MULTICORE DESIGN SIMPLIFIED
imperas

Imperas Debugging with ARM DS-5

CONFIDENTIAL DOCUMENTATION

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

Author:	Imperas Software Ltd.
Version:	1.0
Filename:	Imperas_Debugging_with_ARM_DS-5.doc
Project:	Imperas ARM DS-5 Debugger Integration
Last Saved:	Tuesday, 20 January 2015

Table of Contents

1 Terminology

The following are terms used in this document and their meaning.

OVP – the initiative created by Imperas to promote the opening and standardization of the Imperas developed simulation APIs of VMI, ICM, BHM, PPM, and MMC. Hosted at www.OVPworld.org

OVPsim – the simulator developed by Imperas and made available for download from www.OVPworld.org. Free for non-commercial usage. License fees payable to Imperas for commercial usage. Runs on Windows and Linux.

Imperas Professional Simulators - the simulators licensed commercially by Imperas - marketed in the products C*DEV, S*DEV, M*DEV, M*SDK and known technically as CpuManager.

ARM DS-5 - the Eclipse based GUI and debugger from ARM.

RDDI - ARM API - Remote Device Debug Interface

CADI - ARM API - Cycle Accurate Debug Interface

2 Introduction

Virtual Platforms that incorporate Imperas OVP Fast Processor models of ARM cores can be debugged using the ARM DS-5 Debugger.

The functionality implemented in this Imperas DS-5 interface is similar to that of the GDB used in Eclipse. This interface does not provide to DS-5 all the features in the Imperas simulators, nor does it provide support for all the features that are possible in DS-5 when connected to hardware etc. (see Limitations sections for list.)

This document explains the required setup, walks you through provided example platforms of an ARM Cortex-A5UP and ARM Cortex-A9MPx2, and describes all that is necessary to configure DS-5 to use your own Imperas/OVP virtual platforms and to use DS-5 to debug software running on your platforms.

There are several sections to this document:

- installation - required once to get started
    - files to be installed
    - environment setup
- debugging software running on the example platforms
- configuring DS-5 to use your own platforms - whenever you need to debug on a new platform

The Imperas simulators/platforms communicate with ARM DS-5 using the RDDI interface over a socket as two separate processes. The Imperas RDDI interface provides this communication.

The Imperas RDDI interface is provided as a shared library that is loaded by ARM DS-5.

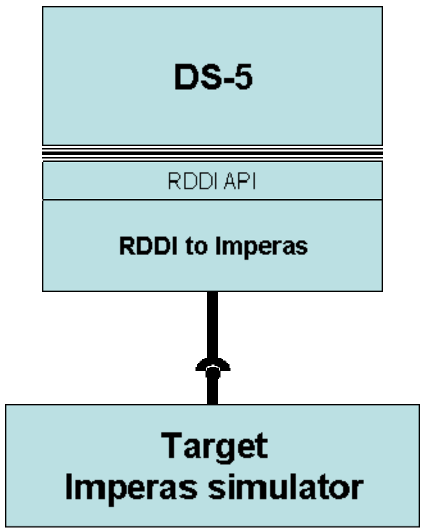


Figure 1: Imperas simulator connection to ARM using RDDI interface and socket

The Imperas simulators and RDDI interface are available for Windows/Linux 32/64 installations.

3 Installation

To be able to debug an application executing on an Imperas/OVP virtual simulation platform with DS-5 requires DS-5 to be setup to include an interface to the Imperas simulators and with knowledge of the platforms. The installation process needs to be followed once for particular versions of ARM DS-5 and Imperas installations. It is not repeated for subsequent debug sessions, just when installing a new version of DS-5 or Imperas.

Before you start with this installation process you need to have installed:

Imperas simulator	: Imperas DEV or SDK package
ARM DS-5	: ARM DS-5 installation[1]

The installation process installs:

- Imperas RDDI interface into the ARM DS-5 installation
- Board Support Package for each platform to be used

Finally the execution environment is setup to allow communication between the interface layers.

3.1 Installing an Imperas Simulator

Please refer to the Imperas installation instructions and confirm you have the Imperas simulator running by executing one of the example/demo platforms and applications

3.2 Installing ARM DS-5

Please refer to the ARM installation instructions and confirm you have a configured installation of DS-5 where you can start Eclipse, connect to one of the ARM supplied Fixed Virtual Platforms (FVP) and can run and debug software on it.

3.3 Installing the Imperas RDDI Interface Library

The Imperas RDDI Library files are found in the standard Imperas installation at IMPERAS_HOME/bin/IMPERAS_ARCH, for example the Linux32 library file, libImperas_RDDI.so[2], would be found at IMPERAS_HOME/bin/Linux32

The Imperas library file needs to be copied into the ARM DS-5 installation. For example the Linux 32-bit file is copied into

<ROOT>/sw/eclipse/dropins/plugins/com.arm.rddi.native_<version-specific>/com/arm/rddi/linux/i386

This directory is version specific but is the same directory that contains the ARM RDDI CADI library, ‘librddi-debug-cadi.so.2’ (on Linux 32-bit)

[1] The current Imperas RDDI interface has been verified with ARM DS-5 version 5.17.0 build 5170015
[2] libImperas_RDDI.so is the shared library for Linux. The library for Windows is libImperas_RDDI.dll

The library is renamed during the copy to adhere to the ARM DS-5 requirements, for example

Linux

libImperas_RDDI.so is copied to libImperas_RDDI.so.2

Windows

libImperas_RDDI.dll is copied to Imperas_RDDI_2.dll

This installation process is carried out by a setup script that is described in detail in section 3.4.1 Setup on Linux Operating Systems and section 3.4.2 Setup on Windows Operating System below.

The Imperas RDDI library is referenced by name in the board definition configuration file, for example the file a5up_bare_metal.xml that is referenced in the example in section 7.2.1.1 Platform file (a5up_bare_metal.xml) later in this document using the rddi_type field

```
<rddi type="ARM_RDDI"/>
```

3.3.1 Installing the FLEXlm license key

To run the Imperas RDDI interface library a FLEXlm feature key is required. Please obtain this from Imperas, and place in same file as other Imperas FLEXlm keys.

3.4 Setting up the Execution environment

3.4.1 Setup on Linux Operating Systems

3.4.1.1 Setup Script

A script is provided, IMPERAS/bin/setup_RDDI.sh, that when sourced in a bourne shell will provide functions to allow the installation and setup of the Imperas RDDI interface.

Function:	**installImperasRDDI**
Arguments:	Installation directory for DS-5, for example /home/user/DS-5
Description:	Installs the Imperas library objects into the ARM DS-5 installation

Function:	**setupImperasRDDI**
Arguments:	Installation directory for DS-5, for example /home/user/DS-5
Description:	Adds additional environment variables so that the Imperas RDDI interface can operate correctly.

3.4.1.2 Configuring to use ARM DS-5 and Imperas RDDI Interface

Source the script setup_RDDI.sh in a shell.

The first time you use an installation of ARM DS-5 with an installation of Imperas you will need to invoke the installImperasRDDI and pass the root directory of the ARM DS-5 installation you wish to use.

```
installImperasRDDI /home/user/DS-5
```

In every new shell that you wish to invoke ARM DS-5 with Imperas you will need to invoke the setupImperasRDDI to create the correct environment.

```
setupImperasRDDI /home/user/DS-5
```

or you may invoke this function without an argument if the IMPERAS_DS5_INSTALL environment variable is set.

```
setupImperasRDDI
```

3.4.2 Setup on Windows Operating System

A script is provided, setup_RDDI.bat, that when executed will copy the Imperas files into the ARM DS-5 installation that is specified in the script and setup the environment for the use of Imperas RDDI.

This script need only be run once for an installation of Imperas and ARM DS-5.

This script includes reference to the default installation location for ARM DS-5 i.e. C:\Program Files\DS-5; if ARM DS-5 has been installed into a different location please update the script.

4 Installing the Imperas Cortex-A5UP Example

The board support packages for the Imperas provided example must be added into the configuration database shipped with DS-5. The database is found within the ARM DS-5 installation at <install root>/sw/debugger/configdb.

Each board support package provides:

• Boards	information about all the platforms you want to connect to
• Cores	if you've added your own custom cores & register definitions,
• Scripts	launch script for your target

The procedure for adding a user database to DS-5 in the Eclipse preferences can be found on the ARM website at the following link:

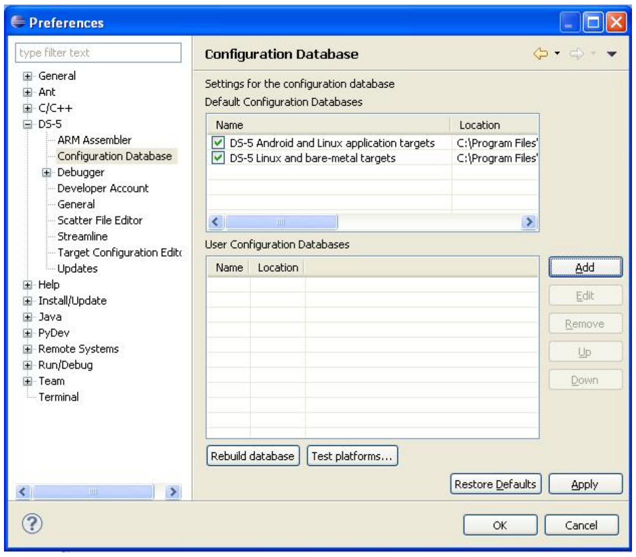
http://infocenter.arm.com/help/topic/com.arm.doc.dui0446p/deb1359985233115.html

The information contained in the above link is also provided in the following paragraphs.

1. Launch Eclipse for DS-5.
2. Select Preferences from Windows menu.
3. Expand the DS-5 configuration group.
4. Select Configuration Database.

5. Locate the new database:

Initially the standard built-in configurations are available. We will add new configuration databases for each project separately. These may have been combined into a single custom database.



Configuration Database

Settings for the configuration database

Default Configuration Databases

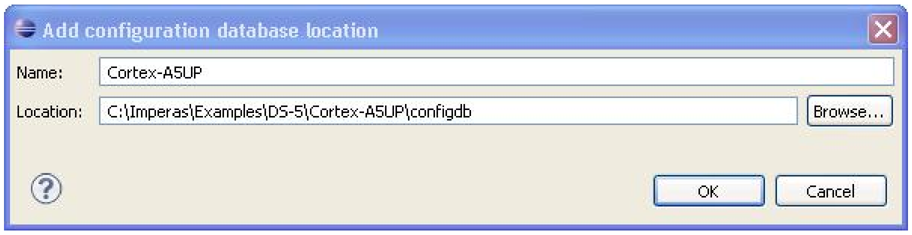
Name	Location
✓ DS-5 Android and Linux application targets	C:\Program Files'
✓ DS-5 Linux and bare-metal targets	C:\Program Files'

User Configuration Databases

Name	Location

Select the 'add' button to select the new configuration database to be imported.

6. Browse to the Imperas installation to find the platforms to use. First select the Cortex-A5UP example. Select the entire directory.



Name: Cortex-A5UP

Location: C:\Imperas\Examples\DS-5\Cortex-A5UP\configdb

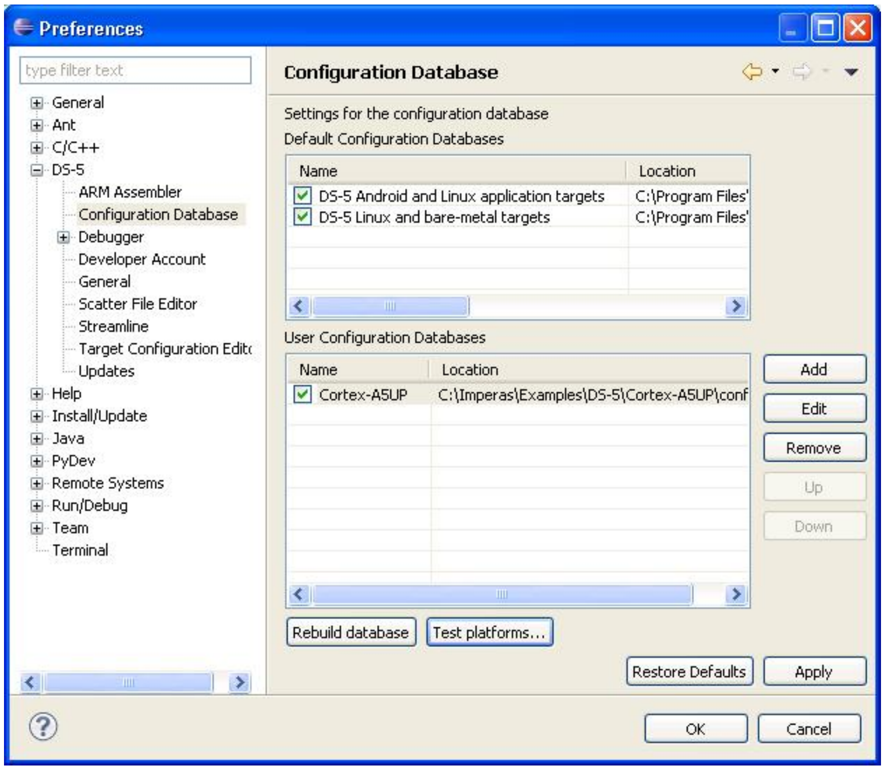
7. Add the name for the entry being added, for this example Cortex-A5UP
   Click OK to close the dialog box.

8. Position the new database:
   a. Select the new database.
   b. Click Up or Down as required.

   Note
   DS-5 provides built-in databases containing a default set of target configurations. You can enable or disable these but not delete them.

9. Click Rebuild database…

Once the database has been selected we need to rebuild the database for the changes to take effect.



Configuration Database

Settings for the configuration database

Default Configuration Databases

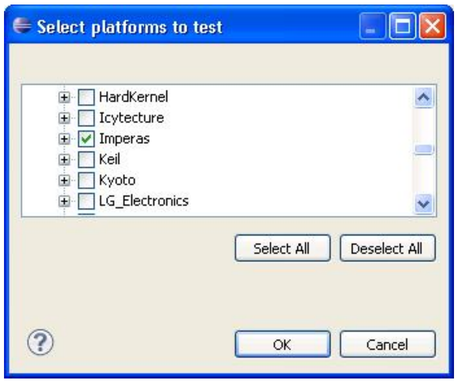
Name	Location
✓ DS-5 Android and Linux application targets	C:\Program Files'
✓ DS-5 Linux and bare-metal targets	C:\Program Files'

User Configuration Databases

Name	Location
✓ Cortex-A5UP	C:\Imperas\Examples\DS-5\Cortex-A5UP\conf

10. Click Test Platforms…

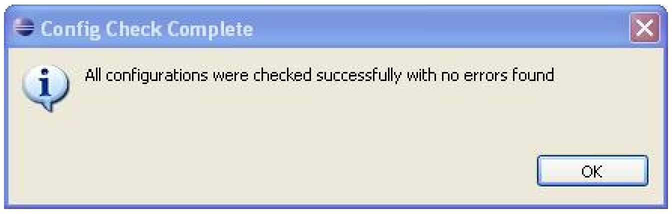
Once the database has been re-built we can test the Imperas platforms we have added by selecting the ‘test platforms’ button and then clicking on the tabs associated with the Imperas database.



- ☐ HardKernel
- ☐ Icytecture
- ✓ Imperas
- ☐ Keil
- ☐ Kyoto
- ☐ LG_Electronics

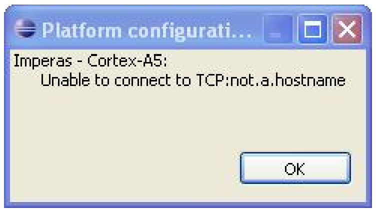
The testing of the platform checks that the project and platform definitions are correct and consistent and checks that the Imperas RDDI interface library can be correctly invoked.

The test should pass without any errors, as shown below



All configurations were checked successfully with no errors found

If you see an error, such as the following, please check that the Imperas RDDI library file for the current Imperas release has been correctly copied into the ARM DS-5 installation directory.



Imperas - Cortex-A5:
Unable to connect to TCP:not.a.hostname

Please see section 3.3 “Installing the Imperas RDDI Interface Library” for further information and the scripts to perform the installation.

11. Click OK to close the dialog box and save the settings.

Whenever changes are made to the files within the configuration database you must repeat steps 9 and 10 to update DS-5.

5 Debugging the Imperas Cortex-A5UP Example

The example is found in the Imperas installation at Imperas/Examples/Interfaces/ARM_DS-5

The following sections show how to import the Cortex-A5UP example as a new project into Eclipse for DS-5 and how to launch debug sessions for the example applications that are provided.

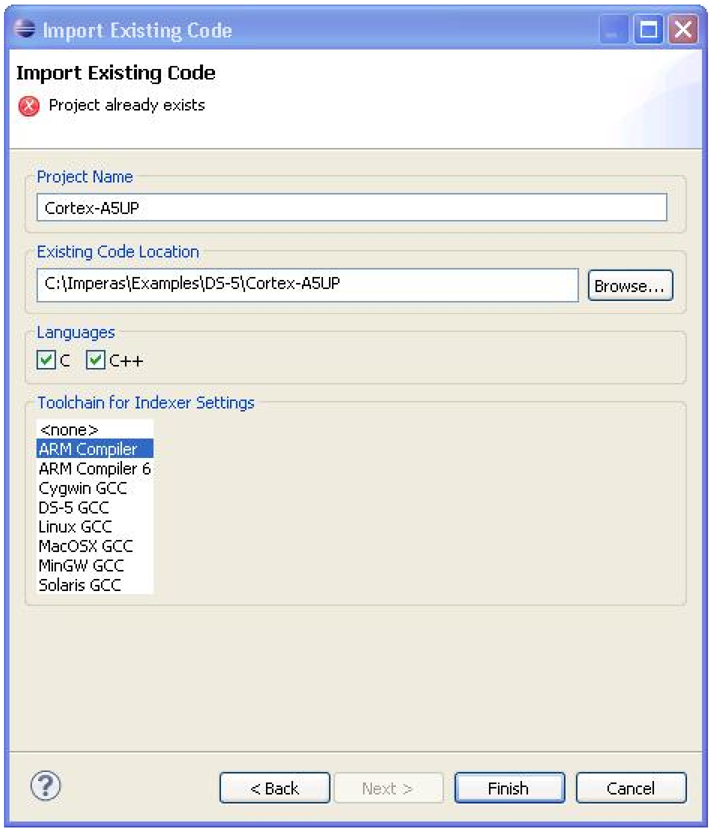
5.1 Importing the example

The example should be imported as a C/C++ ‘Existing Code as Makefile project’

It assumes that you have installed one of the Imperas product packages, Imperas_SDK or Imperas_DEV and so the set of Examples provided with these packages are available.

1. Launch Eclipse for DS-5.
2. Select Import from File menu.
3. Expand the C/C++ tab.
4. Select Existing Code as Makefile Project.

5. Click Next... to locate the OVP Examples directory
6. Select the directory


Import Existing Code

Project already exists

Project Name: Cortex-A5UP

Existing Code Location: C:\Imperas\Examples\DS-5\Cortex-A5UP

Languages: [✓ C] [✓ C++]

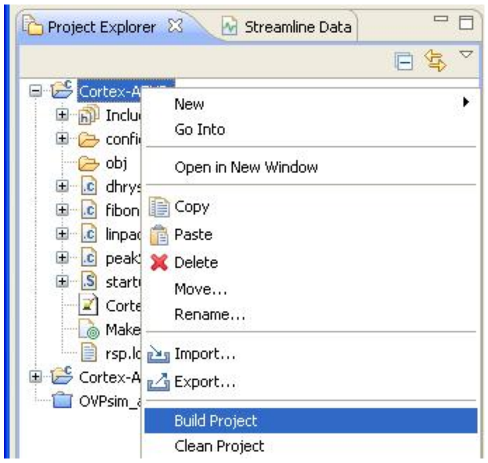
Toolchain for Indexer Settings: <none>, [✓ ARM Compiler], ARM Compiler 6, Cygwin GCC, DS-5 GCC, Linux GCC, MacOSX GCC, MinGW GCC, Solaris GCC

7. Click Finish to close the dialog box and import the project.

The platform execution is started by a launch script, ***imperas_platform_launcher.py***, which is selected from the DS-5 configuration database and requires that the Imperas platform is created using the standard Imperas command line parser. This gives a set of arguments that can be used to define parameters within the platform.

5.2 Building the example

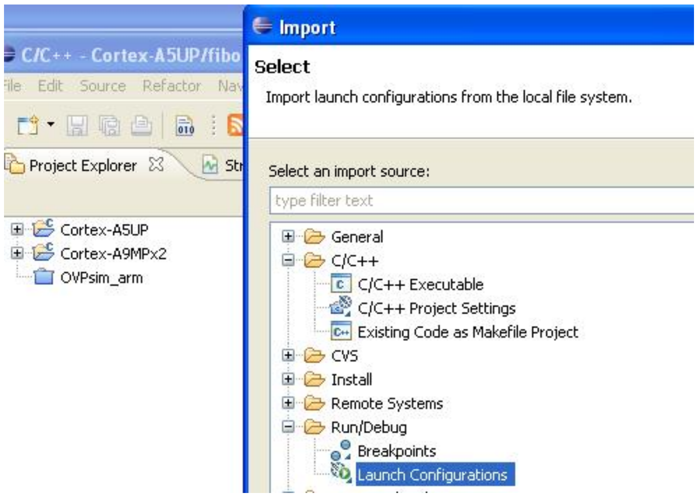
The project can be built using the 'build project' menu entry. This will generate and build the ICM C platform executable and also build the application AXF files. Right click on the project to open the menu …



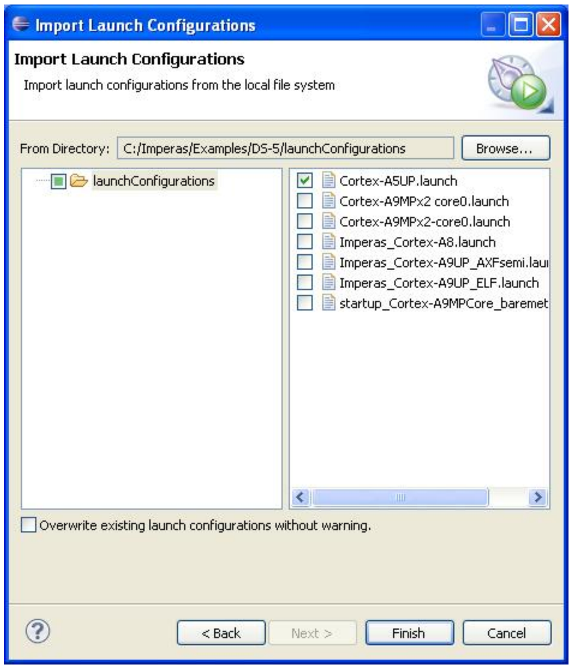
5.3 Importing the debug session launch script

A debug session launch script is available to enable the easy start of a debug session with this platform.

Select Import Run/Debug Launch Configuration



Select the directory containing all the launch configurations and select Cortex-A5UP.launch


Import Launch Configurations
Import Launch Configurations
Import launch configurations from the local file system
From Directory:
Browse...
< Back
Next >
Finish
Cancel

From Directory: C:/Imperas/Examples/DS-5/launchConfigurations

launchConfigurations

- [✓] Cortex-A5UP.launch
- [ ] Cortex-A9MPx2 core0.launch
- [ ] Cortex-A9MPx2-core0.launch
- [ ] Imperas_Cortex-A8.launch
- [ ] Imperas_Cortex-A9UP_AXFsemi.lau
- [ ] Imperas_Cortex-A9UP_ELF.launch
- [ ] startup_Cortex-A9MPCore_baremet

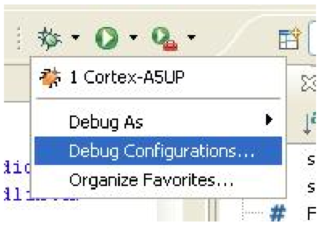
[ ] Overwrite existing launch configurations without warning.

This will create a launch script that we will use in the next section to start the debug session

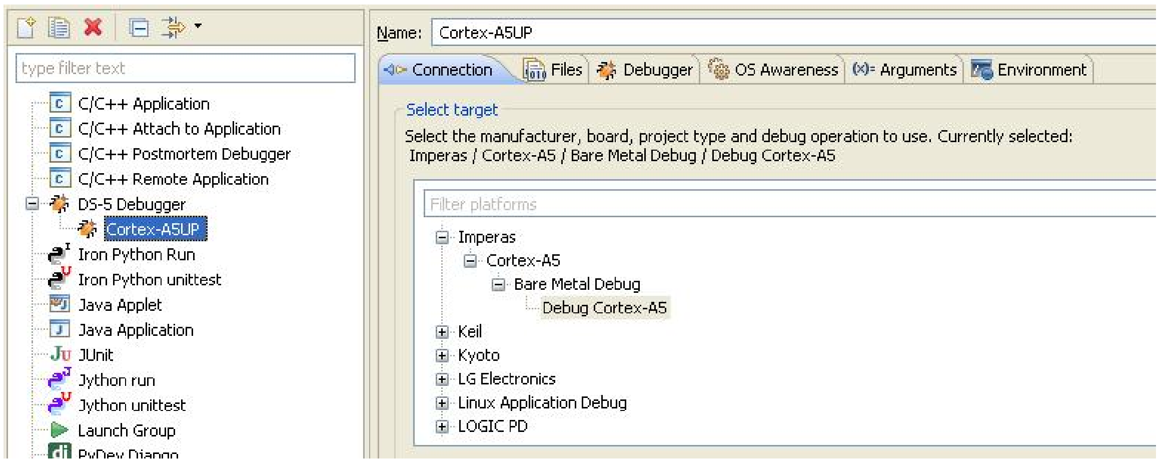
5.4 Starting the debug session

5.4.1 Defining Debug Configuration

Once you have launched a configuration you will see an existing quick launch. Otherwise open the debug configurations menu to create a new debug configuration.



The debug configurations are found below the DS-5 Debugger tab



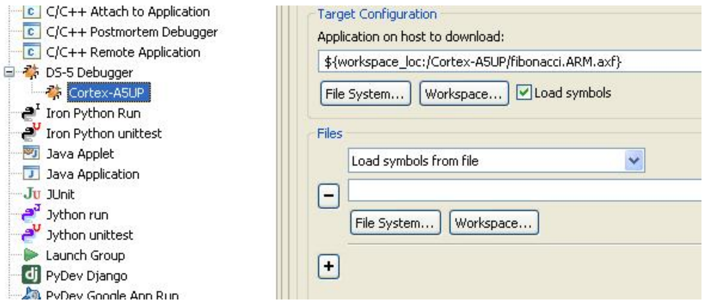
Name: Cortex-A5UP

Select the manufacturer, board, project type and debug operation to use. Currently selected:
Imperas / Cortex-A5 / Bare Metal Debug / Debug Cortex-A5

If we have imported the launch script correctly you will see the ‘Cortex-A5UP’ debug configuration with the Imperas Cortex-A5 Bare Metal Debug connection selected

5.4.2 Selecting Application to Debug

The Files tab allows the selection of the application program to be downloaded and debugged.



Application on host to download: ${workspace_loc:/Cortex-A5UP/fibonacci.ARM.axf}

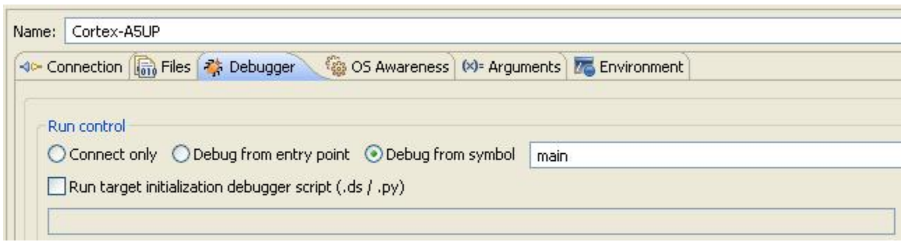
☑ Load symbols

Load symbols from file

Here we are selecting the Fibonacci example from the directory in the workspace.

5.4.3 Defining Run Control and Debug Startup

When we are debugging using a bare metal application we may not have the correct startup code available within the application and so we wish to start execution from the 'entry' symbol. We can do this by specifying the 'Run Control'


Name: Cortex-A5UP

Run control

Connect only Debug from entry point [✓ Debug from symbol] main

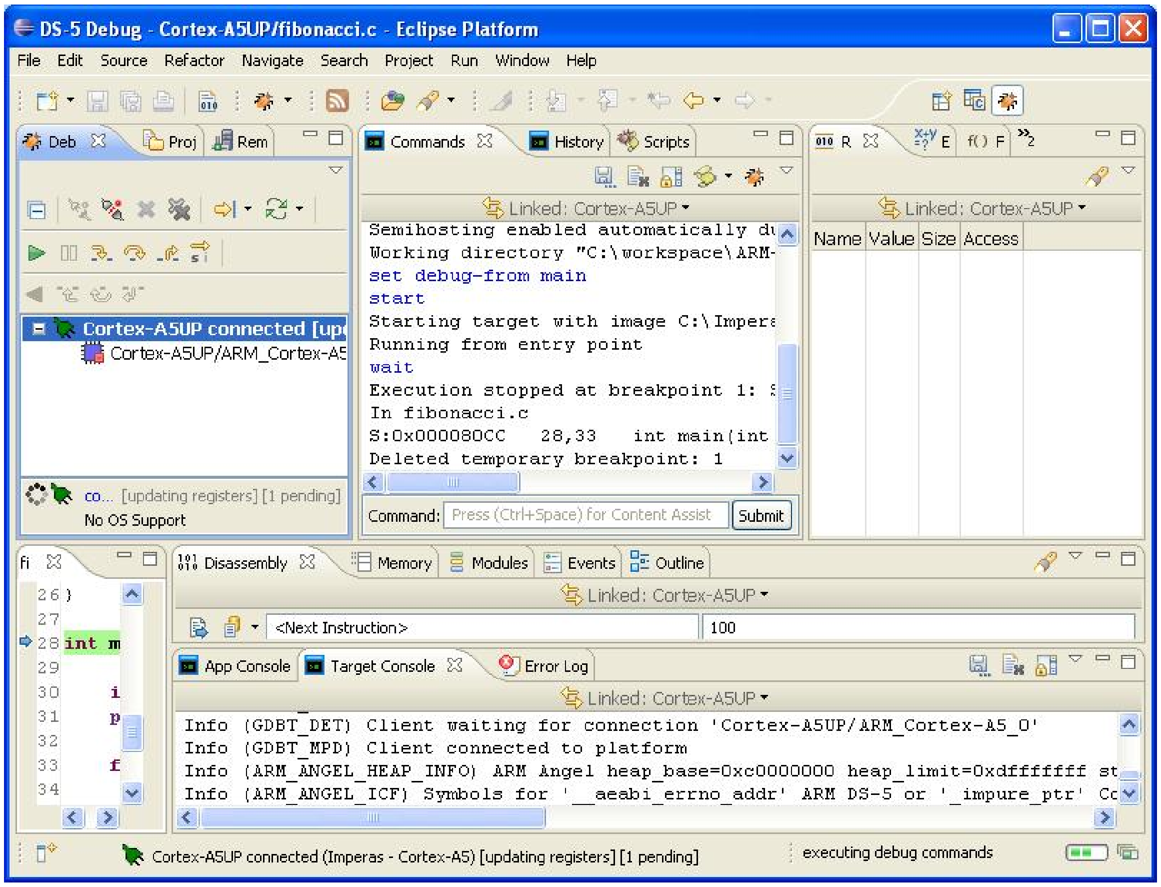
Run target initialization debugger script (.ds / .py)

In this example we are starting our debugging from main. We can choose to start with a connection only, from the application entry point or from a symbol such as main.

5.4.4 The Debug Session

If there is something missing that is required to enable the launch of the debug session the 'Debug' button will not be available. If this is the case, ensure that the configuration database has been correctly loaded and that the project has been built.

When we launch the debug for this configuration the Imperas platform will be launched, the debug session connected and changed to the debug perspective in the DS-5 debugger.


DS-5 Debug - Cortex-A5UP/fibonacci.c - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Deb Proj Rem
Cortex-A5UP connected [upd
Cortex-A5UP/ARM_Cortex-A5
co... [updating registers] [1 pending]
No OS Support
Commands History Scripts
Linked: Cortex-A5UP
Command: Press (Ctrl+Space) for Content Assist Submit
R E f() F
Name Value Size Access
fi
Disassembly Memory Modules Events Outline
<Next Instruction> 100
App Console Target Console Error Log
Cortex-A5UP connected (Imperas - Cortex-A5) [updating registers] [1 pending]
executing debug commands

```
Semihosting enabled automatically du
Working directory "C:\workspace\ARM-
set debug-from main
start
Starting target with image C:\Impera
Running from entry point
wait
Execution stopped at breakpoint 1: S
In fibonacci.c
S:0x000080CC   28,33   int main(int
Deleted temporary breakpoint: 1
```

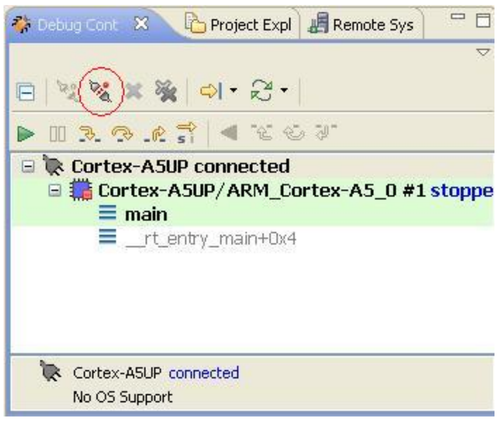
```
26 }
27
28 int m
29
30     i
31     p
32
33     f
34
```

```
Info (GDBT_DET) Client waiting for connection 'Cortex-A5UP/ARM_Cortex-A5_0'
Info (GDBT_MPD) Client connected to platform
Info (ARM_ANGEL_HEAP_INFO) ARM Angel heap_base=0xc0000000 heap_limit=0xdfffffff st
Info (ARM_ANGEL_ICF) Symbols for '__aeabi_errno_addr' ARM DS-5 or '_impure_ptr' Cc
```

5.5 Closing the debug session

5.5.1 Disconnecting the Platform

Once you have finished the debug session you can disconnect the platform using the disconnect button

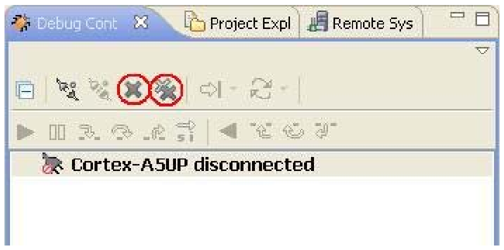


Cortex-A5UP connected
Cortex-A5UP/ARM_Cortex-A5_0 #1 stoppe
main
__rt_entry_main+0x4

Cortex-A5UP connected
No OS Support

5.5.2 Cleaning up Closed Connections

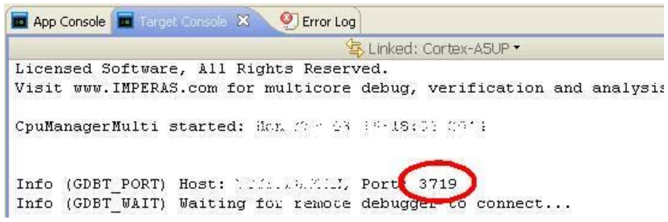
The disconnected sessions are shown as such in the window. These should be cleaned up by selecting either the clean up single (which remove just the selected disconnected session) or clean up multiple (which will remove all disconnected sessions) icons.



Cortex-A5UP disconnected

5.5.3 Forcing Rogue Platform Finish

The utility program IMPERAS_HOME/bin/IMPERAS_ARCH/remoteUtility.exe can be used with the port number reported when a platform started, as shown in the Target Console



```
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis

CpuManagerMulti started: [redacted]


Info (GDBT_PORT) Host: [redacted], Port: 3719
Info (GDBT_WAIT) Waiting for remote debugger to connect...
```

to terminate the platform.

```
remoteUtility.exe --port 3791 --finish
```

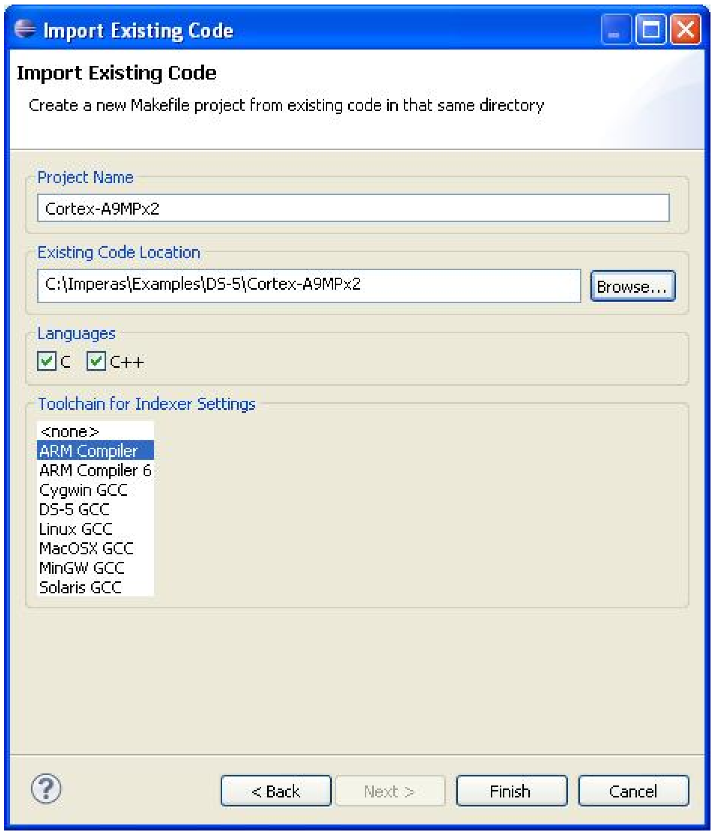
6 Installing the Imperas Cortex-A9MPx2 Multi-Core Example

6.1 Importing the example

The example should be imported as a C/C++ 'Existing Code as Makefile project'

The following section will illustrate how an existing OVP Demo or Example may be imported into ARM DS-5 Eclipse as a project. It assumes that you have installed one of the Imperas product packages, Imperas_SDK or Imperas_DEV and so the set of Demos provided with these packages are available.

1. Launch Eclipse.
2. Select Import from File menu.
3. Expand the C/C++ tab.
4. Select Existing Code as Makefile Project.
5. Click Next... to locate the OVP Demo directory
6. Select the directory



Import Existing Code

Import Existing Code

Create a new Makefile project from existing code in that same directory

Project Name: Cortex-A9MPx2

Existing Code Location: C:\Imperas\Examples\DS-5\Cortex-A9MPx2

Languages: ✓ C ✓ C++

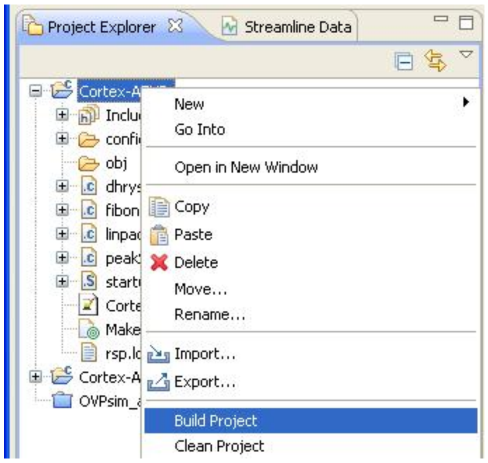
Toolchain for Indexer Settings: <none>, [✓ ARM Compiler], ARM Compiler 6, Cygwin GCC, DS-5 GCC, Linux GCC, MacOSX GCC, MinGW GCC, Solaris GCC

7. Click Finish to close the dialog box and import the project.

The platform execution is started by a launch script, ***imperas_platform_launcher.py***, which is selected from the DS-5 configuration database and requires that the Imperas platform is created using the standard Imperas command line parser. This gives a set of arguments that can be used to define parameters within the platform.

6.2 Building the example

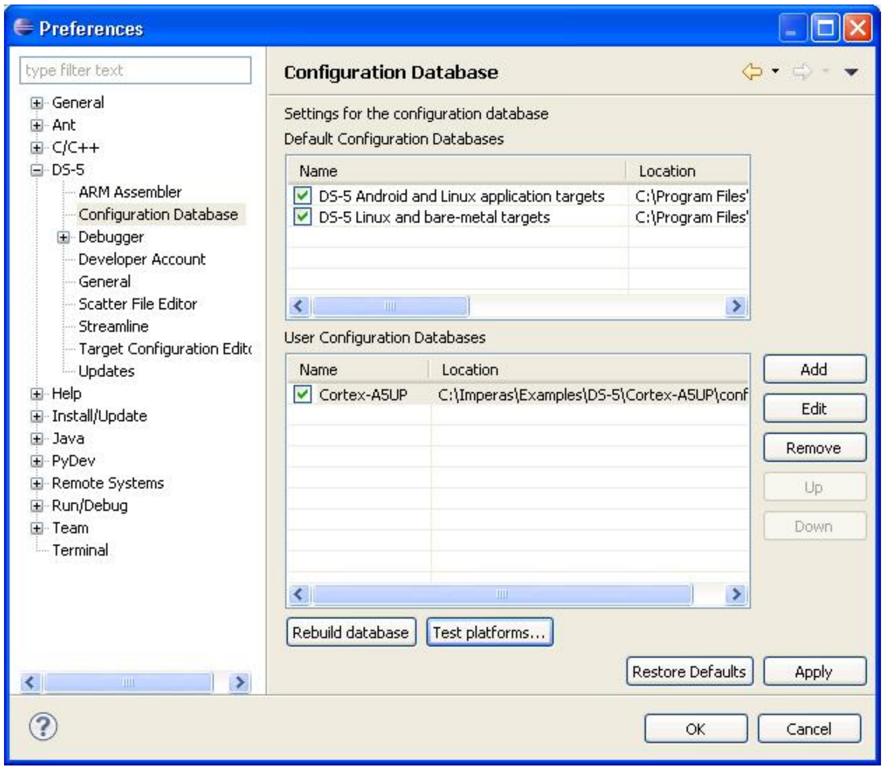
The project can be built using the ‘build project’ menu entry. This will generate and build the ICM C platform executable and also build the application AXF files.



6.3 Importing the configuration database

The configuration database that contains the information for this project is found in the same example directory. This contains both the platform files and the python script used to invoke a bare metal platform.

Initially the standard built-in configurations are available. We will add new configuration databases for each project separately. These may have been combined into a single custom database.



Configuration Database

Settings for the configuration database

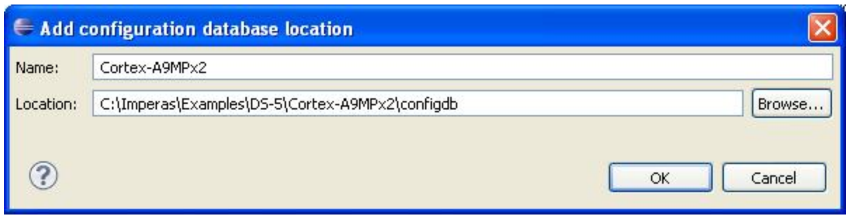
Default Configuration Databases

Name	Location
✓ DS-5 Android and Linux application targets	C:\Program Files'
✓ DS-5 Linux and bare-metal targets	C:\Program Files'

User Configuration Databases

Name	Location
✓ Cortex-A5UP	C:\Imperas\Examples\DS-5\Cortex-A5UP\conf

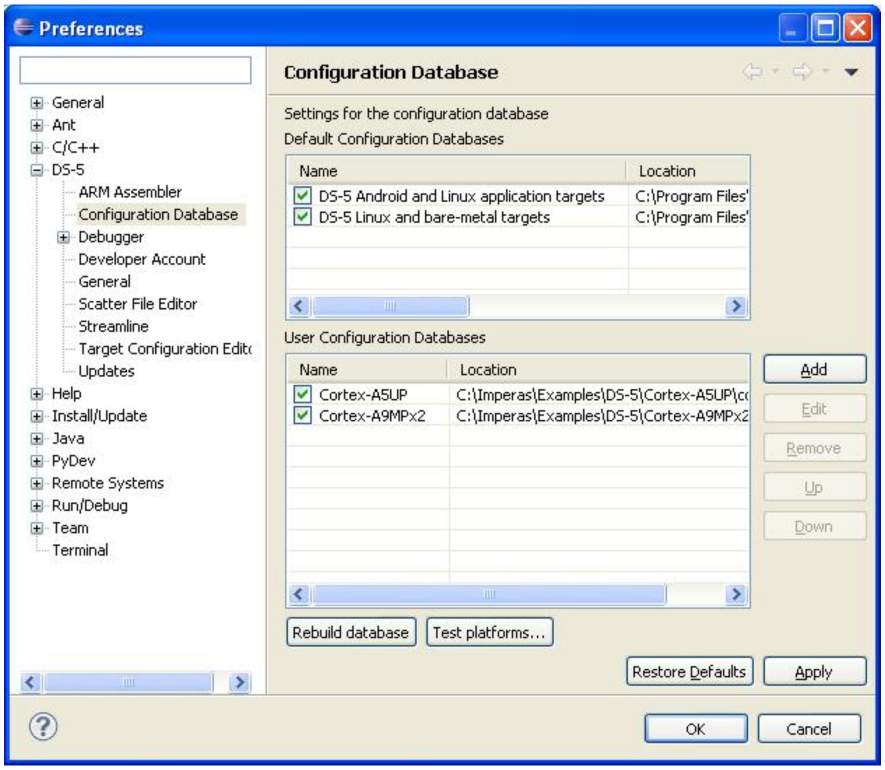
Select the 'add' button to bring in the configuration database for this example.



Name: Cortex-A9MPx2

Location: C:\Imperas\Examples\DS-5\Cortex-A9MPx2\configdb

Once the database has been selected we need to rebuild the database for the changes to take effect.



Configuration Database

Settings for the configuration database

Default Configuration Databases

Name	Location
✓ DS-5 Android and Linux application targets	C:\Program Files'
✓ DS-5 Linux and bare-metal targets	C:\Program Files'

User Configuration Databases

Name	Location
✓ Cortex-A5UP	C:\Imperas\Examples\DS-5\Cortex-A5UP\c
✓ Cortex-A9MPx2	C:\Imperas\Examples\DS-5\Cortex-A9MPx2

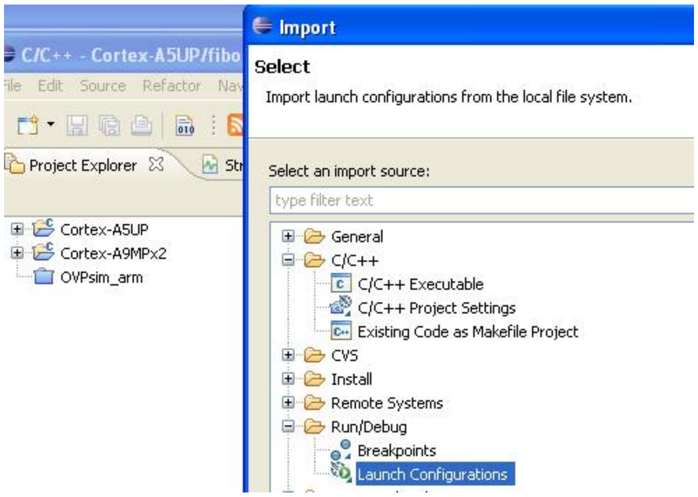
Once the database has been re-built we can test the Imperas platforms we have added by selecting the ‘test platforms’ button and then clicking on the tabs associated with the Imperas database.

The testing of the platform checks that the project and platform definitions are correct and that the RDDI library is correctly loaded and valid.

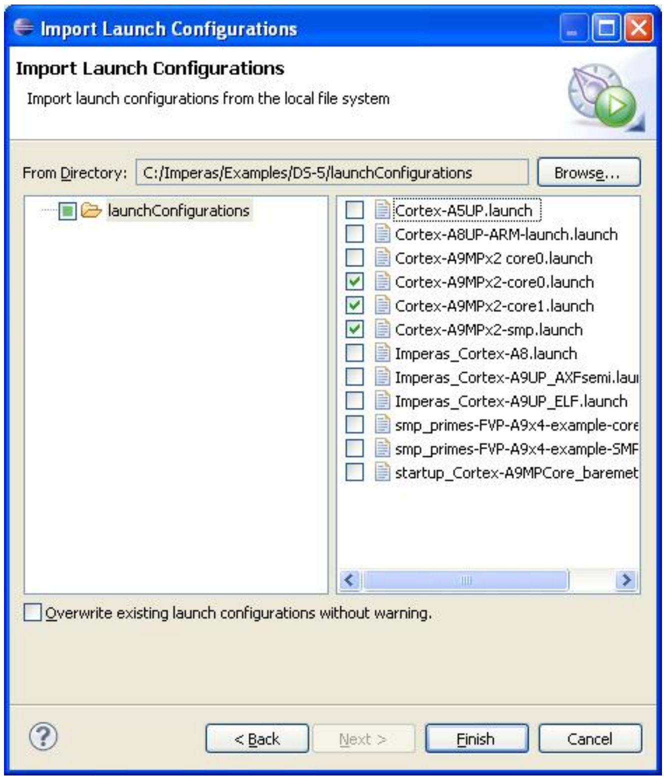
6.4 Importing the debugger session launch script

A debugger session launch script is available to enable the easy start of a debug session with this platform.

Select Import Run/Debug Launch Configuration



Select the directory containing all the launch configurations and select all the Cortex-A9MPx2 launch files.


Import Launch Configurations
Import Launch Configurations
Import launch configurations from the local file system
Browse...
launchConfigurations
< Back
Next >
Finish
Cancel

From Directory: C:/Imperas/Examples/DS-5/launchConfigurations

- [ ] Cortex-A5UP.launch
- [ ] Cortex-A8UP-ARM-launch.launch
- [ ] Cortex-A9MPx2 core0.launch
- [✓] Cortex-A9MPx2-core0.launch
- [✓] Cortex-A9MPx2-core1.launch
- [✓] Cortex-A9MPx2-smp.launch
- [ ] Imperas_Cortex-A8.launch
- [ ] Imperas_Cortex-A9UP_AXFsemi.lau
- [ ] Imperas_Cortex-A9UP_ELF.launch
- [ ] smp_primes-FVP-A9x4-example-core
- [ ] smp_primes-FVP-A9x4-example-SMP
- [ ] startup_Cortex-A9MPCore_baremet

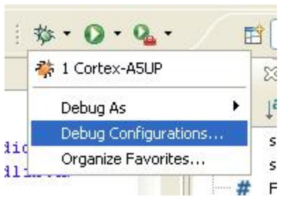
- [ ] Overwrite existing launch configurations without warning.

This will create a launch script that we will use in the next section to start the debug session

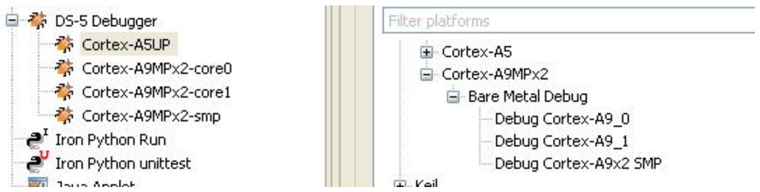
6.5 Starting the debug session

6.5.1 Defining Debug Configuration

Once you have launched a configuration you will see an existing quick launch. Otherwise open the debug configurations menu



The debug configurations are found below the DS-5 Debugger tab



If we have imported the launch script correctly you will see several ‘Cortex-A9MPx2’ debug configurations that use the Imperas Cortex-A9MPx2 Bare Metal Debug connections. Each debug configuration selects a different connections to allow applications on individual cores or on both cores operating as an SMP cluster to be debugged.

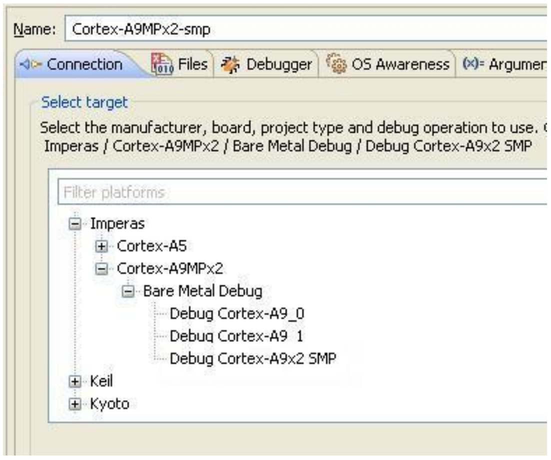
6.6 Attaching to a single core of an MP core

NOTE

When attaching to a single core in a multi core platform DS-5 only loads a program onto the core to which it is attached. The OVP/Imperas multi-core simulators will run all processors within a platform. Furthermore, the OVP/Imperas simulators will generate a warning and stop the simulation under these bare metal platform default conditions if a processor is running from memory that has not been initialized. The independent initialization of the un-attached core is provided in these examples.

6.6.1 Selecting the core to debug

The core to debug is selected using one of the possible target connections available under the connection tab.


Name: Cortex-A9MPx2-smp

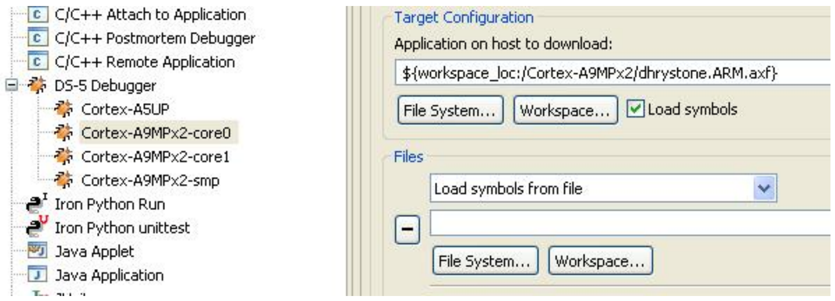
Select target

Select the manufacturer, board, project type and debug operation to use.
Imperas / Cortex-A9MPx2 / Bare Metal Debug / Debug Cortex-A9x2 SMP

```
Imperas
  Cortex-A5
  Cortex-A9MPx2
    Bare Metal Debug
      Debug Cortex-A9_0
      Debug Cortex-A9_1
      Debug Cortex-A9x2 SMP
Keil
Kyoto
```

`

The Files allows the selection of the application program to be downloaded and debugged on the selected core.


```
C/C++ Attach to Application
C/C++ Postmortem Debugger
C/C++ Remote Application
DS-5 Debugger
  Cortex-A5UP
  Cortex-A9MPx2-core0
  Cortex-A9MPx2-core1
  Cortex-A9MPx2-smp
Iron Python Run
Iron Python unittest
Java Applet
Java Application
```

Target Configuration

Application on host to download:

${workspace_loc:/Cortex-A9MPx2/dhrystone.ARM.axf}

☑ Load symbols

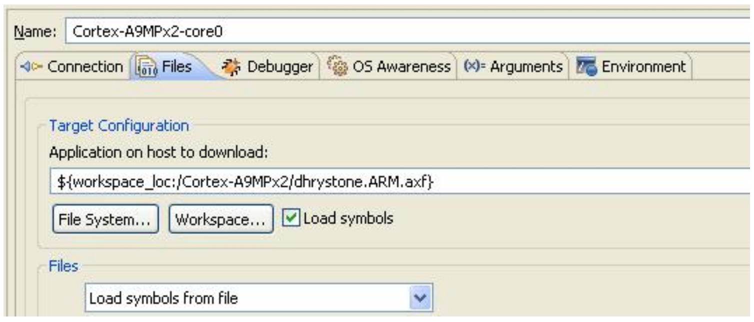
Files

Load symbols from file

Here we are selecting the Fibonacci example to execute.

6.6.2 Selecting Application to Debug

The Files tab allows the selection of the application program to be downloaded and debugged.


Name: Cortex-A9MPx2-core0

Target Configuration

Application on host to download:

${workspace_loc:/Cortex-A9MPx2/dhrystone.ARM.axf}

☑ Load symbols

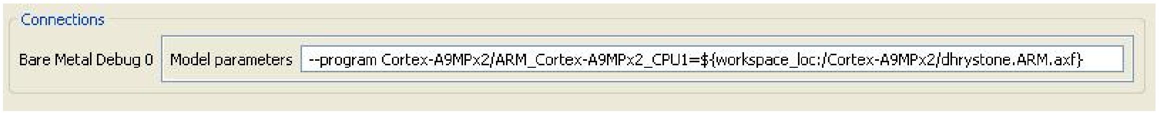
Files

Load symbols from file

Here we are selecting the Dhrystones example from the directory in the workspace.

6.6.3 Adding program to execute on the 'other' un-attached core(s)

We also add an additional argument to the platform being simulated so that a default program is loaded onto the other cores that we are not debugging and are allowing to free run.



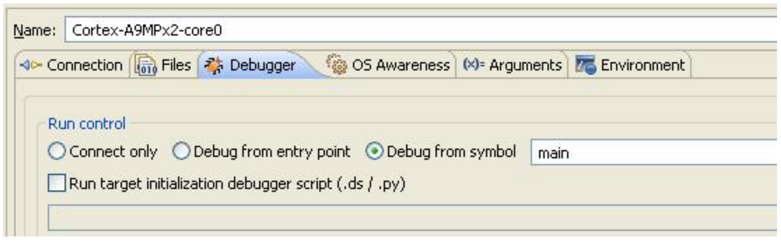
Connections

Bare Metal Debug 0 | Model parameters: --program Cortex-A9MPx2/ARM_Cortex-A9MPx2_CPU1=${workspace_loc:/Cortex-A9MPx2/dhrystone.ARM.axf}

The --program argument is used to load the Dhrystone application onto the core identified by Cortex-A9MPx2/ARM_Cortex-A9MPx2_CPU1

6.6.4 Defining Run Control and Debug Startup

When we are debugging using a bare metal application we may not have the correct startup code available within the application and so we wish to start execution from the 'entry' symbol. We can do this by specifying the 'Run Control'



Name: Cortex-A9MPx2-core0

Run control

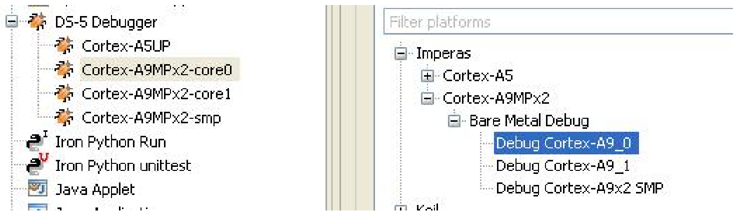
Connect only · Debug from entry point · [✓ Debug from symbol] main

Run target initialization debugger script (.ds / .py)

In this example we are starting our debugging from main. We can choose to start with a connection only, from the application entry point or from a symbol such as main.

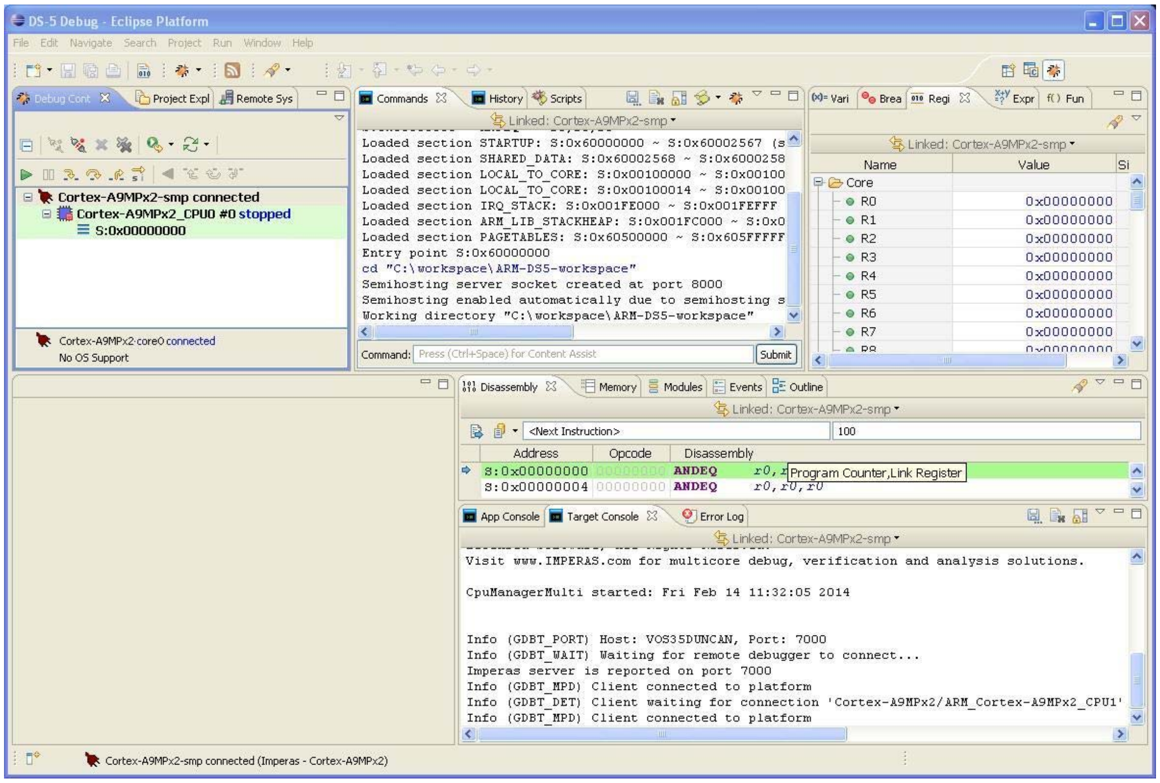
6.6.5 Launching the debug session on core 0

Create or select the debug session required in the Debug Configurations window.



If there is something missing that is required to enable the launch of the debug session the 'Debug' button will not be available. If this is the case, ensure that the configuration database has been correctly loaded and that the project has been built.

When we launch the debug for this configuration the Imperas platform will be launched, the debug session connected and changed to the debug perspective in the DS-5 debugger.


DS-5 Debug - Eclipse Platform
Debug Cont
Project Expl
Remote Sys
Cortex-A9MPx2-smp connected
Cortex-A9MPx2_CPU0 #0 stopped
S:0x00000000
Cortex-A9MPx2-core0 connected
No OS Support
Commands
History
Scripts
Linked: Cortex-A9MPx2-smp
Command:
Press (Ctrl+Space) for Content Assist
Submit
Disassembly
Memory
Modules
Events
Outline
App Console
Target Console
Error Log
Program Counter,Link Register
Cortex-A9MPx2-smp connected (Imperas - Cortex-A9MPx2)

```
Loaded section STARTUP: S:0x60000000 ~ S:0x60002567 (s
Loaded section SHARED_DATA: S:0x60002568 ~ S:0x6000258
Loaded section LOCAL_TO_CORE: S:0x00100000 ~ S:0x00100
Loaded section LOCAL_TO_CORE: S:0x00100014 ~ S:0x00100
Loaded section IRQ_STACK: S:0x001FE000 ~ S:0x001FEFFF
Loaded section ARM_LIB_STACKHEAP: S:0x001FC000 ~ S:0x0
Loaded section PAGETABLES: S:0x60500000 ~ S:0x605FFFFF
Entry point S:0x60000000
cd "C:\workspace\ARM-DS5-workspace"
Semihosting server socket created at port 8000
Semihosting enabled automatically due to semihosting s
Working directory "C:\workspace\ARM-DS5-workspace"
```

Name	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000

<Next Instruction> 100

```
   Address       Opcode     Disassembly
S:0x00000000  00000000   ANDEQ     r0,r
S:0x00000004  00000000   ANDEQ     r0,r0,r0
```

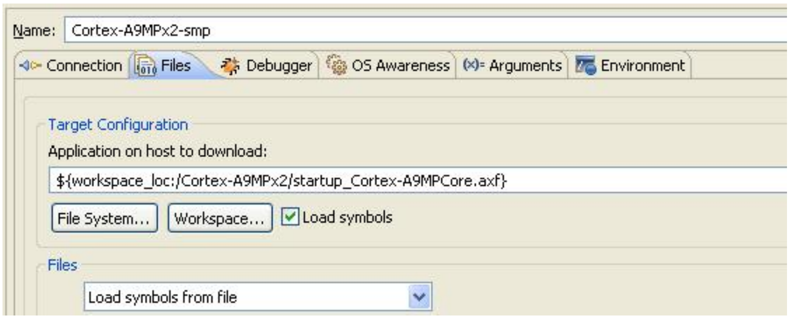
```
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Fri Feb 14 11:32:05 2014


Info (GDBT_PORT) Host: VOS35DUNCAN, Port: 7000
Info (GDBT_WAIT) Waiting for remote debugger to connect...
Imperas server is reported on port 7000
Info (GDBT_MPD) Client connected to platform
Info (GDBT_DET) Client waiting for connection 'Cortex-A9MPx2/ARM_Cortex-A9MPx2_CPU1'
Info (GDBT_MPD) Client connected to platform
```

We can choose to start with a connection only, at main or at another symbol.

6.7 Attaching as a multi-core SMP cluster

6.7.1 Selecting Application to Debug

The Files tab allows the selection of the application program to be downloaded and debugged.



Name: Cortex-A9MPx2-smp

Target Configuration

Application on host to download:

${workspace_loc:/Cortex-A9MPx2/startup_Cortex-A9MPCore.axf}

☑ Load symbols

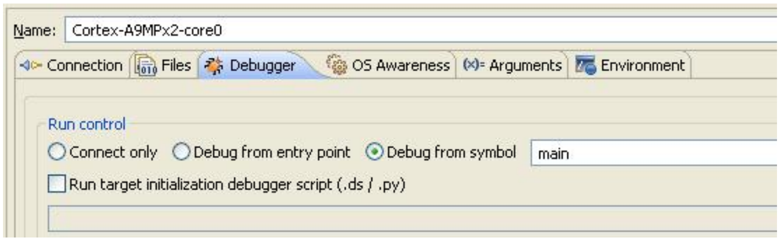
Files

Load symbols from file

Here we are selecting an SMP startup code example.

6.7.2 Defining Run Control and Debug Startup

When we are debugging using a bare metal application we may not have the correct startup code available within the application and so we wish to start execution from the ‘entry’ symbol. We can do this by specifying the ‘Run Control’



Name: Cortex-A9MPx2-core0

Run control

Connect only   Debug from entry point   [✓ Debug from symbol] main

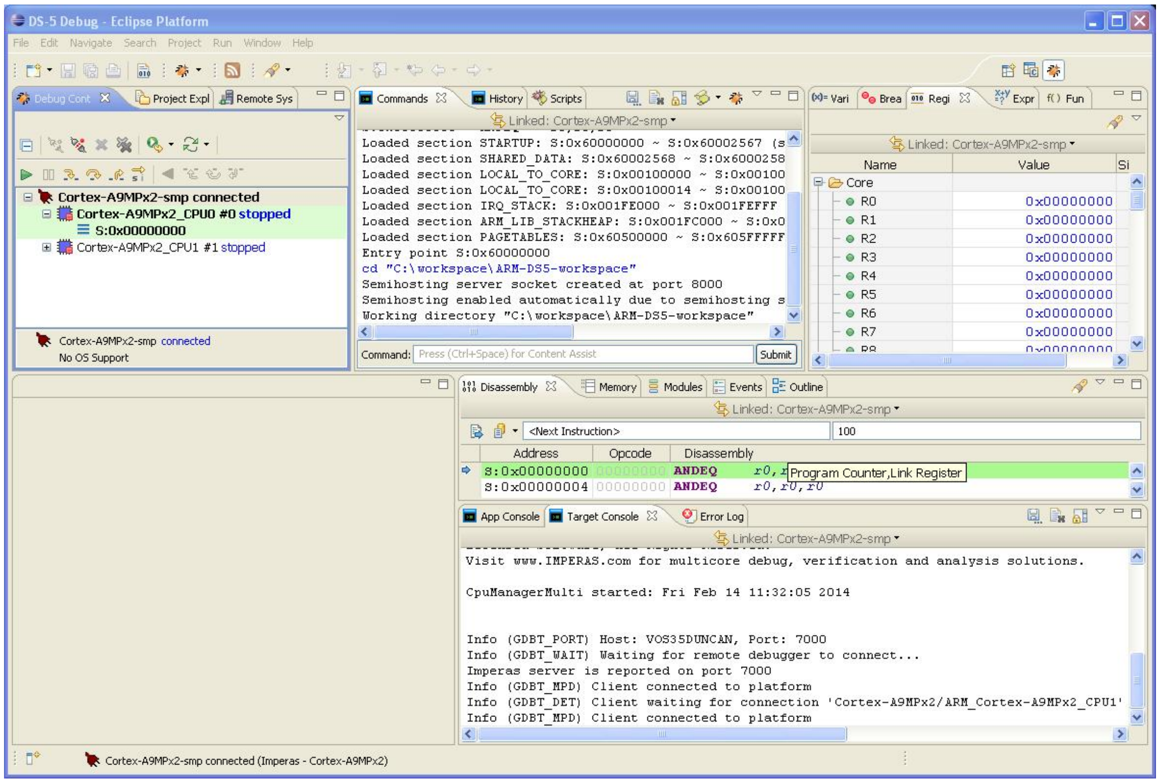
☐ Run target initialization debugger script (.ds / .py)

In this example we are starting our debugging from main. We can choose to start with a connection only, from the application entry point or from a symbol such as main.

6.7.3 The Debug Session

If there is something missing that is required to enable the launch of the debug session the ‘Debug’ button will not be available. If this is the case, ensure that the configuration database has been correctly loaded and that the project has been built.

When we launch the debug for this configuration the Imperas platform will be launched, the debug session connected and changed to the debug perspective in the DS-5 debugger.



Cortex-A9MPx2-smp connected
Cortex-A9MPx2_CPU0 #0 stopped
S:0x00000000
Cortex-A9MPx2_CPU1 #1 stopped

Cortex-A9MPx2-smp connected
No OS Support

```
Loaded section STARTUP: S:0x60000000 ~ S:0x60002567 (s
Loaded section SHARED_DATA: S:0x60002568 ~ S:0x6000258
Loaded section LOCAL_TO_CORE: S:0x00100000 ~ S:0x00100
Loaded section LOCAL_TO_CORE: S:0x00100014 ~ S:0x00100
Loaded section IRQ_STACK: S:0x001FE000 ~ S:0x001FEFFF
Loaded section ARM_LIB_STACKHEAP: S:0x001FC000 ~ S:0x0
Loaded section PAGETABLES: S:0x60500000 ~ S:0x605FFFFF
Entry point S:0x60000000
cd "C:\workspace\ARM-DS5-workspace"
Semihosting server socket created at port 8000
Semihosting enabled automatically due to semihosting s
Working directory "C:\workspace\ARM-DS5-workspace"
```

Command: Press (Ctrl+Space) for Content Assist

Name	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000

<Next Instruction> 100

Address	Opcode	Disassembly
S:0x00000000	00000000	ANDEQ r0,r
S:0x00000004	00000000	ANDEQ r0,r0,r0

```
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Fri Feb 14 11:32:05 2014


Info (GDBT_PORT) Host: VOS35DUNCAN, Port: 7000
Info (GDBT_WAIT) Waiting for remote debugger to connect...
Imperas server is reported on port 7000
Info (GDBT_MPD) Client connected to platform
Info (GDBT_DET) Client waiting for connection 'Cortex-A9MPx2/ARM_Cortex-A9MPx2_CPU1'
Info (GDBT_MPD) Client connected to platform
```

Cortex-A9MPx2-smp connected (Imperas - Cortex-A9MPx2)

We can choose to start with a connection only, at main or at another symbol.

7 Adding your own platforms

7.1 Example DS-5 Configuration Database

There are two examples of ARM DS-5 configuration databases that works with Imperas/OVP platforms provided at Examples/Interfaces/ARM_DS-5/ as Cortex-A5UP/configdb and Cortex-A9MPx2/configdb.

The example databases contains two board support packages

1. Cortex-A5UP is a bare metal Cortex-A5UP platform
2. Cortex-A9MPx2 is a bare metal dual core Cortex-A9MP that can be used attached to one of the individual cores or attached as an SMP cluster.

7.2 Creating a New DS-5 Configuration Database

7.2.1 Creating the board support files

Copy the configdb directory from one of the examples into the example with which you wish to use ARM DS-5 for debugging. Correct the name, if required, of the directories and files to correctly represent the names of the board that you wish to debug.

Select one of the board support directories in the provided example and copy it locally, renaming as required to match your platform requirements.

7.2.1.1 Platform file (a5up_bare_metal.xml)

This file defines the connections available on the platform and the interface library that should be used.

Changes that can be made to the example

The ***RDDI Type*** is the library shared object that is loaded by DS-5 and provides the interface between the RDDI API and the platform.

```
    <rddi type="ARM_RDDI"/>
```

The should be a ***Device Entry*** for each of the processor cores that are in the platform. These must have a unique name.

```
        <DEVICE Number="1">
            <ConnectionID>ARM_Cortex-A5UP</ConnectionID>
            <CP15ID>CP15_ID</CP15ID>
        </DEVICE>
```

7.2.1.2 Project file (project_types.xml)

The project file provides a list of project types. Each of the project types represents a different activity that can be performed on the platform, for example connecting and debugging a single core or perhaps connecting and debugging as an SMP cluster.

Within each project definition there is a ***execution_environment*** section that contains ***param*** and ***config_file*** sections that identify the platform file, for example a8_bare_metal.xml that should be used to define the board.

```
            <execution_environment id="bare_metal">
                <param default="CDB://a5_bare_metal.xml" id="config_file" type="string"
visible="false"/>
                <config_file>CDB://a5_bare_metal.xml</config_file>
```

The project definition also contains a ***setup_script*** entry that defines the way in which the virtual platform simulation will be started. In the case below the Imperas Platform launcher Python script is used and passed one argument, which corresponds to the platform name, Cortex-A5UP. Please note the relationship between the directory containing the configuration database files and the common launchScripts directory.

```
        <setup_script>
           <name>CDB://../../../../../launchScripts/imperas_platform_launcher.py</name>
           <arguments>Cortex-A5UP</arguments>
        </setup_script>
```

The launch script is referenced relative to the **CDB** built in definition.

The final entry in the project file execution_environment is the ***activity*** section which contains ***core*** information. The connection_id matches the name of the core within the platform and the core_definition is used to define all the registers contained within the processor core. This name must match a core definition from the ***cores*** directory that can be a new custom device defined in the new configuration database or be a standard processor core from the ARM DS-5 database.

```
            <core connection_id="cpu0" core_definition="Cortex-A5UP"/>
```

7.3 Adding Configuration Data Base to DS-5

Once you have created a board support package for your new platform.

The board support package that you have defined by the files referenced in the previous sections may be added into the configuration database shipped with DS-5. The database is found within the ARM DS-5 installation at <install root>/sw/debugger/configdb.

Each board support package provides:
- Boards: information about all the platforms you want to connect to
- Cores: if you've added your own custom cores & register definitions,
- Scripts: launch script for your target

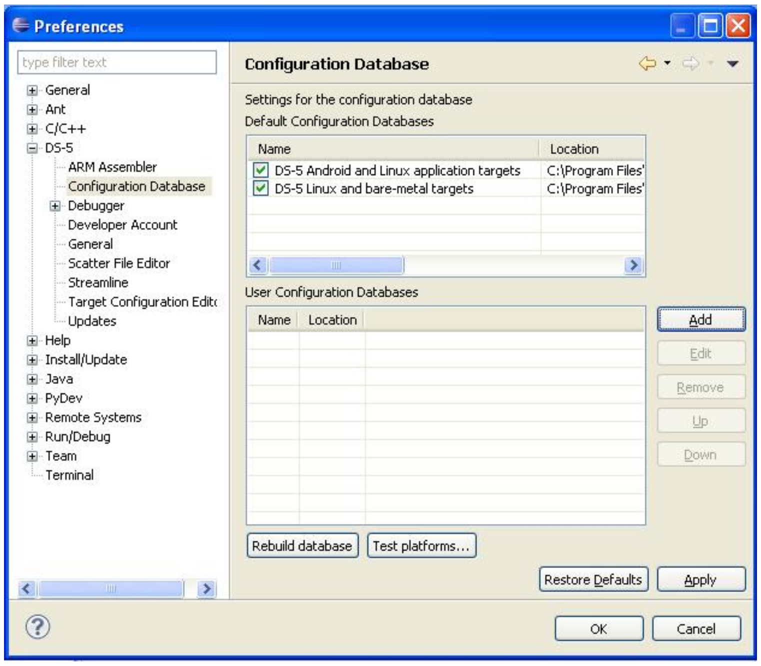
The procedure for adding a user database to DS-5 in the eclipse preferences can be found on the ARM website at the following link:

http://infocenter.arm.com/help/topic/com.arm.doc.dui0446p/deb1359985233115.html

The information contained in the above link is also provided in the following paragraphs.

1. Launch Eclipse for DS-5.
2. Select Preferences from Windows menu.
3. Expand the DS-5 configuration group.
4. Select Configuration Database.
5. Click Add... to locate the new database:

Initially the standard built-in configurations are available. We will add new configuration databases for each project separately. These may have been combined into a single custom database.


Preferences

Configuration Database

Settings for the configuration database

Default Configuration Databases

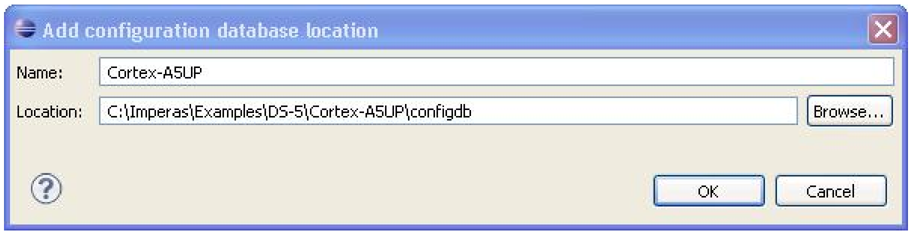
Name	Location
✓ DS-5 Android and Linux application targets	C:\Program Files'
✓ DS-5 Linux and bare-metal targets	C:\Program Files'

User Configuration Databases

Name	Location

Select the ‘add’ button to select the new configuration database to be imported.

6. Select the entire directory.


Add configuration database location

Name: Cortex-A5UP

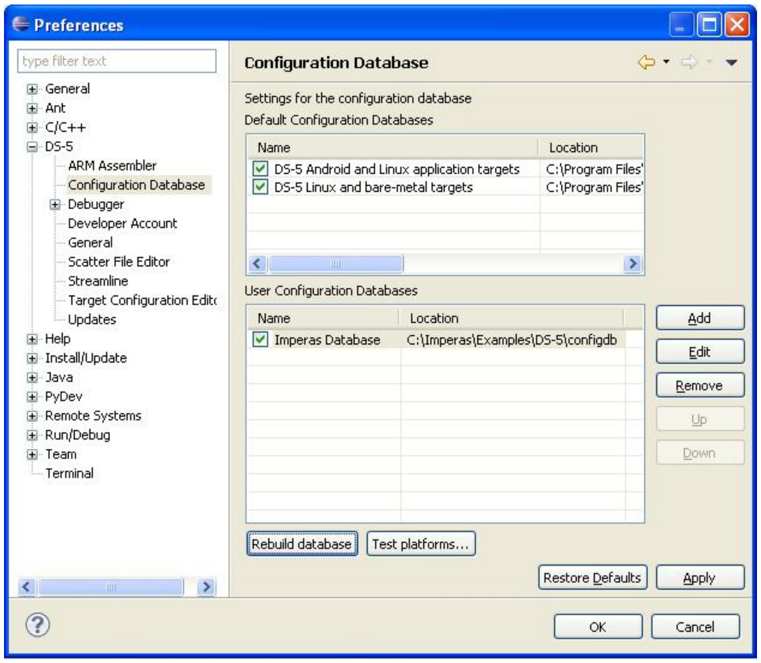
Location: C:\Imperas\Examples\DS-5\Cortex-A5UP\configdb

7. Click OK to close the dialog box.
8. Position the new database:
    c. Select the new database.
    d. Click Up or Down as required.

    Note
    DS-5 provides built-in databases containing a default set of target configurations. You can enable or disable these but not delete them.

9. Click Rebuild database…

Once the database has been selected we need to rebuild the database for the changes to take effect.



Settings for the configuration database

Default Configuration Databases

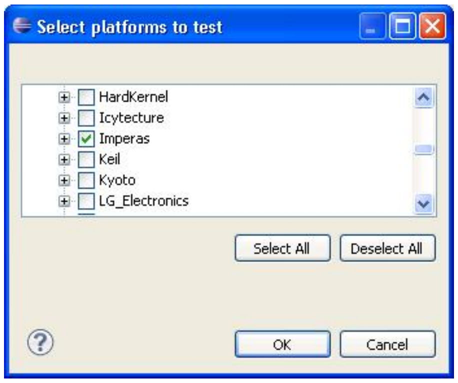
Name	Location
✓ DS-5 Android and Linux application targets	C:\Program Files'
✓ DS-5 Linux and bare-metal targets	C:\Program Files'

User Configuration Databases

Name	Location
✓ Imperas Database	C:\Imperas\Examples\DS-5\configdb

10. Click Test Platforms…

Once the database has been re-built we can test the Imperas platforms we have added by selecting the ‘test platforms’ button and then clicking on the tabs associated with the Imperas database.



The testing of the platform checks that the project and platform definitions are correct and consistent.

11. Click OK to close the dialog box and save the settings.

Whenever changes are made to the files within the configuration database you must repeat steps 9 and 10 to update DS-5

8 DS-5 Configuration Database Overview

The ARM DS-5 configuration database contains details of how to start up and connect to a platform. Additional configuration databases may be loaded into DS-5. This section provides the details required to create a new configuration database ready to be loaded into ARM DS-5.

8.1 Directory structure

The following is the structure of a configuration database that must be created so that a new platform can be loaded into ARM DS-5

```
configDB/Boards/Imperas/<board name>/<board definition>.xml
configDB/Boards/Imperas/<board name>/project_types.xml
configDB/Boards/Cores/<core definition>.xml
configDB/Scripts/<launch Script>
```

8.1.1 Board Definition directory

The directory *Boards/Imperas/<board name>* contains two files.
A board definition file that defines the board contents, for example the processor cores that can be debugged.
A project types file that defines the ways in which ARM DS-5 can connect to this board and the types of debug tasks that can be performed, for example bare metal debug. This also contains the details of how the platform is launched. Common launch scripts are referenced relative to this configuration data base entry.

8.1.2 Launch Scripts directory

The directory *Scripts* contains one or more scripts that are referenced from the project types file and used to start the execution of a simulation by launching a platform executable. The script to launch bare metal platforms is written in python. The arguments required for the platform to be correctly invoked are passed from the board specification.

8.1.2.1 Platform Launch script (imperas_platform_launcher.py)

The platform execution is started by a launch script, ***imperas_platform_launcher.py***, which is selected from the DS-5 configuration database and requires that the Imperas platform is created using the standard Imperas command line parser. This gives a set of arguments that can be used to define parameters within the platform.

8.1.3 Cores directory

This directory contains files to define the registers of a processor core that does not already appear as part of the standard ARM cores of DS-5. The cores are referenced from an entry in the project types directory.

8.2 Debug Session Configuration Files Contents

The configuration files use XML to define the platform processor contents and available debug connections.

8.2.1 Board Definition File

The board definition file defines:

1. The processor(s) that are found on the board. This is the device types and their instance names.
2. The interface library shared object that should be loaded to talk to the board
3. It may also define multiple cores used and debugged as an SMP cluster

```
<?xml version="1.0"?>
<!--Copyright (C) 2009-2012 ARM Limited. All rights reserved.-->
<?RVConfigUtility MajorVersion = "0" MinorVersion = "0" PatchVersion = "0"?>
<RVConfigUtility>
    <rddi type="ARM_RDDI"/>
    <RDDICADI>
        <DEVICE Number="1">
            <ConnectionID>ARM_Cortex-A9_0</ConnectionID>
            <CP15ID>CP15_ID</CP15ID>
        </DEVICE>
        <DEVICE Number="2">
            <ConnectionID>ARM_Cortex-A9_1</ConnectionID>
            <CP15ID>CP15_ID</CP15ID>
        </DEVICE>
        <SimName>Imperas_EB_Cortex_A9MPx4</SimName>
    </RDDICADI>
</RVConfigUtility>
```

Figure 2: Example Board Configuration (a9mpx2_bare_metal.xml)

8.2.2 DS-5 Project Definition File

The project definition file defines the different types of debug connections that are supported on the board. Each debug connection defines

1. the launch script to use to start the execution of the platform
2. the launch script arguments
3. the core type to which the debug connection is made
4. any further python definition scripts to be included

Defines the different way you can connect to a board

```
        <core connection_id="Cortex-A9" core_definition="Cortex-A9"/>
```

The *connection_id* must match the name of one of the cores in the platform

The *core_definition* must match one of those provided in the Cores directory, this can be included in the configuration database to be added or be part of the current ARM DS-5 configuration database, if not we must supply our own.

```
<?xml version="1.0" encoding="UTF-8"?>
<platform_data
    type="RTSM"
    xmlns:peripheral="http://com.arm.targetconfigurationeditor"
    xmlns:xi="http://www.w3.org/2001/XInclude"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://www.arm.com/project_type"
    xsi:schemaLocation="http://www.arm.com/project_type ../../../Schemas/platform_data-
1.xsd">
```

Figure 3: Example Board Connection Definition (project_types.xml)

The list of projects available that can be selected in DS-5.

```
 <project_type_list>
   <project_type type="BARE_METAL" os_abstraction="BARE_METAL">
     <name language="en">Bare Metal Debug</name>
     <description language="en">This allows a bare metal debug connection to the ARM
Cortex-A9 Platform</description>
```

The project environment to connect to core 0 in the platform

```
  <execution_environment id="bare_metal_0">
      <name language="en">Bare Metal Debug 0</name>
      <description language="en">Connect to core 0 of a model to perform bare metal
debug</description>
      <param type="string" default="CDB://a9mpx2_bare_metal.xml" id="config_file"
visible="false"/>
      <param type="string" default="" id="model_params" visible="true">
          <name>Model parameters</name>
          <description>Model parameters</description>
      </param>
      <setup_script>
          <name>CDB://../../../../../launchScripts/imperas_platform_launcher.py</name>
          <arguments>Cortex-A9MPx2</arguments>
      </setup_script>
      <activity type="Debug" id="ICE_DEBUG">
          <name language="en">Debug Cortex-A9_0</name>
          <description language="en">Debug a bare metal application on the Cortex-A9 Core
0</description>
```

```
          <core core_definition="Cortex-A9" connection_id="ARM_Cortex-A9MPx2_CPU0"/>
      </activity>
  </execution_environment>
```

The project environment to connect to core 1 in the platform

```
  <execution_environment id="bare_metal_1">
      <name language="en">Bare Metal Debug 1</name>
      <description language="en">Connect to core 1 of a model to perform bare metal
debug</description>
      <param type="string" default="CDB://a9mpx2_bare_metal.xml" id="config_file"
visible="false"/>
      <param type="string" default="" id="model_params" visible="true">
          <name>Model parameters</name>
          <description>Model parameters</description>
      </param>
      <setup_script>
          <name>CDB://../../../../../launchScripts/imperas_platform_launcher.py</name>
          <arguments>Cortex-A9MPx2</arguments>
      </setup_script>
      <activity type="Debug" id="ICE_DEBUG">
          <name language="en">Debug Cortex-A9_1</name>
          <description language="en">Debug a bare metal application on the Cortex-A9 Core
1</description>
          <core core_definition="Cortex-A9" connection_id="ARM_Cortex-A9MPx2_CPU1"/>
      </activity>
  </execution_environment>
```

The following project environment is used to connect to both cores as an SMP cluster in the platform.

```
 <execution_environment id="bare_metal_SMP">
    <name language="en">Bare Metal SMP Debug</name>
    <description language="en">Connect to all model cores to perform bare metal debug
</description>
    <param type="string" default="CDB://a9mpx2_bare_metal.xml" id="config_file"
visible="false"/>
    <param type="string" default="" id="model_params" visible="true">
        <name>Model parameters</name>
        <description>Model parameters</description>
    </param>
    <setup_script>
       <name>CDB://../../../../../launchScripts/imperas_platform_launcher.py</name>
       <arguments>Cortex-A9MPx2</arguments>
    </setup_script>
    <activity id="ICE_DEBUG" type="Debug">
       <name language="en">Debug Cortex-A9x2 SMP</name>
       <description language="en">DS-5 Debugger will connect to platform to debug a bare
metal SMP application</description>
       <core core_definition="Cortex-A9" connection_id="Cortex-A9MPx2_SMP"/>
       <param type="string" default="CDB://a9mpx2_dtsl.py" id="dtsl_config_script"
visible="false"/>
       <param type="string" default="a9mpx2_SMP" id="dtsl_config" visible="false"/>
    </activity>
 </execution_environment>
```

```
    </project_type>
  </project_type_list>
</platform_data>
```

This file defines how the platform is launched using the <setup_script> definition

```
  <setup_script>
```

```
      <name>CDB://../../../../../launchScripts/imperas_platform_launcher.py</name>
      <arguments>OVPsim_arm Cortex-A9MPx2</arguments>
  </setup_script>
```

The arguments are passed to the launch script. In this case it represent the name of the demonstration directory within the Imperas installation from which the simulation will be executed and the variant of the processor to launch

8.2.3 Python scripts

There may be additional Python scripts required to define the SMP cluster for the debug session.

In this example the class ***a9mpx2_SMP*** is defined. This is used in the matching project environment entry for the setup of the SMP debug session. The names of the cores must match the names in the platform to which the debug session is connected.

```
from com.arm.debug.dtsl.configurations import DTSLv1
from com.arm.debug.dtsl.components import Device
from com.arm.debug.dtsl.components import CadiSyncSMPDevice


class a9mpx2_SMP(DTSLv1):
  def __init__(self, root):
        DTSLv1.__init__(self, root)

        # Create devices representing each core
        coreDevices = [ Device(self, 1, "Cortex-A9MPx2_0") , Device(self, 2, "Cortex-
A9MPx2_1")]

        # Expose each core to the debugger
        for c in coreDevices:
            self.addDeviceInterface(c)

        # Create SMP cluster of all cores
        self.smp = CadiSyncSMPDevice(self, "Cortex-A9MPx2_SMP", coreDevices)
        self.addDeviceInterface(self.smp)
```

Figure 4: Example SMP Cluster Definition (a9mpx2_SMP.py)

9 Limitations

The functionality supported in the Imperas DS-5 integration is similar to that of the GDB supported in Eclipse.

Specifically these features of DS-5/ARM hardware platforms/ARM Fast Models are not supported by this DS-5 Imperas integration:

Streamline
OS aware features
Passing arguments into main
Manage processor signals into breakpoints
Manage simulated Linux signals into breakpoints
Examine memory on specific buses - AHB, APB, and AXI
    Probably depends on the target config of mem map
Distinguish between hypervisor and guest memory (VMID reg)
    break-stop-on-vmid command
    print current-vmid command etc
    $vmid debugger variable for use in scripts
big.LITTLE support
Specifying Normal or Secure world for TrustZone platforms, e.g.:
    load myimage.axf N:0
    file myimage.axf S:0
    loadfile myimage.axf S:0
Debugging UEFI
Stackbase and heapbase addresses
Automatic semihosting
Logging control
RTOS view
Screen view
Target view
Trace points
    Trace view
    Tracepoint properties dialog box
    Export trace report dialog box
    DTSL Configuration Editor dialog box
Auto Refresh Properties dialog box
Manage Signals dialog box
Functions Filter dialog box
Remote System
    Explorer, View, Details View
Target management terminal for serial and SSH connections
Remote Systems terminal for SSH connections
New Terminal Connection dialog box
Page support

Viewing into OVP peripherals
Debugger continue running backwards, and reverse
Snapshot Viewer
VFS: virtual file system support
Event Viewer
Event Viewer dialog box
Conditional breakpoints
Thread specific breakpoints (linux apps)
Call stack per thread
Debugger tracking of multi threaded applications
    script variable $thread
    separate stack frame shown for each thread
Thread, process control

Application Debug (in simulated Linux) with rewind support
ARM ASM view
ARM assembler editor
Remote Scratchpad view

##