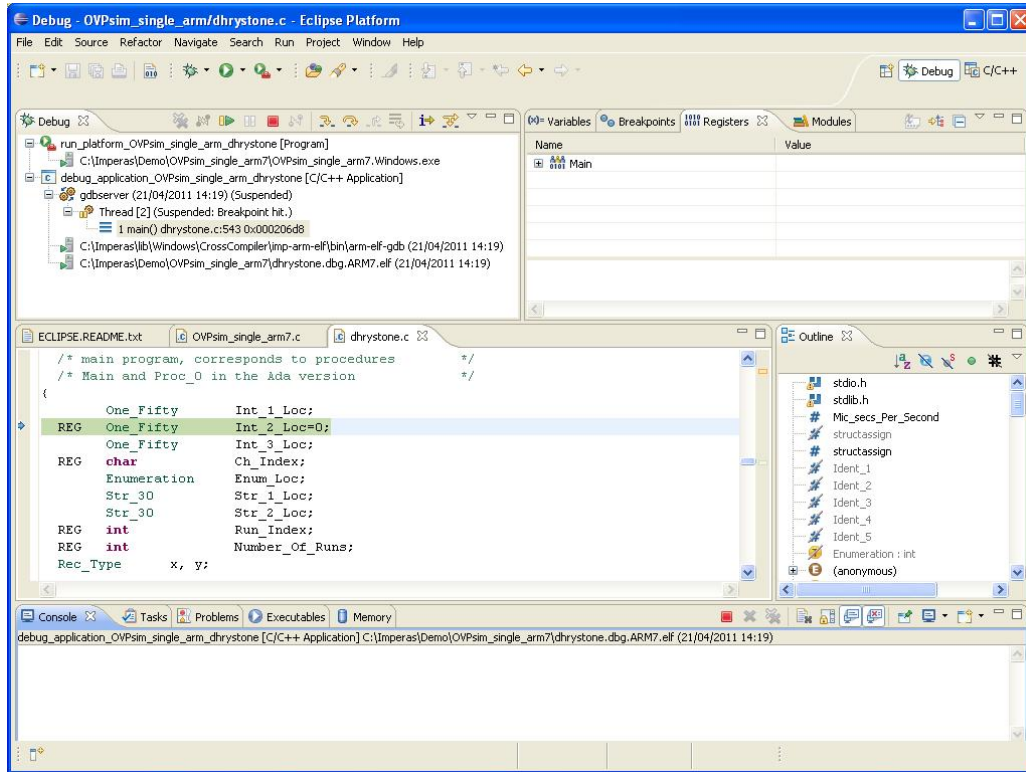
This is invoked as a 'Debug Configuration' from the Menu "run->Debug" or the drop down menu button identified by the same green run arrow icon

The 'launch' may already be visible in the quick launch section or will have to be selected from the "Debug Configuration..." menu

The launch to invoke is "debug_application_OVPsim_single_arm_<application name>



You should now be in the 'Debug' Perspective with control over application execution and visibility of all registers, variables etc.



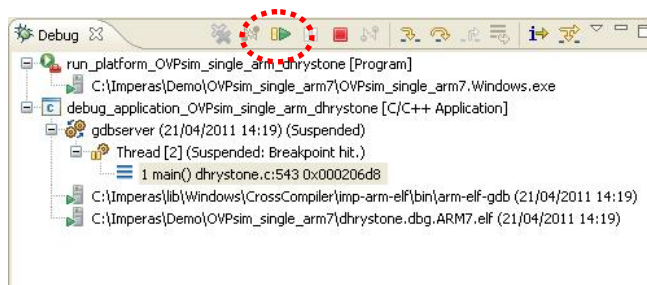
4.1.3 Example Debug Session for Dhrystone Benchmark application

You should be familiar with the general debug control 'buttons' within Eclipse in order to work through the following

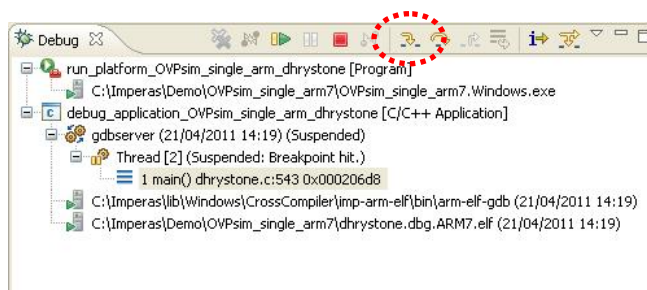
1. Set a breakpoint on the line containing the call to 'Proc_5();' by double clicking beside this line; this is found on line 608 (the line number may change slightly with versions of the application)



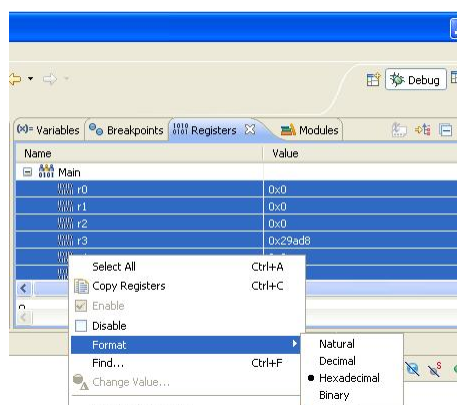
2. Continue the simulation using the green 'play' symbol



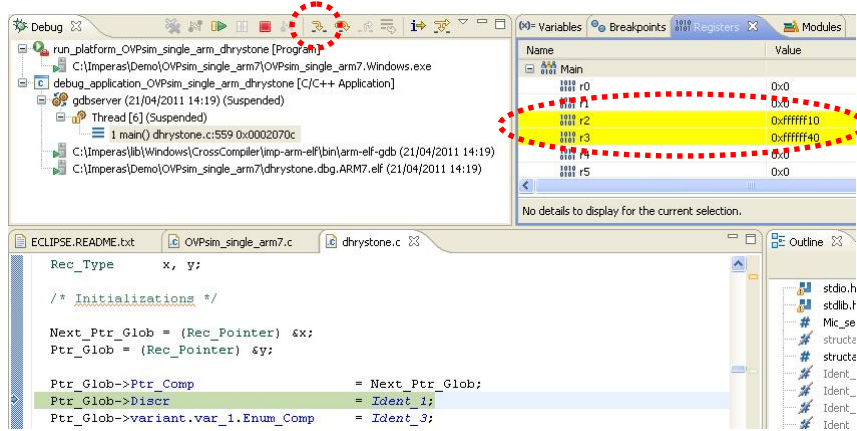
3. Step the application forward using the 'Step Into' button



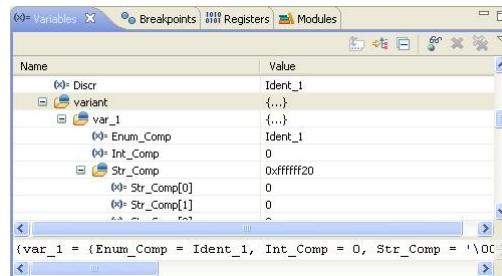
4. Open the register view; expand 'main', select registers and set the format to hexadecimal



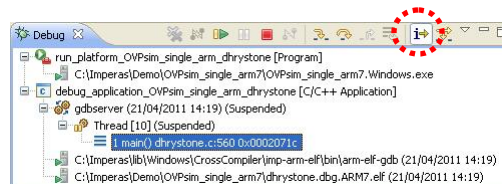
- Step the application forward using the 'Step Into' button. This will show you the colored highlighting of the registers that change



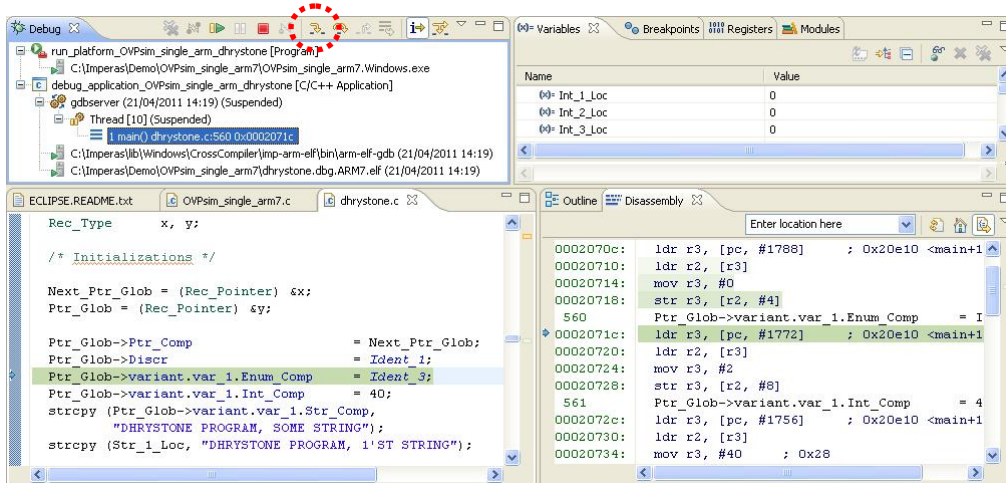
- Open the Variables view; showing the current local variables



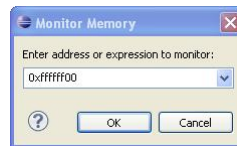
- Select 'Instruction Stepping Mode' using the button.



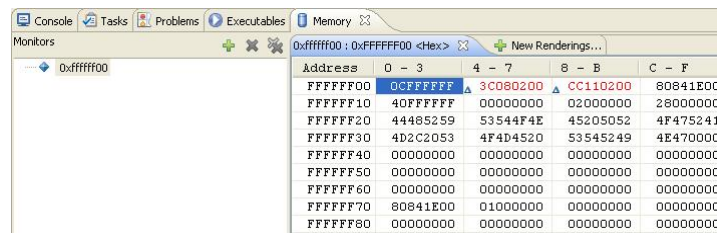
8. Step the application forward using the 'Step Into' button this will open the disassembly view and highlight the instructions as they are stepped.



9. In the 'Memory' window, select 'Add Memory Monitor' Use the value of the register 'SP' with bottom two bits as zeros to set the memory monitor base address so that it will show the program stack.



10. Move the application forward using the 'Step Return' button. The memory locations that are modified will be highlighted.



11. To finish the simulation you may either:
 - a. Use 'Resume' to continue to the end of the application
 - b. Use 'Terminate' After either of these the debug perspective can be cleaned up with the 'Remove All terminated Launches' button

4.1.4 Re-building the Platform and Application

In order to re-build the platform file and the application source code you will require the following packages installed:

1. MSYS and MinGW environments (for use under Windows)
2. Processor Cross Compiler toolchain
3. OVPsim

With these packages installed re-building can be carried out.

This is done from the C/C++ perspective using the build button (a hammer icon) or using the menu item Project->Build All

5 A New Debugging Example

The following section provides information for the configuration of Eclipse Helios to allow the debugging of an application running on an OVP platform.

5.1 Command Line arguments for debugging an ICM Platform

If the platform has the Imperas Command Line Parser (CLP) included it makes available command line options to enable and control debugging of an application on the processor in a platform without the need to make any changes.

5.1.1 Specifying the debugger connection details

The debug port is enabled by specifying the argument *--port <port number>* on the command line. A specific port number may be specified or by setting *port number* to 0 the next available port is opened.

5.1.2 Nominating the debugged processor

In an OVPSim simulation only a single processor may be connected to a GDB debugger¹. this requires that the processor is selected using the *--debugprocessor <processor name>*. In this case the processor name is the instance name in the platform, for example *platform/cpu0*

5.2 Creating a Debuggable ICM Platform

The following platform shows the modifications required to open a debug port for connection:

```
int main(int argc, char ** argv) {

    int portNum;

    if(argc != 2) {
        icmPrintf("usage: %s <port number>\n", argv[0]);
        return -1;
    }

    sscanf(argv[1], "%d", &portNum);

    // initialize CpuManager
    icmInitPlatform(ICM_VERSION, 0, "rsp", portNum, "platform");

    // create a processor
    icmProcessorP processor = icmNewProcessor(
        "CPU_0",           // processor name
        CPU_TYPE,          // CPU type
        0,                 // processor cpuId
        0,                 // processor model flags
        32,                // address bits
    );
}
```

¹ The Imperas Professional products allow the ability to attach a GDB debugger to any or all the processors defined in a platform. Imperas also provide alternative debugging solutions.


```
MODEL_FILE,          // model file
"modelAttrs",        // morpher attributes
ICM_ATTR_DEBUG,      // DEBUG THIS PROCESSOR
0,                  // user-defined attributes
SEMIHOST_FILE,       // semi-hosting file
"modelAttrs"         // semi-hosting attributes
);

// load the processor object file
icmLoadProcessorMemory(processor, "application.elf", False, False);

// run simulation
icmSimulatePlatform();

// terminate simulation
icmTerminate();

return 0;
}
```

For a full explanation of OVPsim platform construction please see the *OVPsim and CpuManager User Guide*. This section describes only those aspects of platform construction that relate to debugging.

5.2.1 Specifying the debugger connection details

The ICM kernel is initialized by calling `icmInitPlatform`:

```
void icmInitPlatform(ICM_VERSION, Uns32 simAttrs, const char *dbgProtocol, Uns32
dbgPort, char *platform)
```

The second and third arguments of `icmInitPlatform` are used for single processor debugging:

```
icmInitPlatform (ICM_VERSION, 0, "rsp", portNum, "platform");
```

Eclipse uses the GNU debugger for C/C++ debugging. GDB Remote Serial Protocol (RSP) debugging as supported by OVPsim uses standard operating system sockets on the host running OVPsim and GDB.

If a NULL value is given for `dbgProtocol`, debugging is disabled. To enable debugging, specify the protocol for the debug connection, currently only the 'rsp' protocol is supported.

The `dbgPort` argument specifies the socket port on which to accept a debugger connection. The special value of zero allows OVPsim to choose any free port on the host, otherwise the specified port number is used. For the example platform the port number is given by the first command line argument.

5.2.2 Nominating the debugged processor

A single instance of a processor is defined by calling `icmNewProcessor`:

```
icmProcessorP processor0 = icmNewProcessor(
```



```

"CPU_0",          // processor name
CPU_TYPE,        // CPU type
0,              // processor cpuId
0,              // processor model flags
32,             // address bits
MORPHER_FILE,    // model file
"modelAttrs",    // morpher attributes
ICM_ATTR_DEBUG,  // DEBUG THIS PROCESSOR
0,              // user-defined attributes
SEMIHOST_FILE,   // semi-hosting file
"modelAttrs"     // semi-hosting attributes
);

```

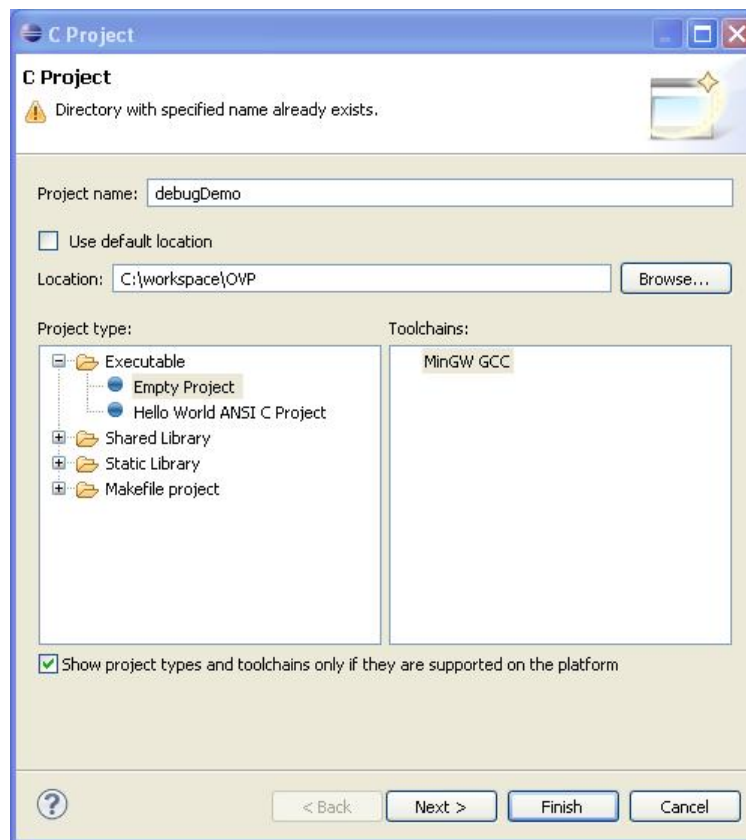
The processor to be debugged is indicated by specifying the ICM_ATTR_DEBUG processor attribute.

5.3 Preparing an Eclipse Project

This section shows all the steps needed to create an Eclipse project which allows the example platform and target application to be built and debugged in the Eclipse development environment.

5.3.1 Creating a New Project

Create a new C project by selecting “New” from the “File” menu, then selecting “C Project”. If this option is not present you may not have the C development plugins (CDT) for Eclipse – see section 2.1.



Select “Makefile project” as we will use an external Makefile within the project. Specify a project name and optionally a location. Note that the project location path *must not* include spaces as Eclipse passes the project directory to an underlying Gnu debugger in a way that does not handle spaces correctly.

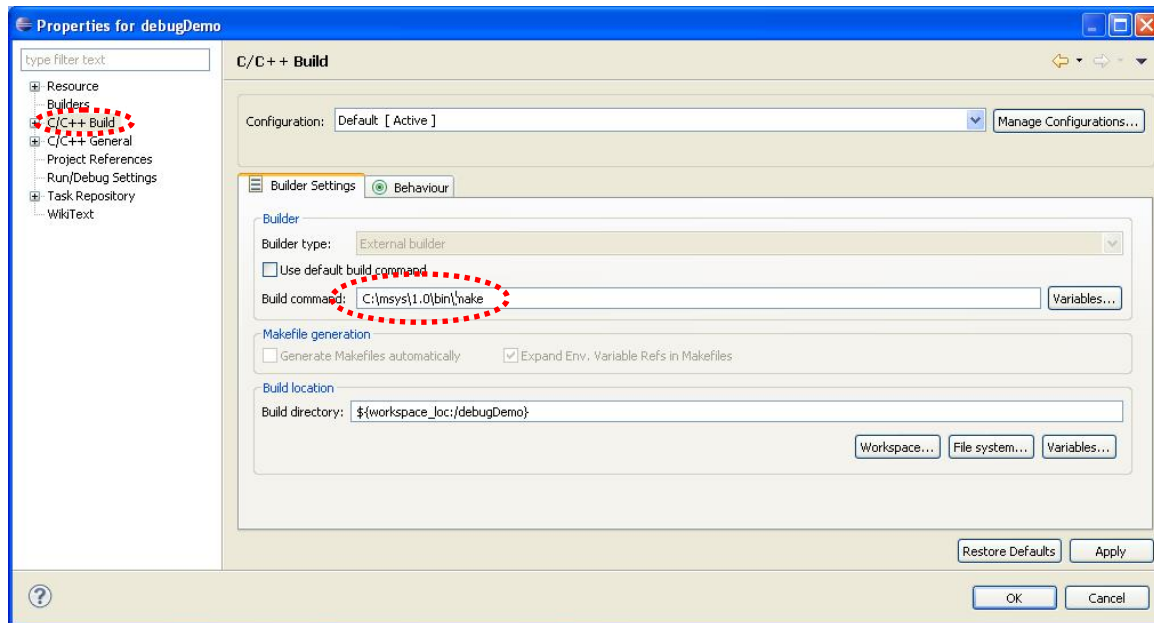
The default workspace and hence project location on Windows includes “Documents and Settings” so in this case you should uncheck the “Use default location” checkbox and specify a location which does not include spaces in the name. Click the “Finish” button to create the project.

The new project contains no Makefile or source files so Eclipse will show an error message if it attempts to build at this point.

5.3.2 Configuring Project Build Commands

5.3.2.1 Using MINGW ‘make’

Change the project build configuration so that Eclipse can find the MinGW “make” tool used to build the project. This step may not be necessary if you already have a version of GNU “make” on your system path. Select “Properties” from the “Project” menu and select the “C/C++ Build” pane.



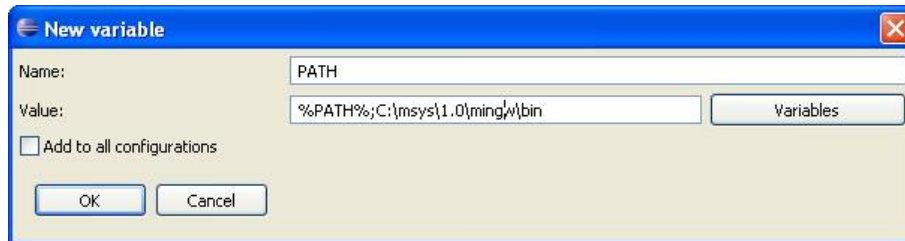
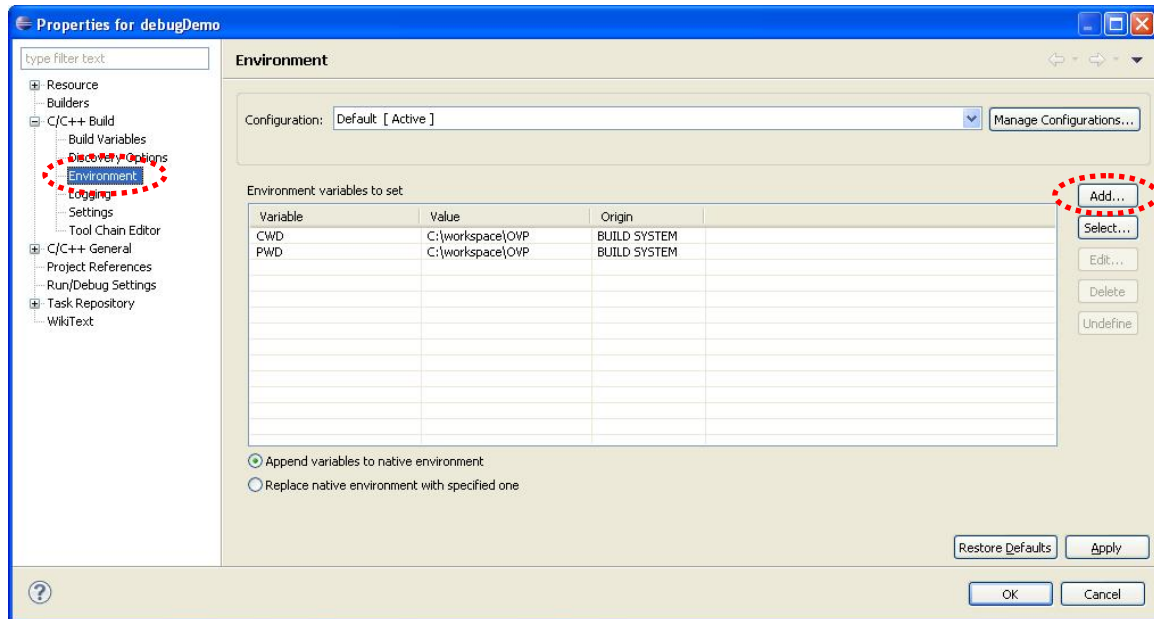
Uncheck the “Use default build command” checkbox and specify the full path of the MinGW/MSYS make executable on your system in the “Build command:” text box. In the example shown the default MSYS install location of “c:\msys\1.0\bin\make” is used.

5.3.2.2 Adding MinGW tools onto PATH

It is recommended that your MinGW binary path is appended to the system path within your Windows environment. In the example the MinGW install location of “c:\msys\1.0\mingw\bin” is used.

However, it may be added into your Eclipse environment by adding a variable “PATH” which includes the current value of PATH (given by %PATH%) and the MinGW path separated by a semi-colon.

Add the MinGW native compilation tools to the path by selecting the “Environment” tab of the same dialog and pressing the “Add...” button.

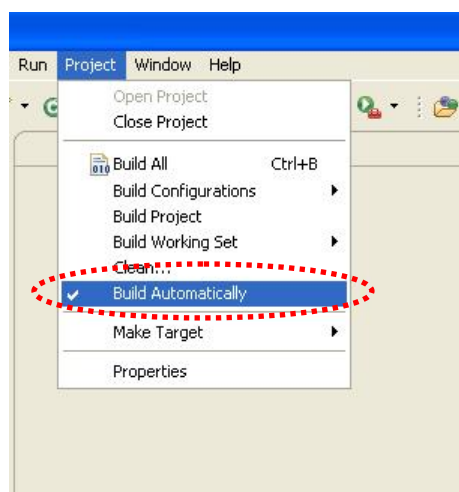


Close the properties dialog by pressing “OK”.

5.3.3 Deselect 'Autobuild' from Eclipse

By default, a new project in Eclipse will have autobuild selected. This can be useful, but when debugging cross compiled applications on an ICM platform this may cause the build process to be re-invoked between the launching of the simulation platform and the attaching of the debugger to allow application debug.

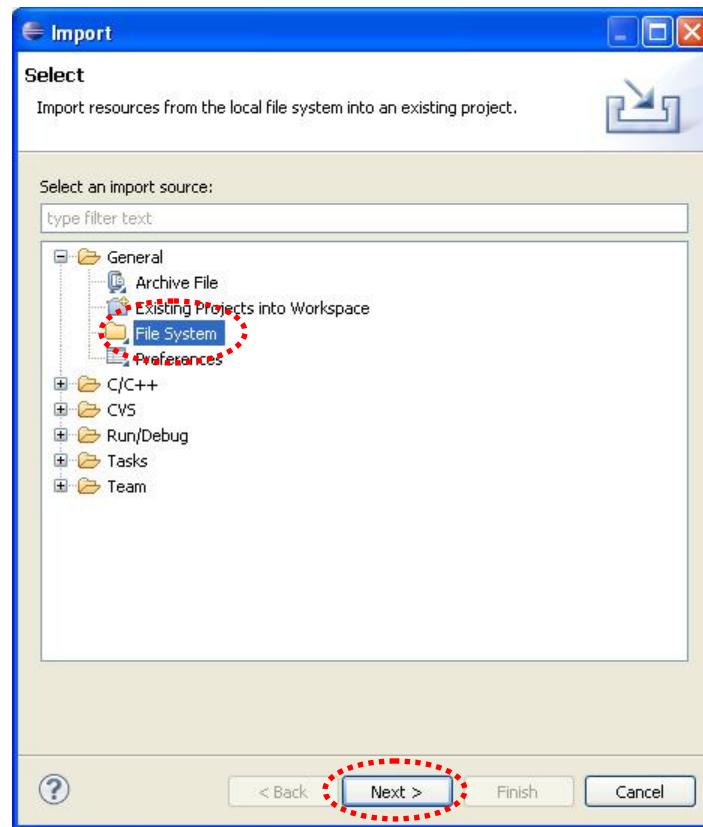
Select the "Project" menu and click on 'build automatically' to de-select it.



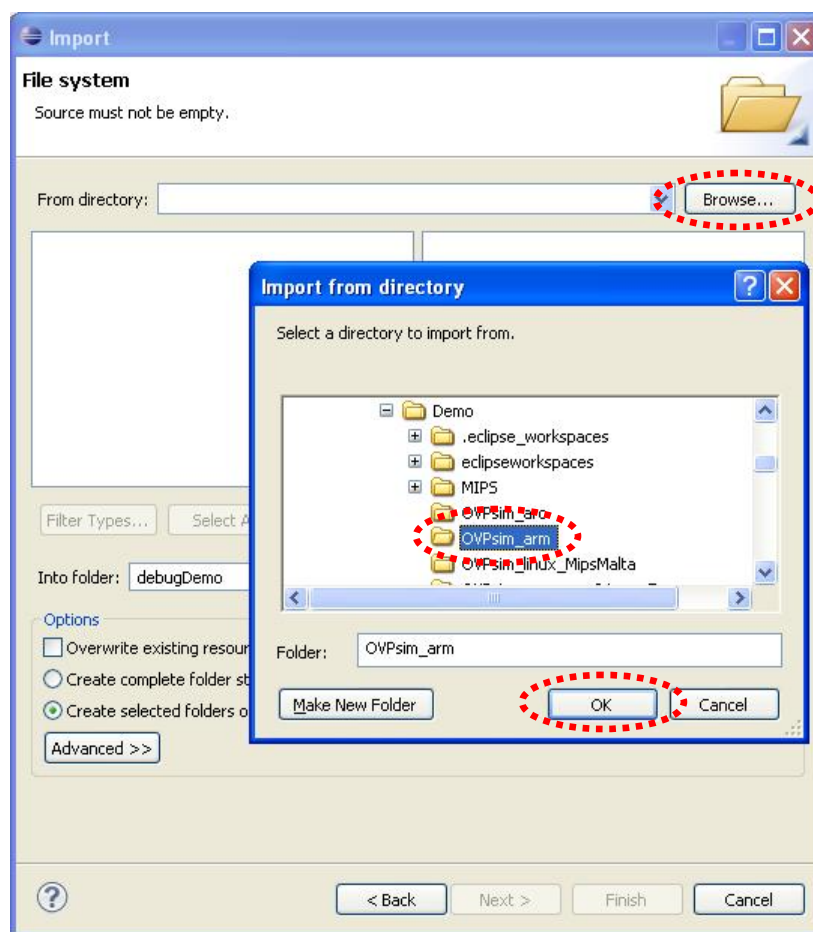
5.4 Importing a Demo to the Project

For the purposes of this document we will import the demo provided as part of an OVPsim installation for the ARM processor, that is, OVPsim_arm. We could equally choose the OVPsim_mips32 or other processor demo. The only difference is the Cross Compiler toolchain setting and the GDB debugger selected. These will be notified in the later sections.

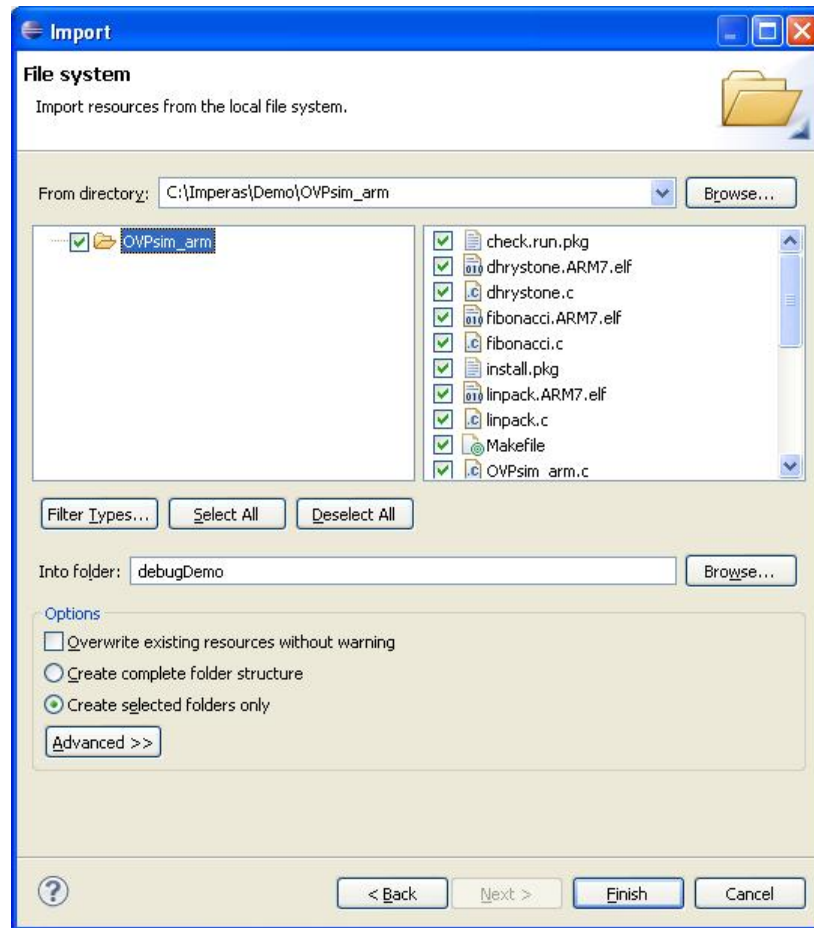
The files are imported to a project by selecting the File->Import menu and selecting 'File System' and followed by 'Next'



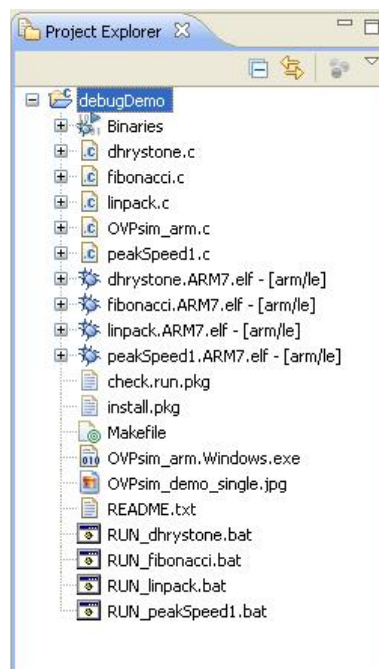
This brings up a further menu from which you may browse to the Imperas installation and the installed Demos. Select the demo directory that you wish to import. This may be any demo supplied as part of an OVP installation.



Once the directory containing the demo files has been selected, you will be prompted for which files you wish to import. In general this is ALL files, if you wish to exclude some files, please ensure that the source files and the exe and elf files are imported.



Once imported your project will contain all the files

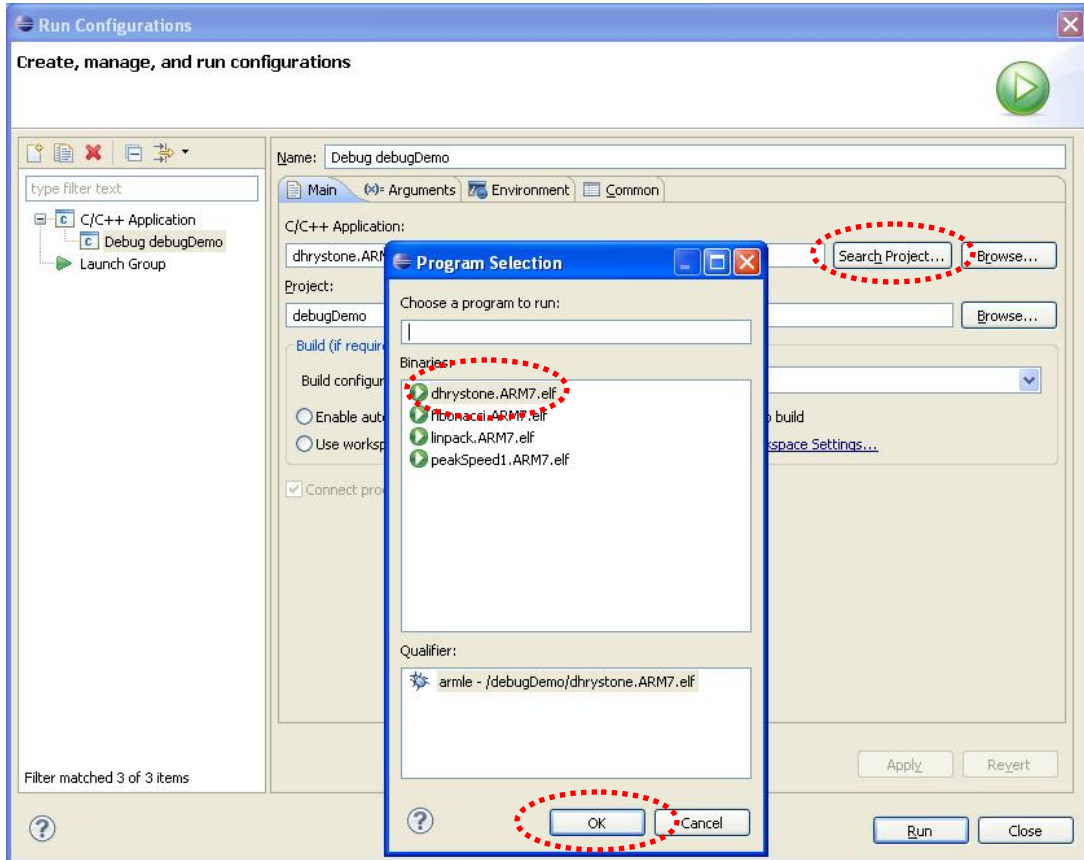


5.5 Creating Launch Configurations

5.5.1 A Debug Configuration

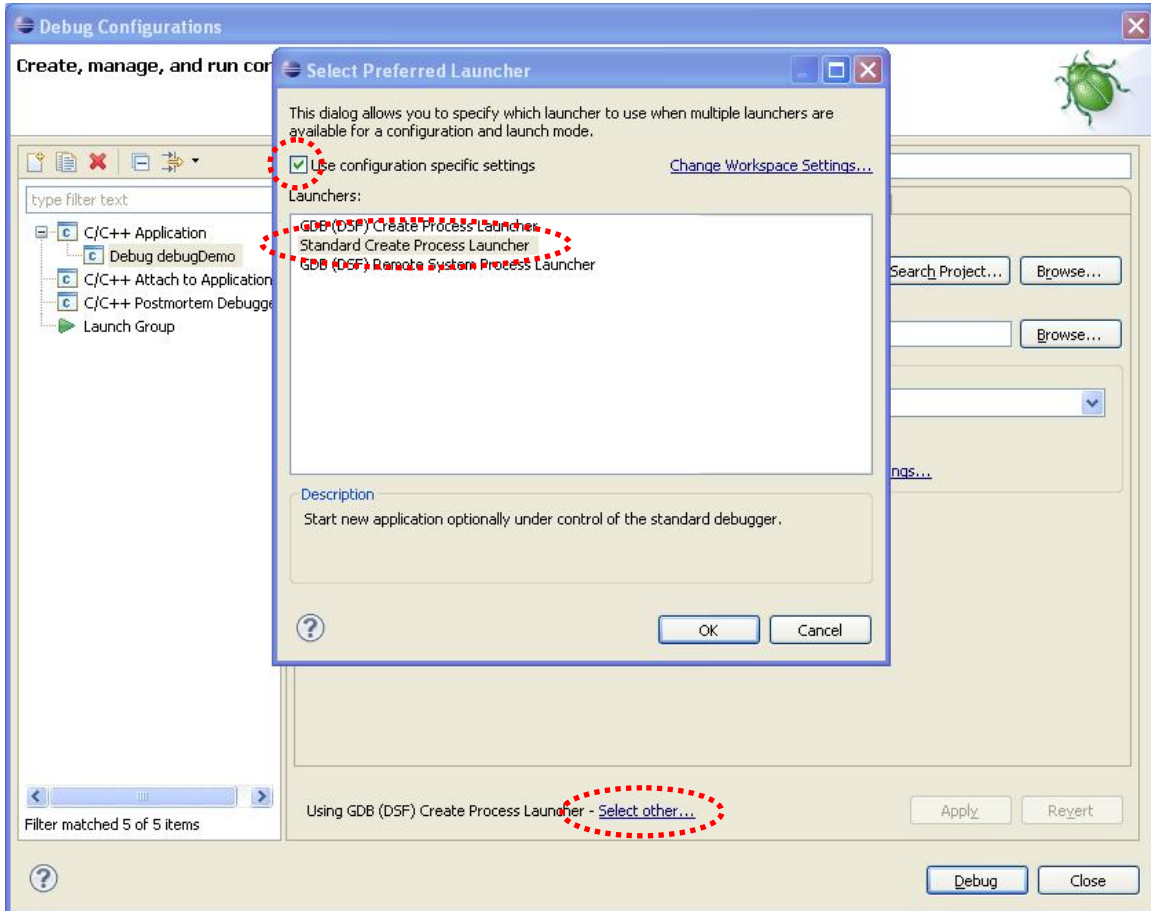
Create a debug configuration for the simulated application by selecting “Open Debug Dialog...” from the “Run” menu. Create a new “C/C++ Application” configuration.

Select the application executable as the C application file using the “search project” button



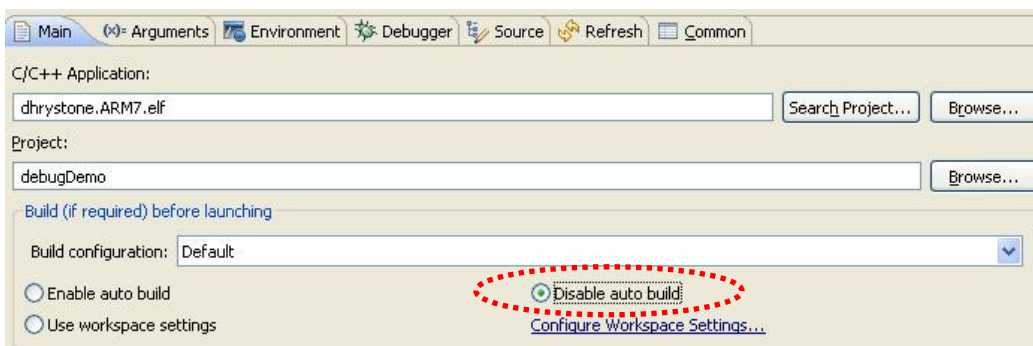
We now need to select the use of the standard debug launcher, not the DSF launcher. This is done by selecting the ‘select other’ option.

Select the ‘Use configuration specific settings’ and select the use of the ‘standard Create Process Launcher’



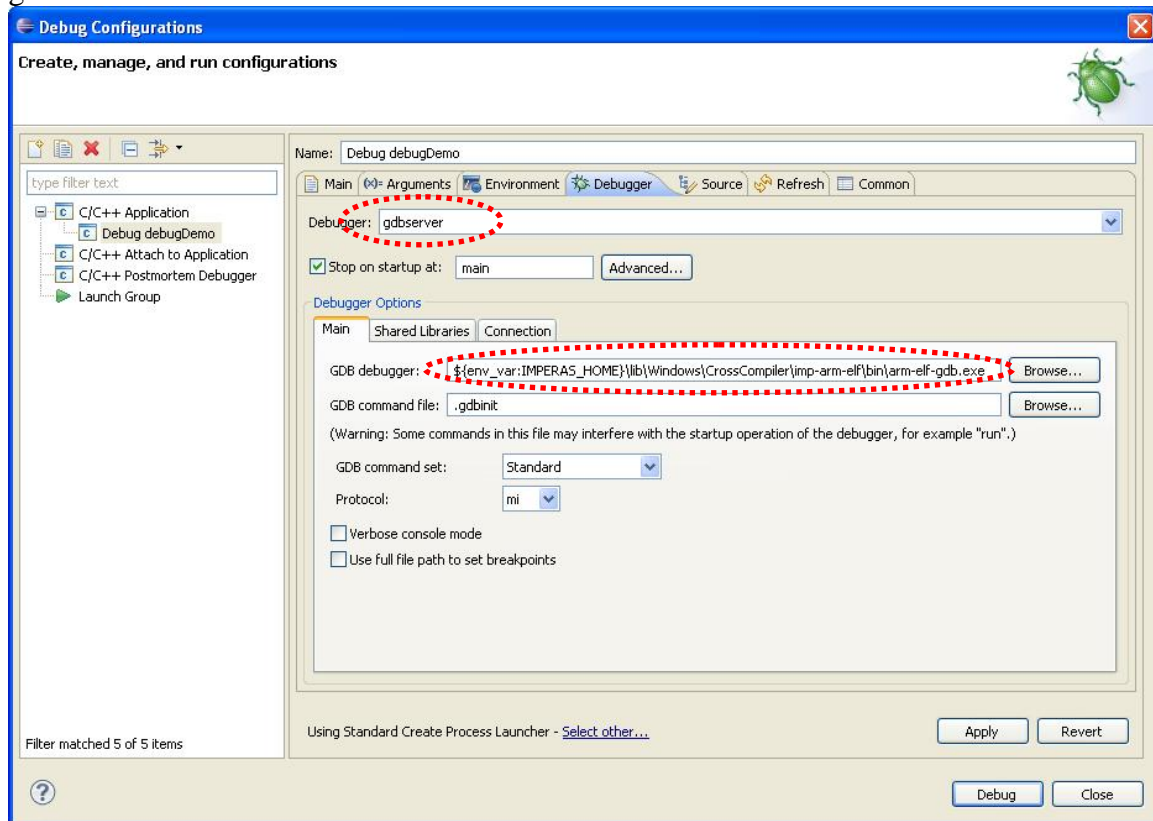
On the “Debugger” tab of this dialog select “gdbserver” as the debugger type and browse for the GDB executable for the simulated processor type.

It is recommended that the auto build feature is also disabled in this launcher.



Assuming you installed OVPsim and the toolchains to the default location “C:\Imperas”, the Gnu tools for simulated processors, including cross-compilers and the Gnu debugger can be found in “lib\Windows\CrossCompiler”.

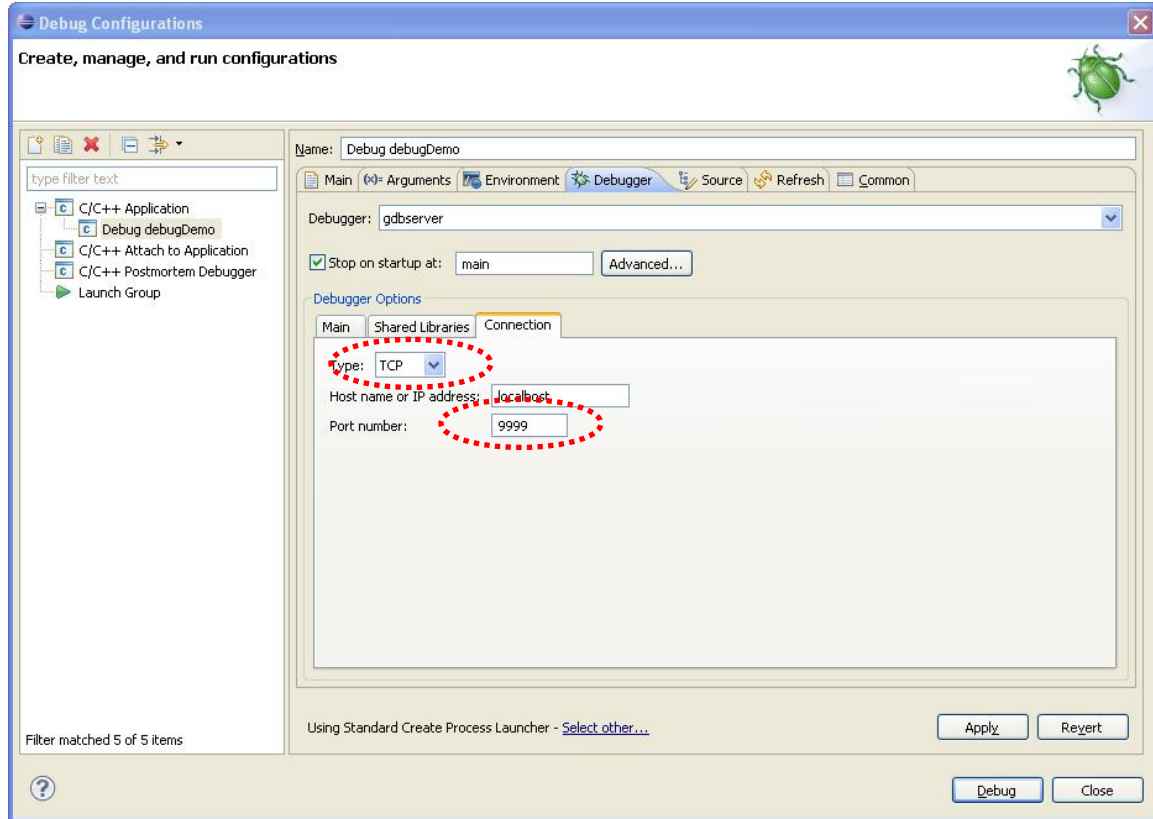
The ARM GDB in this case is found at
“C:\Imperas\lib\Windows\CrossCompiler\CrossCompiler\imp-arm-elf\bin\arm-elf-gdb.exe”.



Instead of using the full path to the fixed location C:\Imperas it may instead be replaced with the environment variable IMPERAS_HOME.

In Eclipse the environment variable is accessed using the `${env_var:IMPERAS_HOME}` construct, as shown.

Finally, specify the target connection settings:



Select “TCP” as the connection.

Choose a port number, here we have initially chosen 9999. You may need to change this if there is a conflict on your host. The port number selected here must match the port number we select on the platform launch we will define next.

Press “Apply” then “Close” to save the debug configuration without starting it.

5.5.2 Creating a Shortcut to Launch the OVPSim Platform

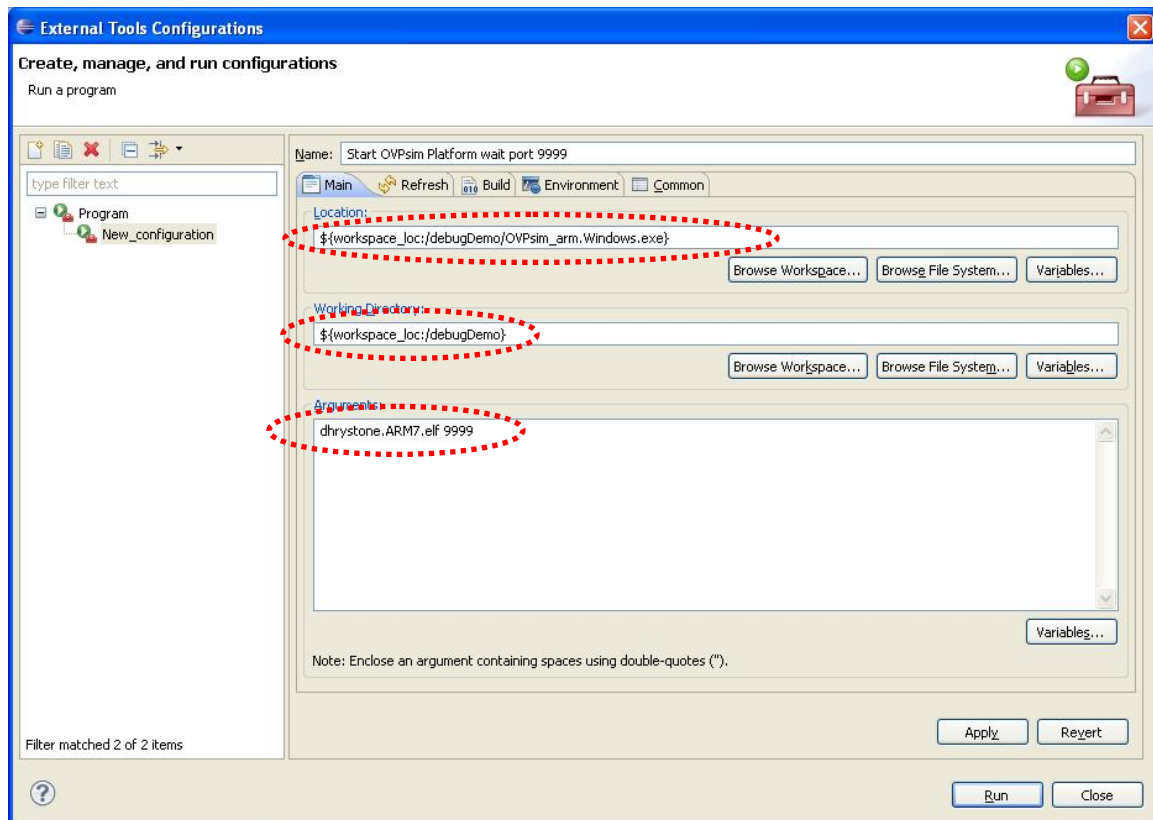
Before each debug session the OVPSim platform must be launched on the host machine. The Eclipse “External Tools” facility is a convenient way to do this. Select “Open External Tools Dialog...” from the “External Tools” menu within the “Run” menu.

Create a new tool configuration and give it a meaningful name. Specify the location for the platform executable by pressing the “Browse Workspace...” button and selecting the “platform.<IMPERAS_ARCH>.exe” file built earlier. Press the “Browse Workspace...” button for the working directory, and select the top-level project directory.

The arguments added into the arguments section must match the arguments that the executable platform expects.

In this example the platform takes two arguments; the first argument is the application elf file and the second argument is the port number on which it waits for a debugger connection.

Specify the port 9999 initially, matching the port we chose in the debug configuration in section 5.5.1.



Press “Apply”, then “Close” to save the external tool configuration without starting it.

5.6 NOTE: Depending upon the Demo you have selected, you may need to modify the platform C source file so that it is a

“debuggable platform” as described in section 5.1, Command Line arguments for debugging an ICM Platform

If the platform has the Imperas Command Line Parser (CLP) included it makes available command line options to enable and control debugging of an application on the processor in a platform without the need to make any changes.

5.6.1 Specifying the debugger connection details

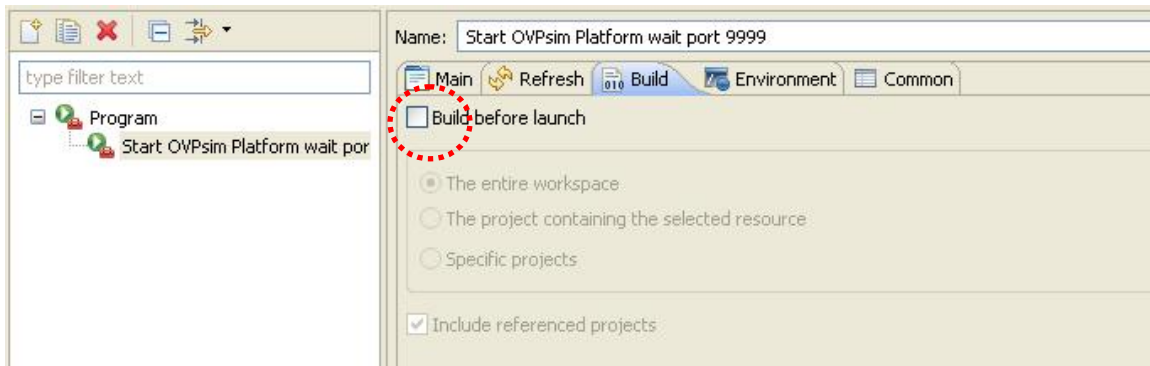
The debug port is enabled by specifying the argument `--port <port number>` on the command line. A specific port number may be specified or by setting *port number* to 0 the next available port is opened.

5.6.2 Nominating the debugged processor

In an OVPSim simulation only a single processor may be connected to a GDB debugger. this requires that the processor is selected using the `--debugprocessor <processor name>`. In this case the processor name is the instance name in the platform, for example *platform/cpu0*

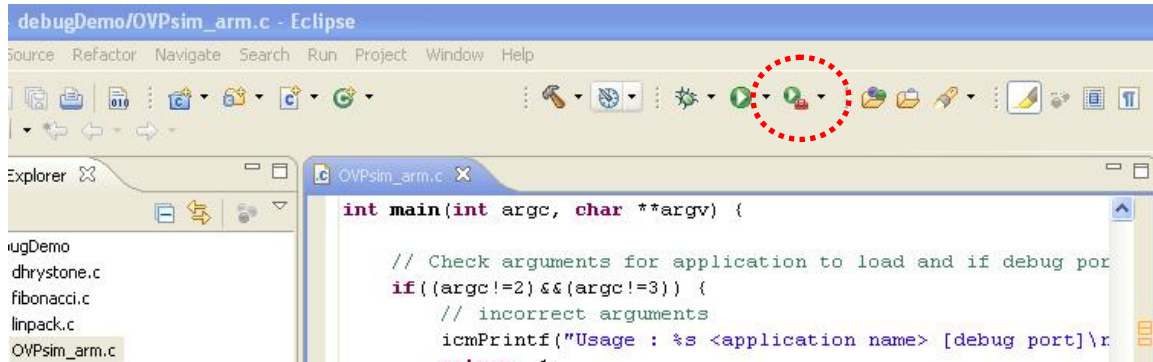
Creating a Debuggable ICM Platform

Ensure the ‘Build before launch’ checkbox is not selected. We want to ensure we have control over when the project is built, as we are using multiple launches.



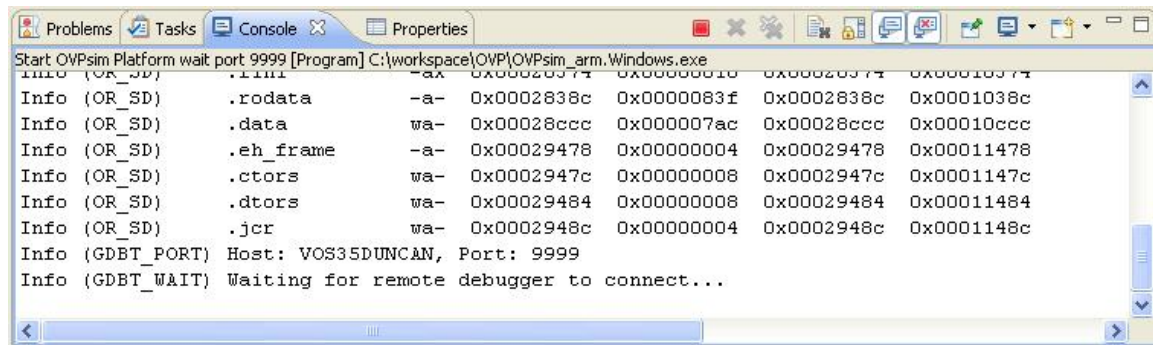
5.6.3 Testing the Debug Configuration

Earlier sections showed how to create a debug configuration for a simulated application and an external tool shortcut to launch an OVPSim platform. To test that we are ready to debug, first launch the OVPSim platform by running the external tool shortcut. The easiest way to do this is to press the external tool icon on the Eclipse toolbar.



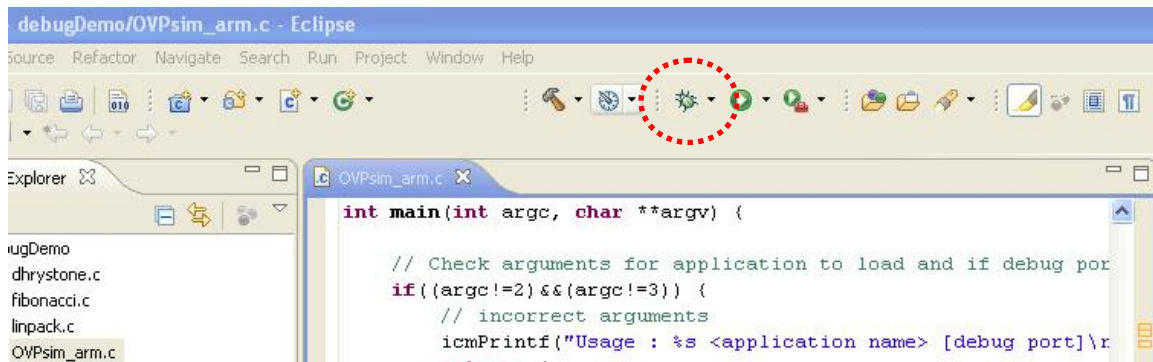
The first time you press this icon you will need to select the OVPSim platform shortcut from the list in the external tool dialog, then press the “Run” button. Subsequently it will automatically re-run the external tool you ran last.

The Eclipse Console view will show the output from the OVPSim platform executable:



If port 9999 is already in use on your system the console will show an error message and you will need to modify both the debug configuration and the external tool configuration to select a new port number.

Once the OVPSim platform is running and waiting for a debugger to connect, launch the debug configuration we created earlier. Again, the easiest way to do this is to press the debug icon on the Eclipse toolbar.

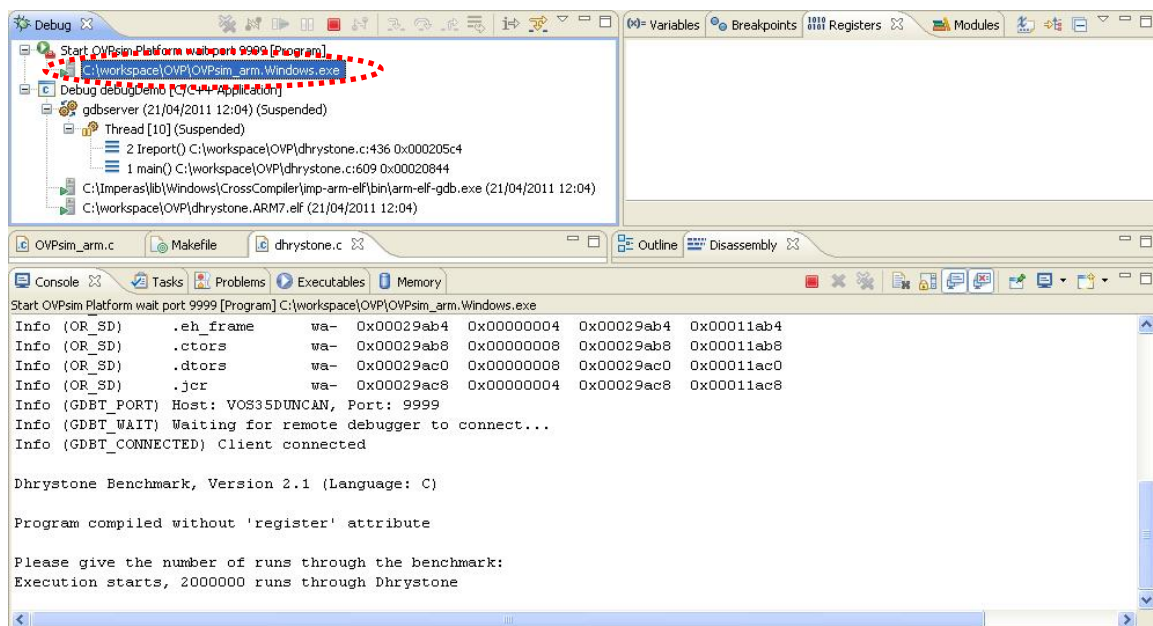


If debugging is configured and started correctly, Eclipse will switch (or prompt to switch) to the debug perspective, with the simulated application stopped at the entry to its main function.

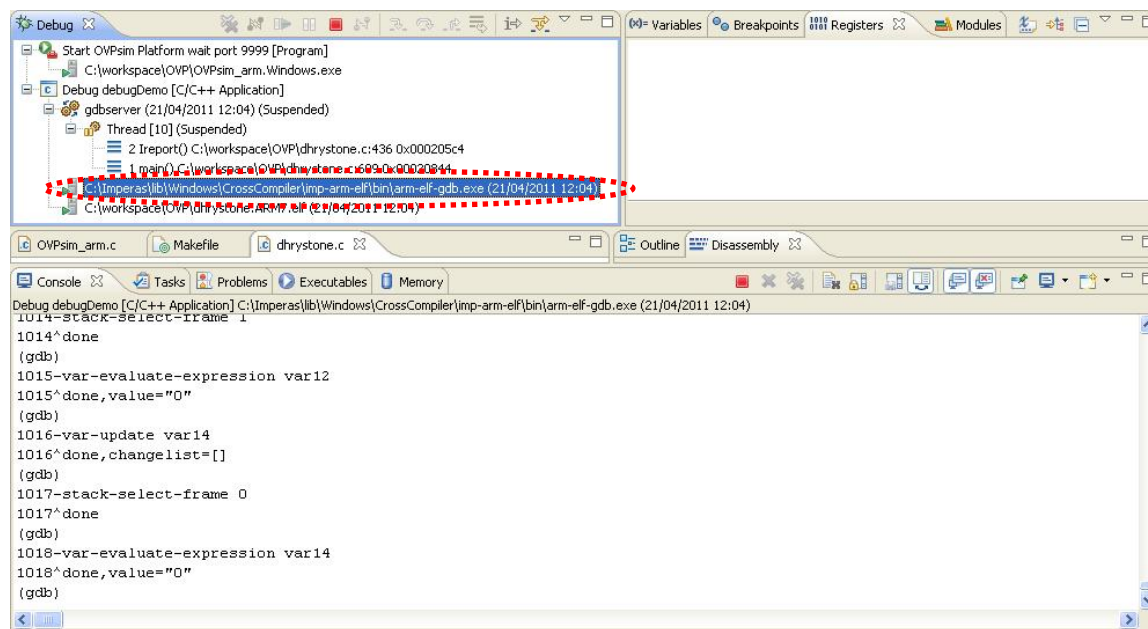
5.6.4 Switching Console Output

The output from the console terminal window depends upon the selection made in the Debug window.

Here we have the output from the OVPSim platform executing after selecting the platform executable.

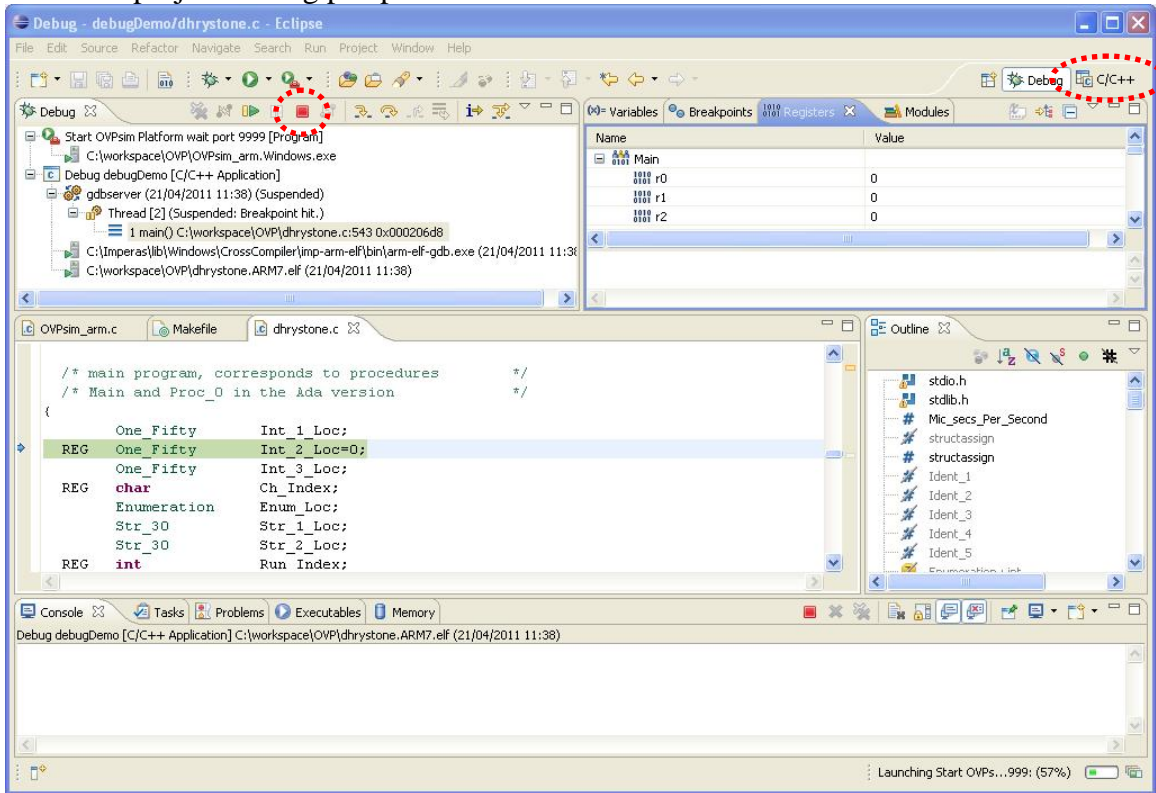


In the next picture we see the GDB output, showing the commands passed between Eclipse and GDB with verbose selected, in the console after selecting the GDB debugger.

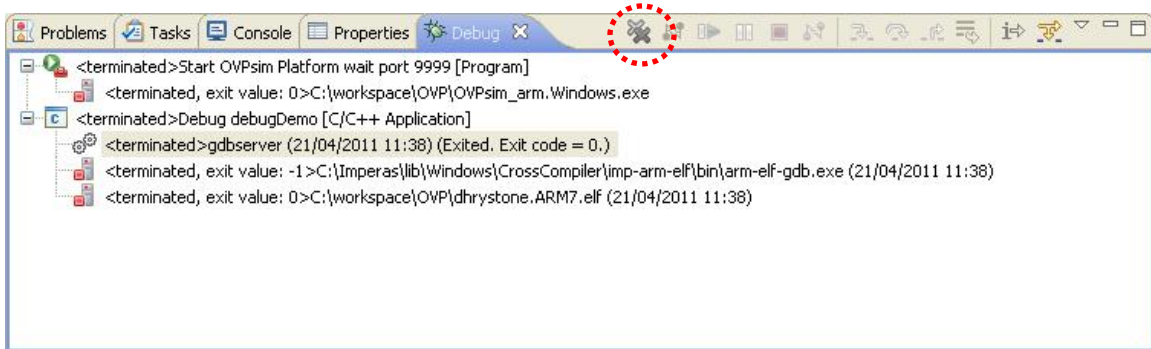


5.6.5 Terminating the Debug Session

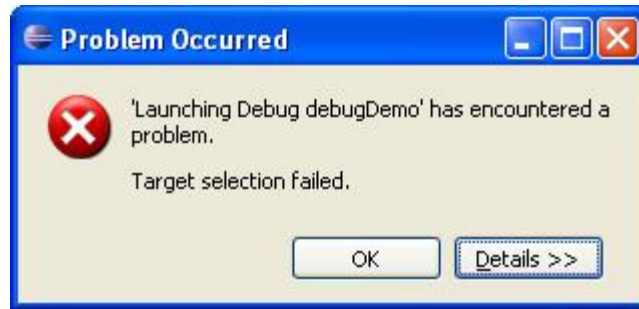
Press the stop icon to stop debugging, then the “C/C++” perspective icon to return to the normal C project editing perspective.



There will be a number of terminated processes shown in the debug session window that can be cleaned up using the icon shown.



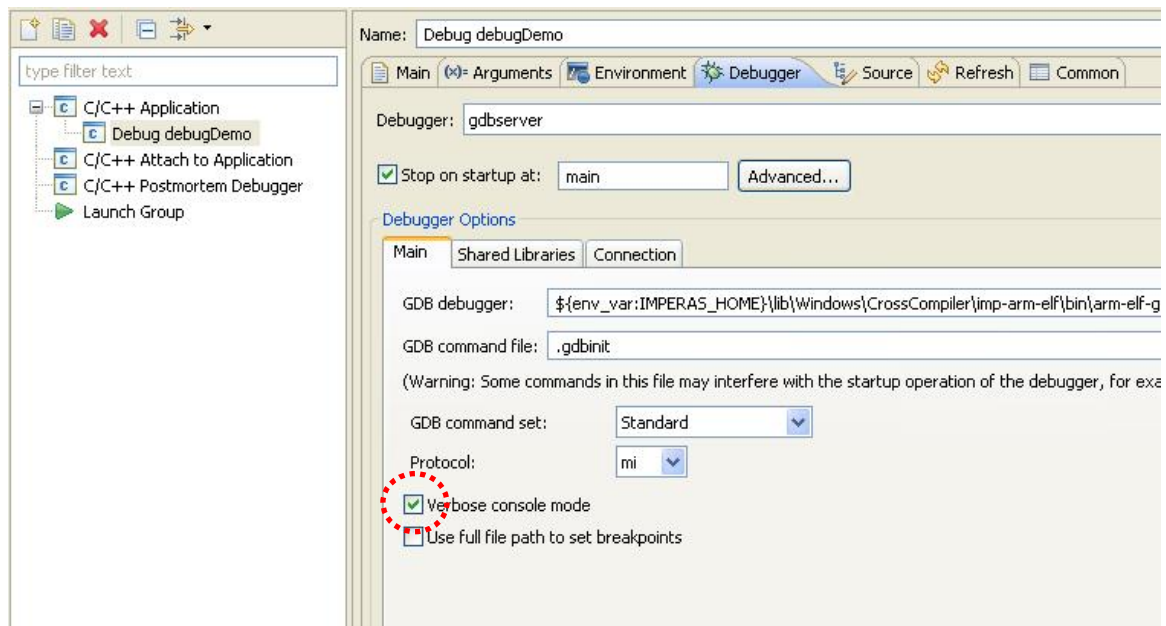
If the debugger is unable to connect to the OVPSim platform you will see an error dialog.



Check that the OVPsim platform is running, and that the port number it is waiting for (section 5.2.1) matches the port number in the debug configuration settings (section 5.5.1).

5.6.6 Verbose GDB Console Output

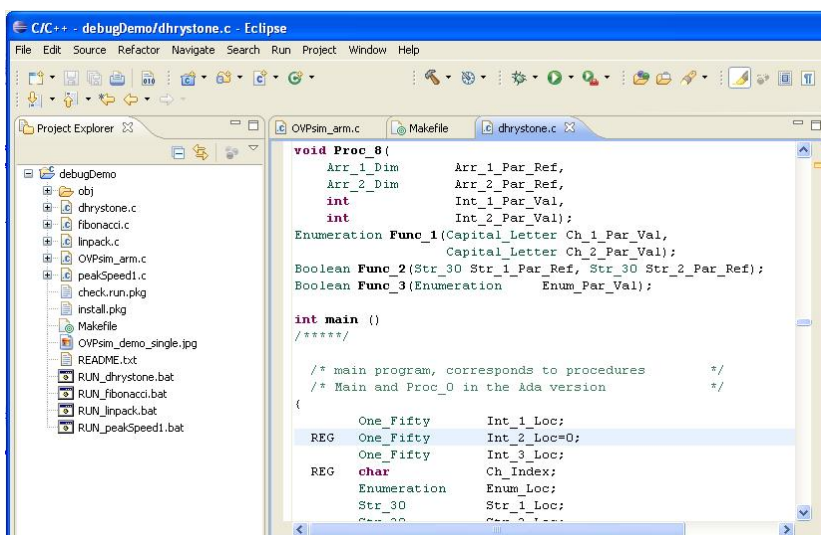
To enable further information from the interface between Eclipse and the GDB you may turn on a verbose output from the debugger tab of the GDB launcher.



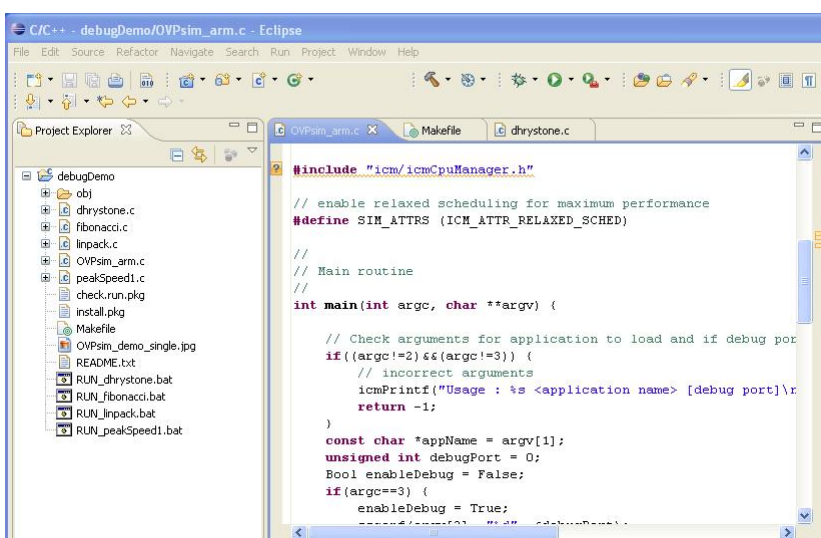
5.7 An Example Debug Session

Section 5.3 showed how to prepare an Eclipse project to debug an application running on a processor in an OVPsim virtual platform. This section provides a quick overview of some common debugging tasks in Eclipse. For a full explanation of Eclipse C/C++ debugging please see the *C/C++ Development User Guide* available in the Eclipse help system or online at www.eclipse.org.

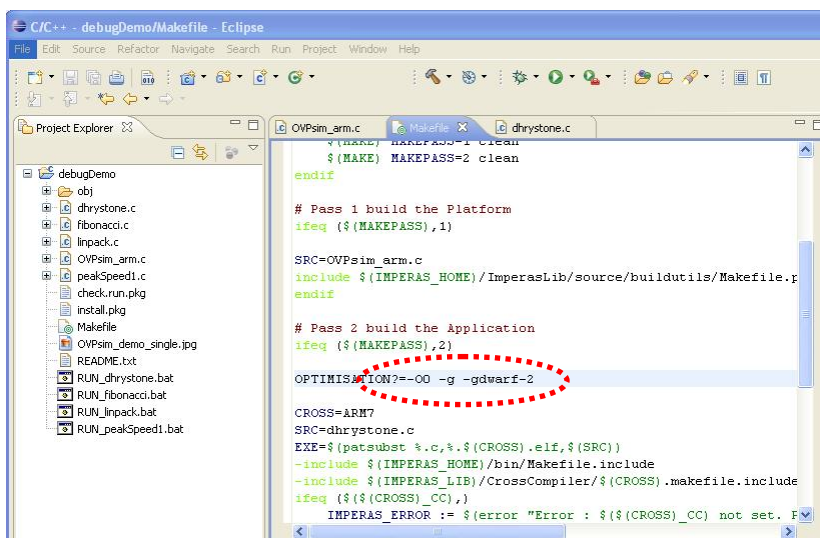
The application we will look at is the Dhrystone benchmark code.



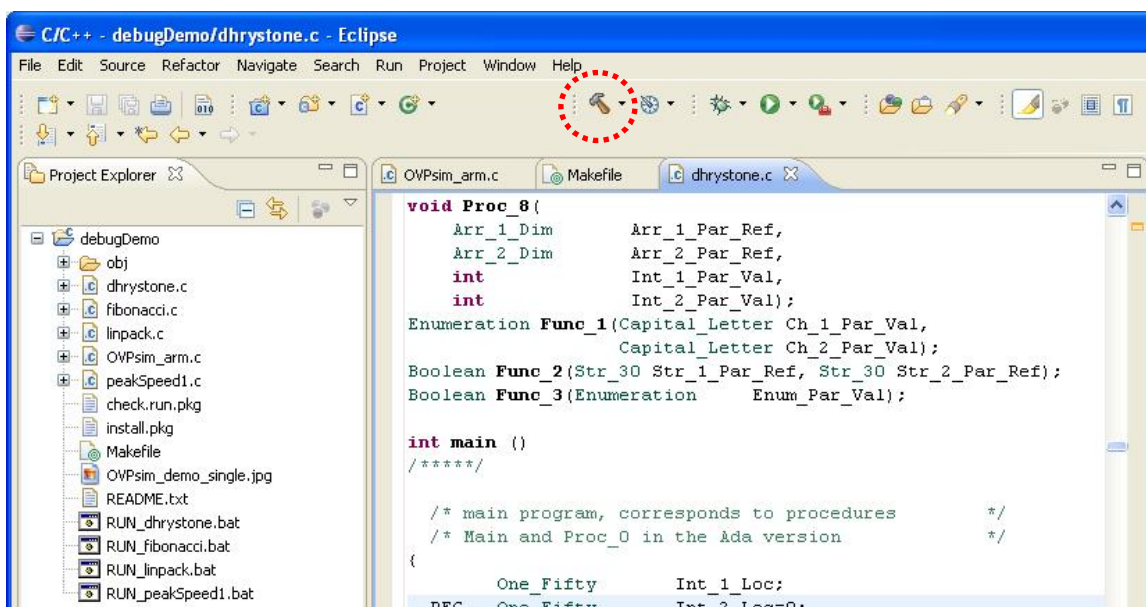
With a single processor platform setup to receive a port number for opening the debug connection.



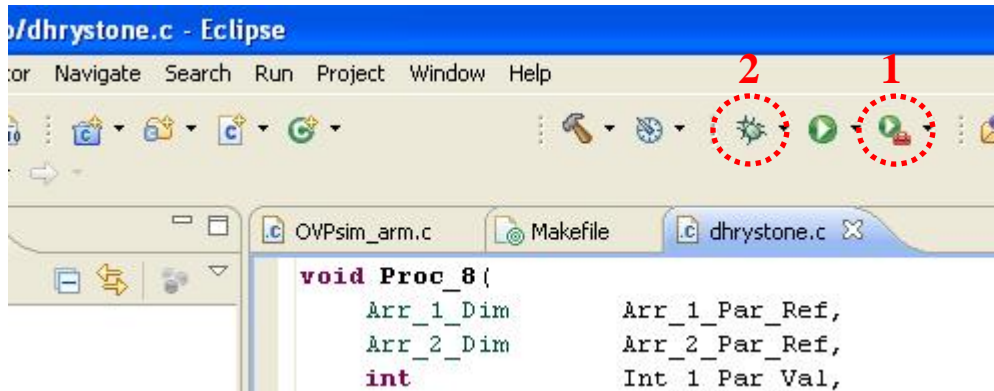
We also need to ensure that the external Makefile that is being used, is setup to build an application elf file containing debug information.



After any editing in the project of the application or platform, rebuild by pressing the build icon on the Eclipse toolbar. Any syntax errors will be reported in the console view and highlighted in the editor window.



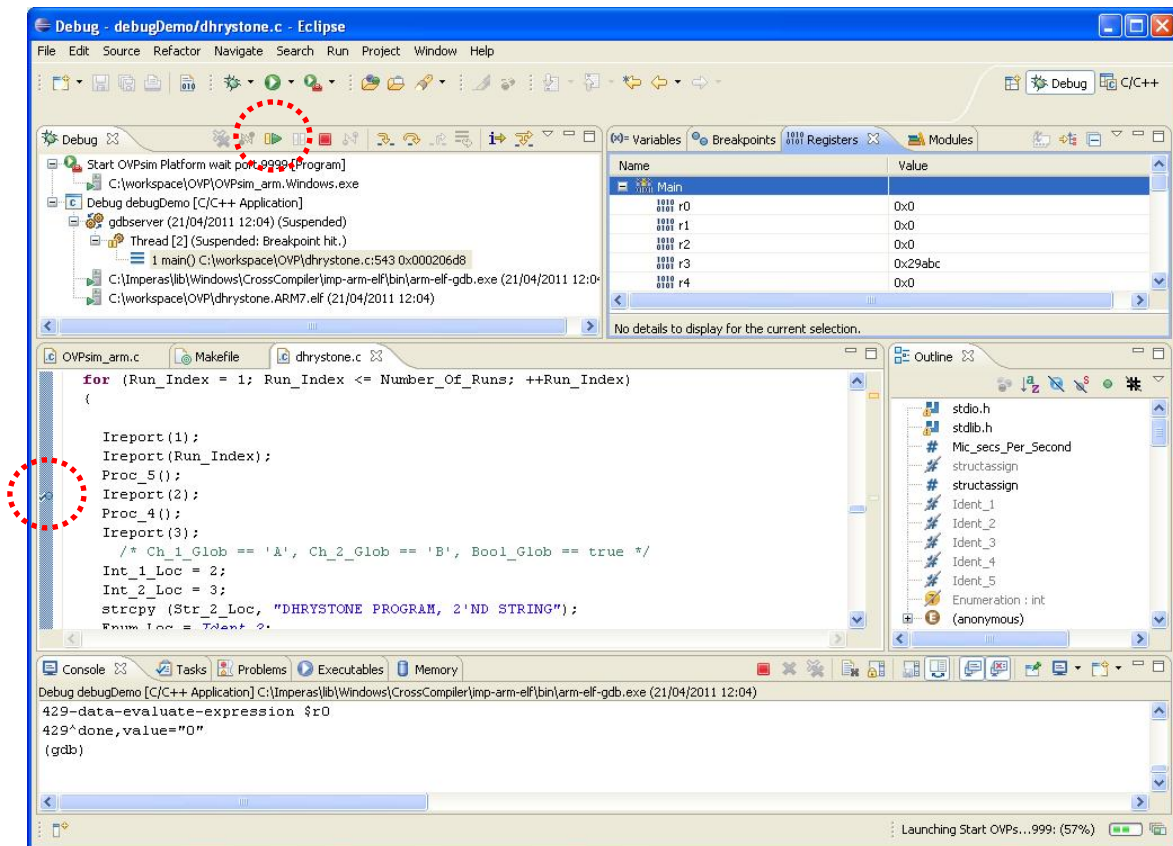
Start debugging by clicking the external tool toolbar button (1), then the debug toolbar button (2).



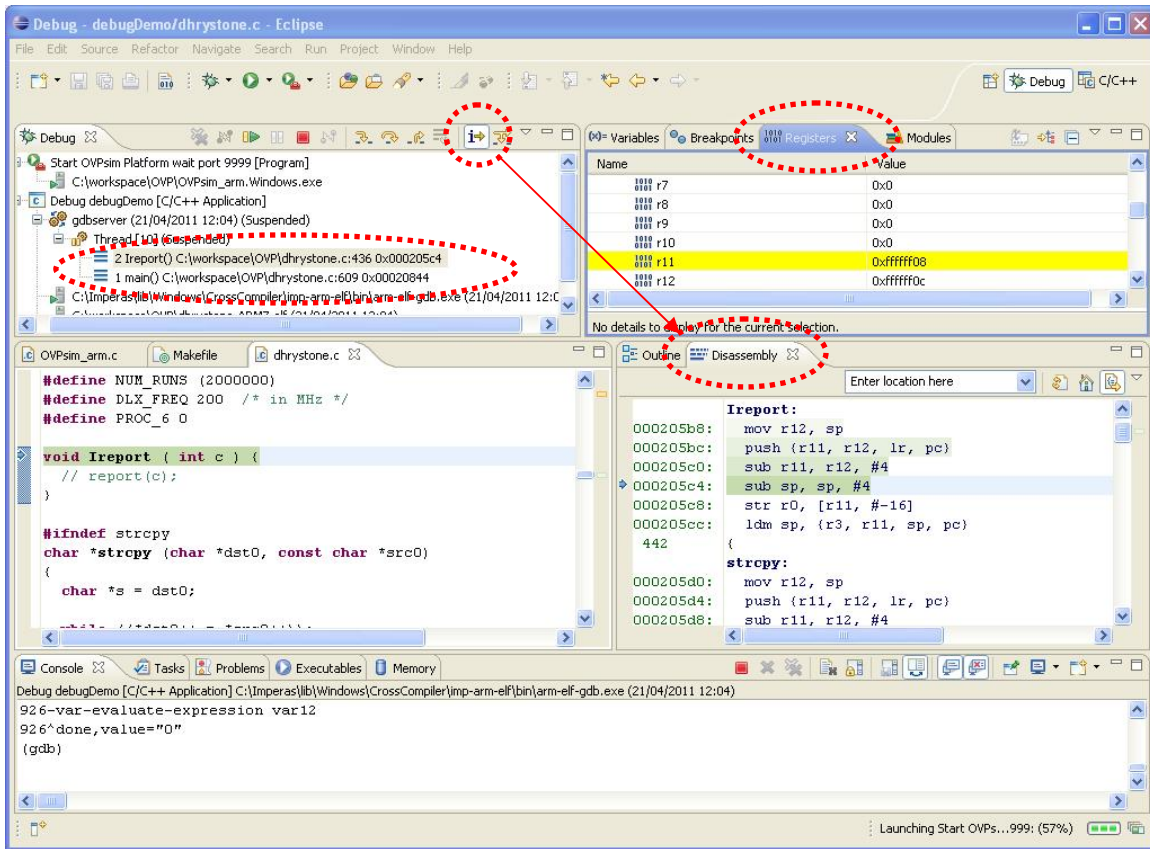
Eclipse will switch to the debug perspective automatically.

Set a breakpoint on the call to `Ireport(2)` by double-clicking in the margin next to the call.

Run to the breakpoint by pressing the resume toolbar button.



Show the processor registers by switching to the registers pane. Bring up the disassembly view by clicking the instruction stepping mode toolbar button. The debug view shows the call stack for the debugged application.



As you step, you will see the registers that are modified being highlighted.

Finally, run to completion by double clicking on the breakpoint to remove it, then pressing the resume button again. Program output is shown in the Eclipse platform console view.

###