



Imperas Installation and Getting Started Guide

This document explains how to install and get started with the OVPsim simulator / models and the Imperas professional products under Linux and Windows.

Imperas Software Limited

Imperas Buildings, North Weston,
Thame, Oxfordshire, OX9 2HA, UK

docs@imperas.com



Author:	Imperas
Version:	2.3.10
Filename:	Imperas_Installation_and_Getting_Started.doc
Project:	Imperas / Open Virtual Platforms Installation and Getting Started
Last Saved:	Tuesday, 26 January 2021
Keywords:	Installation, Getting Started, Linux, Windows, Licensing

Copyright Notice

Copyright © 2021 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.....	7
2	Introduction.....	7
3	Hardware and Software Requirements	8
3.1	Operating System.....	8
3.2	Hardware.....	8
3.3	Host Compiler Versions.....	8
4	Installation.....	9
4.1	Installation Tutorial Video.....	9
4.2	Packages.....	9
4.2.1	OVPsim.....	9
4.2.2	Imperas Developer.....	10
4.2.3	Imperas Advanced Multicore Software Development Kit.....	10
4.3	Access and Download of Installation Packages.....	11
4.3.1	Login.....	11
4.3.2	Download the Appropriate Package	11
4.3.3	Package Selection	11
4.4	Installing Under Linux.....	13
4.5	Installing Under Windows	15
4.6	Setting up the Environment	17
4.6.1	Required Environment.....	17
4.6.2	Additional Environment.....	18
4.6.3	Script to Setup Required Environment	18
4.6.4	Script to Select Simulator Runtime.....	19
4.6.5	Script to Select Personality	19
4.6.6	Explicit Environment Configuration.....	20
4.6.7	Internet Access Via a Proxy Server	23
5	Setting up Licensing	25
5.1	OVPsim Node Locked License keys	25
5.1.1	Obtaining the computer hostname and Host ID.....	25
5.1.2	Setup License File.....	26
5.2	Imperas Floating Licenses	27
5.2.1	Obtaining the computer hostname and Host ID.....	27
5.2.2	Starting the License Server	27
5.2.3	License Queuing	28
6	Windows Development Environment.....	29
6.1	Introduction.....	29
6.2	MSYS2 / MinGW Environment	29
6.2.1	Obtaining MSYS2.....	29
6.2.2	Install Utilities.....	34
6.2.3	Install Host Toolchains	35
6.2.4	Environment.....	36
6.2.5	Completion and test of MSYS/MinGW installation.....	40
7	Additional Toolchain Packages	43

7.1	Application Cross Compiler Toolchains.....	43
7.2	Peripheral Simulation Engine (PSE) Toolchain	45
8	Getting Started	46
8.1	ISS Tutorial Video	46
8.2	Check Installation	46
8.2.1	Simulator Execution.....	46
8.2.2	iGen Installation.....	48
8.2.3	Harness Installation.....	49
8.2.4	MPD Installation.....	50
8.2.5	eGui Installation.....	52
8.3	Build Environment.....	53
8.3.1	Introduction.....	53
8.3.2	Standard Makefiles	55
8.3.3	Build Output Selection.....	55
8.3.4	VLNV Component Library.....	56
8.4	Hello World Example	61
8.4.1	Compiling an application.....	61
8.4.2	Running the simulation using the ISS.....	62
8.4.3	Execute example using provided script	63
8.4.4	Creating a Virtual Platform Tutorial Video	63
8.4.5	Creating and Simulating with a Platform/Module	64
8.4.6	Writing your own test harness	65
9	Understanding Semihosting Support	67
9.1	In Imperas and OVP simulations	67
9.2	Replacing function and/or instruction behavior.....	67
9.3	Specific to a Cross Compiler and C Library	67
9.4	Used to terminate the simulation	68
9.5	Caution using with EPKs and non-baremetal platforms.....	68
10	Understanding Simulation Time Statistics	69
11	Understanding the operation of a code morphing simulator.....	71
11.1	Instruction Fetch	71
11.2	SystemC Interface Transaction Types	72
11.2.1	OP API (Current)	72
11.2.2	ICM API (Deprecated).....	72
	APPENDIX A Installation Packages, Products and Licensing Features.....	73
A.1	Installation Packages.....	73
A.1.1	Imperas_SDK package.....	73
A.1.2	Imperas_DEV package	73
A.1.3	OVPsim package.....	74
A.2	License Features.....	74
A.2.1	Executable Programs	74
A.2.2	CpuManager Simulator and its Personalities.....	75
A.2.3	CpuManager Simulator and Non-Interactive (batch) Usage	75
A.2.4	OVPsim Simulator.....	76
	APPENDIX B Obtaining an Imperas License	77
	APPENDIX C Accessing Imperas User Area	78

C.1	Initial Login at Imperas User	78
C.2	Selecting Imperas Product download.....	78
C.3	Logging into Download area	79
C.4	Selecting Files.....	79
APPENDIX D Setting Environment Variables		80
D.1	Opening System properties on Windows XP.....	80
D.2	Opening System properties on Windows 7 and 10.....	80
D.3	Modifying Environment Variables in Windows.....	82
APPENDIX E Additional FlexNet Licensing Information		83
E.1	Locating the license server software.....	83
E.2	Types of Licenses	83
11.2.3	E.2.1 Uncounted Node-locked Licenses	83
11.2.4	E.2.2 Floating Licenses.....	84
E.3	Starting the License Server	85
E.4	Configuring the Host Computer.....	86
E.5	Other License File Configurations.....	86
E.6	The daemon options file.....	87
E.7	License Administration tools	88
E.8	License Server Manager as a Windows Service using LMTOOLS	88
APPENDIX F Some Common Problems		91
F.1	Segmentation Fault when Native Debug of an OVP Platform	91
F.2	Simulator Reports Internal Abort (ASRT).....	92
F.3	Building on Windows using mingw32-make	92
F.3.1	Error in Makefile.pse when building peripherals	92
F.4	Licensing.....	93
F.4.1	Feature Not Supported.....	93
F.4.2	Long Delay (30 second) at Simulator Start Up	94
F.4.3	Cannot Access Date Server	94
F.5	Ethernet Adapter Naming/HostId is zero.....	95
F.5.1	How to change the default 'ens33' network device to old 'eth0'	95
F.5.2	How to reorder or rename logical interface names in Linux	96
F.5.3	Based on the physical properties	96
F.5.4	Based on the MAC address.....	97
F.5.5	Based on the driver	100
F.5.6	Based on the physical location in the computer	101
F.5.7	Ubuntu directly changing logical names	102
F.5.8	More Information.....	102
F.6	What to try if license server (lmgrd) fails to run	102
APPENDIX G Abort Detected in Program		104
G.1	Obtaining Backtrace.....	105
G.2	Connecting a Debugger to a running simulation	105
APPENDIX H Other Windows Development Environments		107
H.1	Using a Cygwin Environment.....	107
H.1.1	Use MINGW GNU Toolset	107
H.2.2	Verify use of MINGW GNU Toolset	107
H.3.3	Cannot Build OVPsim Examples.....	108

H.4	Building Virtual platforms with Microsoft MSVC.....	110
H.4.1	Running the Example.....	110

1 Preface

This document describes how to install the Imperas Professional products, Advanced Multicore Software Development Kit (M*SDK), Virtual Platform Development and Simulation (C*DEV, S*DEV and M*DEV), and the Open Virtual Platforms OVPsim simulator on both Windows and Linux computers. The installation includes tools, models, documents and examples.

This document further introduces how to compile applications using the supplied example GNU cross compiler toolchains/tool kits and how to use this binary with a virtual platform.

Please see other documents that explain how to create platforms / harnesses, processor models, and peripheral / behavioral models.

2 Introduction

This installation guide is for the installation of the OVP or Imperas Professional products on a Linux or Windows host computer. We will need the installation files of either OVPsim from www.OVPworld.org or the Imperas products from www.Imperas.com, and at least one example compiler toolchain, also from www.OVPworld.org.

OVP is provided as a single installation package ‘OVPsim’.

Imperas products are provided as two installation packages ‘Imperas_SDK’ for M*SDK and ‘Imperas_DEV’ for C*DEV, S*DEV and M*DEV (configured on installation or post-install with configuration script).

In order to build native host executables and to cross compile applications under Windows, we suggest the installation of an MSYS/MinGW¹ environment www.mingw.org. See section 6 for instructions on setting up this environment. This will make compilation of Applications, Models and Platforms / Simulation harnesses simple.

Part of the installation includes examples files and these will be used to illustrate the processes and tools used to get your applications running on virtual simulation platforms. In this document, we will use the RISC-V and the openCores openRISC OR1K as the target embedded processor.

¹ It is also possible to use a Cygwin / MinGW environment but care must be taken to ensure that the MINGW GNU toolset is used.

3 Hardware and Software Requirements

3.1 Operating System

These are the versions of the operating systems that are currently used by Imperas for development and product verification; the product has been shown to operate correctly on other operating systems.

<u>Product Build</u>	<u>Operating System</u>
Linux 32-bit	Fedora Core 12
Linux 64-bit ²	Fedora Core 18
Windows 32-bit	Windows 7 64-bit ³ Professional with SP1
Windows 64-bit	Windows 7 64-bit Professional with SP1

3.2 Hardware

The product is developed to work on x86 hardware.

Supported Processors: x86 32-bit and 64-bit

Disk space requirements⁴:

OVP or Imperas Installations:	140MB to 220MB
Toolchain Installations (each):	60MB to 300MB

3.3 Host Compiler Versions

The following are the GCC compiler versions that are used by imperas for generation of host code. The products are developed in such a way as to maintain, as far as possible, backwards compatibility with previous compiler versions, in particular the GLIBC versions.

<u>Product Build</u>	<u>GCC Compiler Version</u>
Linux 32-bit	4.9.1 i686-nptl-linux-gnu (Crosstool-ng)
Linux 64-bit	4.9.1 x86_64-unknown-linux-gnu (Crosstool-ng)
Windows 32-bit	4.4.7 ⁵ i686-unknown-mingw32
Windows 64-bit	4.9.1 x86_64-unknown-mingw32

² This is only supported by the Imperas installations, Imperas_SDK and Imperas_DEV.

³ The Imperas 32-bit products are also supported in emulation mode on 64-bit hosts.

⁴ The disk space indicated is per package install or toolchain install. The actual disk space required is dependent upon the type and number of packages or toolchains installed.

⁵ The Windows 32-bit ABI changed at release 4.6 which introduced incompatibilities in the Windows ABI. The OVP products are compiled with the legacy ABI.

4 Installation

The Imperas professional products and the OVPsim simulator may be installed on a Linux or Windows platform.

Section 4.2 provides specific information regarding the product packages.

The details of the package installations are shown for Linux users in section 4.4, and Windows users in section 4.5.

Different major versions of the software should not be installed on top of each other. The new installation should either be made into a different directory or the old version uninstalled before the new version is added.

You should avoid making changes to files in the installation directories, as those changes will be lost when the version is un-installed and a subsequent version is installed. Different minor versions may be installed together; later minor versions typically provide updates to products.

4.1 Installation Tutorial Video

The video found on the OVPWorld website page <http://www.ovpworld.org/getting-started-with-ovp-and-imperas> will provide a guide through the installation and setup of OVP and Imperas environments.

4.2 Packages

4.2.1 OVPsim

The OVPsim simulation package (OVPsim) contains:

1. the OVPsim reference simulator
2. the OVP model library
3. the Imperas iGen model template wizard (productivity tool)
4. examples and demonstrations

Refer to section 4.3 for information about downloading this package. Section 4.4 gives information about installing the package on Windows. Section 4.5 gives information about installing the package on Linux.

4.2.2 Imperas Developer

The Imperas Developer package, (`Imperas_DEV`) contains:

1. a professional version of the simulator that can be licensed for different capabilities
 - a. Controller (C*DEV) : single processor / multi-core processor
 - b. Standard (S*DEV) : multiple processors / multi-core processors (homogeneous processor vendor)
 - c. Multi (M*DEV) : multiple processors / multi-core processors (heterogeneous processor vendors)
2. the OVPsim package contents
3. Imperas Developer specific examples and demonstrations

Refer to section 4.3 for information about downloading these packages. Section 4.4 gives information about package installation on Windows. Section 4.5 gives information about package installation on Linux.

Once the package is installed

- the `IMPERAS_PERSONALITY` environment variable must be set to one of the following:

For Controller (C*DEV)	: <code>CPUMAN_CONTROLLER</code>
For Standard (S*DEV)	: <code>CPUMAN_STANDARD</code>
For Multi (M*DEV)	: <code>CPUMAN_MULTI</code>
- the `IMPERAS_RUNTIME` environment variable must be set to `CpuManager`

4.2.3 Imperas Advanced Multicore Software Development Kit

The Imperas SDK package (`Imperas_SDK`) contains:

1. a professional version of the simulator that is licensed for capability
 - a. Multi (M*SDK) : multiple processors / multi-core processors (heterogeneous processor vendors)
: Capable of executing VAP Tools and other advanced features.
2. the OVPsim package contents
3. the Imperas_DEV package contents
4. The Imperas Verification, Analysis and Profiling (VAP) tools
5. The Imperas Interactive control and Multi-Processor Debugger (MPD)
6. Imperas M*SDK specific examples and demonstrations

Refer to section 4.3 for information about downloading this package. Section 4.4 gives information about installing the package on Windows. Section 4.5 gives information about installing the package on Linux.

Once the package is installed

- the `IMPERAS_PERSONALITY` environment variable must be set to `CPUMAN_MULTI`
- the `IMPERAS_RUNTIME` environment variable must be set to `CpuManager`

4.3 Access and Download of Installation Packages

First you must download the installation files from either OVPworld.org or Imperas.com.

4.3.1 Login

4.3.1.1 OVP Users

The OVPworld download page can only be accessed when you are registered and logged into the sites forum. Please register (it can take up to 24 hours to receive an authorization email) and be logged in from the forums page before continuing.

4.3.1.2 Imperas Users

Imperas customers should go to www.imperas.com and select user login. This accepts the same username and password as the OVPworld.org site. This will take you to the Imperas User Home Page. Click on Imperas Product Downloads Imperas link (or Beta if you require an updated pre-release version to download) and you will be requested to enter your Imperas user name and password, which should have been provided to you by Imperas. The imperas user area also provides 64-bit versions of the OVP packages via the OVP link. See “APPENDIX C Accessing Imperas User Area” for additional information about accessing the Imperas User site and downloading packages.

4.3.2 Download the Appropriate Package

To download OVPsim after logging in to the OVPworld website go to the downloads page <http://www.ovpworld.org/download.php>, look on the right hand side list and select the appropriate link for the version you wish to download (Windows or Linux). This will download the Windows or Linux installer executable for OVPsim.

To download the Imperas professional tools or the Imperas development tools, log into the Imperas.com site and navigate to the Imperas Product Downloads area. Then, select the product you wish to download (Windows or Linux).

4.3.3 Package Selection

On the OVPWorld website, a single package is available on the *Main Downloads* page:

OVPsim package

- `OVPsim.<version>.<dot release>.<HOST>32.exe`

On the Imperas website, the following packages are available after logging in and following the link Imperas *Current Release Product Downloads*:

Developer packages

- `Imperas_DEV.<version>.<dot release>.<HOST>32.exe`
• `Imperas_DEV.<version>.<dot release>.<HOST>64.exe`

Advanced Multicore Software Development package

- `Imperas_SDK.<version>.<dot release>.<HOST>32.exe`
`Imperas_SDK.<version>.<dot release>.<HOST>64.exe`

In all cases <HOST> can be Windows or Linux

4.4 Installing Under Linux

This section tells you how to install the Linux version of the products. Windows users should refer to section 4.5.

Linux Product versions are provided as pre-built binaries in a self-extracting executable file.

Execute the self-extracting executable file to install. You may need to change the installer to be executable first:

```
$ chmod +x Imperas_SDK.<major version>.<minor version>.Linux64.exe6
```

The self-extracting executable will install the files in the current directory or in a selected directory. Execute the installer to extract the files:

```
$ ./Imperas_SDK.<major version>.<minor version>.Linux64.exe
```

You will be asked to accept a license agreement. It should be read and, if acceptable, agreed to by typing yes at the prompt. For example:

```
Software License Agreement for Imperas Proprietary Software
Imperas Limited.
-----
<.. lines removed ..>
You are not allowed to use this software without a specific signed license
agreement.

Imperas_Software_License_Note (v.1.1.0)

Please indicate whether you accept the license agreement
yes/no >
```

Once the license agreement has been accepted, you will be prompted for the directory to install into:

```
Install into directory <current directory>/Imperas.<release major version>
yes/no >
```

If you reply yes, the installation will carry on into the specified directory. If you reply no, you will be prompted for a directory to install into:

```
Please provide an installation directory:
```

⁶ Note that these instructions assume that an installation of the 64-bit Imperas_SDK package is being done, but the same general flow is followed for all packages.

This will create `<installDir>/Imperas.<release major version>`, where `<installDir>` is the directory that was entered at the prompt. Since the tools are installed in a directory tree whose root contains the version, multiple versions may coexist side by side.

The Linux binary executables and shared objects are in the directory:

`<installDir>/Imperas.<major version>/bin/Linux64.`

Model libraries are provided, as shared objects, in the directory:

`<installDir>/Imperas.<major version>/lib/Linux64/ImperasLib`

This path must be specified using the `IMPERAS_VLNV` environment variable.

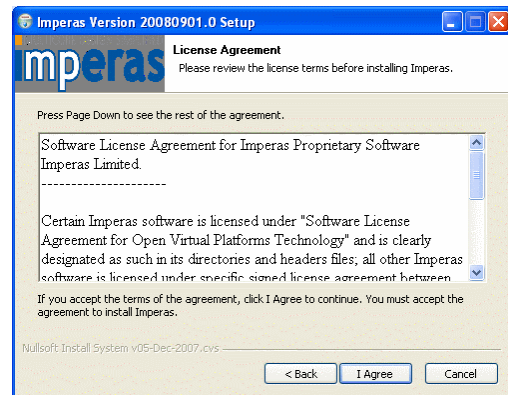
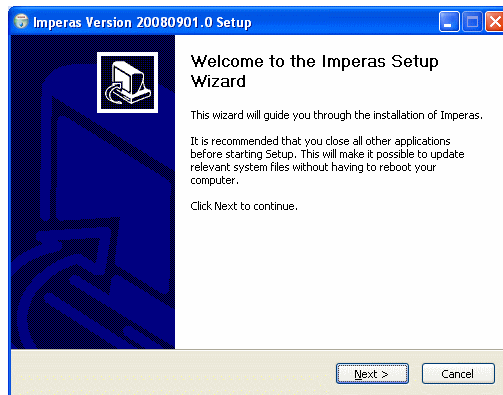
4.5 Installing Under Windows

This section tells you how to install the Windows version of the tools. Linux users should refer to section 4.4.

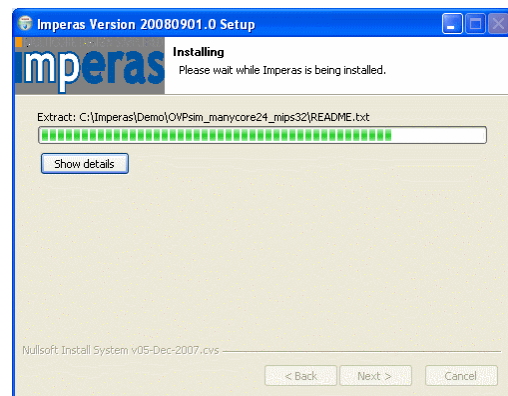
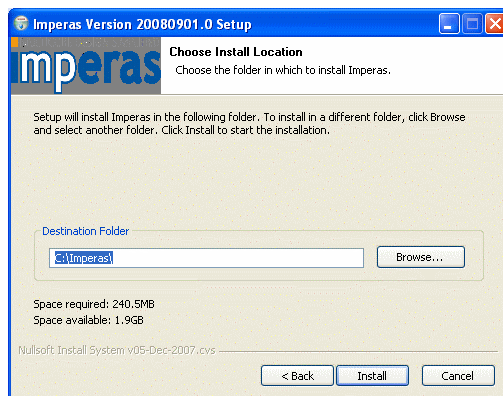
Windows Product versions are provided as executable installers.

The installer, when executed, will extract the tools and set up the environment. Remember to uninstall any existing versions before installing. An uninstall program, `uninstall.exe` is provided in the root install directory (i.e. `<installDir>\uninstall.exe`, e.g. `C:\Imperas\uninstall.exe`)

1. Execute the provided installer
2. Read and click through the license agreement

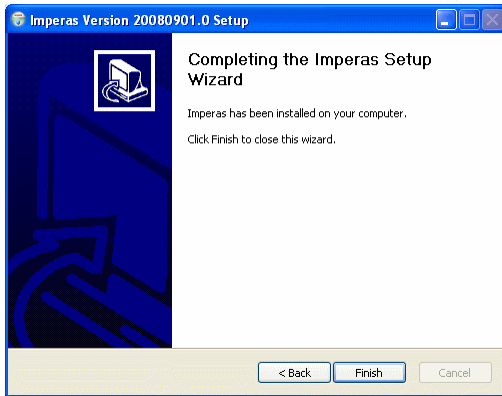


3. Choose the location for installation
4. Start the installation



5. When completed click Finish
6. Finally, read the notes for this release.

Imperas Installation and Getting Started Guide



4.6 Setting up the Environment

In order to run Imperas products, environment variables have to be set to configure settings controlling product location, features and licensing.

On Windows, the execution of the installer will setup the required environment or the environment can be modified on each shell.

On Linux, the environment must be setup in each shell. This could be done manually or as part of an automated setup script on starting a shell.

The environment variables are divided into *required* and *additional* groups, as follows:

4.6.1 Required Environment

Required Environment Variables

IMPERAS_HOME	Points to the root of the Imperas installation
IMPERAS_UNAME	Is set to the Host OS type, Linux or Windows
IMPERAS_ARCH	Is set to the Host architecture, Linux32, Windows64 etc.
IMPERAS_SHRSUF	Is set to the suffix for shared libraries, so or dll
IMPERAS_VLNV	Points to the root of the compiled library in the Imperas installation
IMPERAS_RUNTIME	Specifies which simulator, Imperas (CpuManager) or OVPsim, to load at runtime ⁷
IMPERAS_PERSONALITY	Specifies the features available from the Imperas product runtime. This also modifies the license features that will be required.

NOTE

Prior to product release 20170511.0 a default personality would be assumed if the environment variable was not set.

This is no longer the case; the personality must always be specified.

The standard environment variable **PATH** should include
\$IMPERAS_HOME/bin/\$IMPERAS_ARCH

On Linux, the standard environment variable **LD_LIBRARY_PATH** should include
\$IMPERAS_HOME/bin/\$IMPERAS_ARCH and
\$IMPERAS_HOME/lib/\$IMPERAS_ARCH/External/lib

⁷ Although specified in the required list the IMPERAS_RUNTIME environment variable may be left unset in which case the default OVPsim simulator is selected.

4.6.2 Additional Environment

Additional Environment Variables

`IMPERAS_ISS` If this is set it will select the Imperas Instruction Set Simulation executable version.

The following must be set manually, if required:

`IMPERAS_PROXY_SERVER` Only required for Demo and Web licenses, if the internet is accessed through a proxy server, this should be set to the value `<hostname>:<port>`, e.g. `myproxyserver.com:3128`

`IMPERAS_RUNTIME` This may be set or left unset. If unset the default OVPsim simulator will be loaded, this will require an OVPsim license to be available.

4.6.3 Script to Setup Required Environment

A script `setup` is provided which can be used to setup the required environment. The script is provided in the `bin` directory below the Imperas installation directory.

On Linux, this script should be sourced in a Bourne shell; it defines the shell function `setupImperas`, which is then executed passing the *full* path to the Imperas directory as the argument.

```
$ source <installDir>/Imperas.<major version>/bin/setup.sh
$ setupImperas <installDir>/Imperas.<major version>
```

On Windows, the script can be double clicked in the Windows Explorer or can be called from a shell. The script will update the Windows environment.

```
$ call <installDir>/Imperas.<major version>/bin/setup.bat
```

4.6.3.1 32-bit Product on 64-bit Host

The `setupImperas` script also allows for the use of a 32-bit product on a 64-bit host. This requires that the 64-bit host has 32-bit compatibility libraries installed and available.

To specify this configuration, add the `-m32` argument to the `setupImperas` call on Linux:

```
$ source <installDir>/Imperas.<major version>/bin/setup.sh
$ setupImperas <installDir>/Imperas.<major version> -m32
```

Or to the call to the setup script on Windows:

```
$ call <installDir>/Imperas.<major version>/bin/setup.bat -m32
```

4.6.3.2 32-bit Compatibility libraries on 64-bit Linux Hosts

When running a 32-bit product on a 64-bit host, or when using some of the OVP toolchains that are only provided as 32-bit executables in the 64-bit package files, you will need to have 32-bit compatibility libraries on the 64-bit host machine.

Specific libraries to download will depend on your particular host environment.

If you are using Ubuntu try the following command line:

```
$ sudo apt-get install lib32z1
```

If you are using Fedora try the following command line:

```
$ sudo yum install zlib.i686
```

If you are using Centos try the following command line:

```
// Runtime libraries for libc and libcstd++
```

```
$ sudo yum install glibc.i686
```

```
$ sudo yum install libstdc++-4.8.5-28.el7_5.1.i686
```

```
// Compile time libraries for libc(i686) and libc(x64)
```

```
$ sudo yum install glibc-devel.i686 glibc-devel
```

4.6.4 Script to Select Simulator Runtime

A script `switchRuntime` is provided which can be used to setup the required runtime in environment variable `IMPERAS_RUNTIME`. The script is provided in the `bin` directory below the Imperas installation directory. A list of the available simulator runtimes installed is provided which is selected from and used to set the environment variable

On Linux, this script should be sourced in a Bourne shell; it defines the shell function `switchRuntimeImperas`, which is then executed.

```
$ source <installDir>/Imperas.<major version>/bin/switchRuntime.sh  
$ switchRuntimeImperas
```

On Windows, the script can be double clicked in the Windows Explorer or can be called from a shell. The script will update the Windows environment.

```
$ call <installDir>/Imperas.<major version>/bin/switchRuntime.bat
```

4.6.5 Script to Select Personality

A script `selectPersonality` is provided which can be used to setup the required simulator personality in environment variable `IMPERAS_PERSONALITY`. The script is provided in the `bin` directory below the Imperas installation directory. A list of the

available personalities is provided which is selected from and used to set the environment variable

On Linux, this script should be sourced in a Bourne shell; it defines the shell function `selectPersonalityImperas`, which is then executed.

```
$ source <installDir>/Imperas.<major version>/bin/selectPersonality.sh
$ selectPersonalityImperas
```

On Windows, the script can be double clicked in the Windows Explorer or can be called from a shell. The script will update the Windows environment.

```
$ call <installDir>/Imperas.<major version>/bin/selectPersonality.bat
```

4.6.6 Explicit Environment Configuration

4.6.6.1 Linux

As an alternative to using the setup script, on Linux, or using the pre-defined environment created by the installer, on Windows, the environment can be configured by setting environment variables explicitly. These sections show how this is done.

As an alternative to using the `setup.sh` script, the Linux environment can be configured by setting environment variables explicitly. This section shows how this is done.

Create an environment variable `IMPERAS_HOME` pointing to the root of the Imperas tree, for example:

```
$ export IMPERAS_HOME=<installDir>/Imperas.20130630
```

Create an environment variable `IMPERAS_ARCH` for the current architecture, for example:

```
$ export IMPERAS_ARCH=Linux64
```

Create an environment variable `IMPERAS_SHRSUF` for the current architecture, for example:

```
$ export IMPERAS_SHRSUF=so
```

Create an environment variable `IMPERAS_VLNV` pointing to the root of the Imperas library, for example:

```
$ export IMPERAS_VLNV=$IMPERAS_HOME/lib/$IMPERAS_ARCH/ImperasLib
```

Create an environment variable `IMPERAS_RUNTIME` specifying which library to load at runtime. If you have a license for the Imperas tools this should be:

```
$ export IMPERAS_RUNTIME=CpuManager
```

If you have a license for OVPsim then this may be left unset (as this is the default value) or set to:

```
$ export IMPERAS_RUNTIME=OVPSim
```

Add the Imperas executables directory to the search path, for example:

```
$ export PATH=${PATH}:${IMPERAS_HOME}/bin/${IMPERAS_ARCH}
```

Add Imperas shared library directories to the variable `LD_LIBRARY_PATH`, for example⁸:

```
$ export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${IMPERAS_HOME}/bin/${IMPERAS_ARCH}:\
$IMPERAS_HOME/lib/${IMPERAS_ARCH}/External/lib
```

Set the product personality:

```
$ export IMPERAS_PERSONALITY=CPUMAN_STANDARD
```

Next, you must also set up licensing for the program. This is described for both Linux and Windows installations in section 4.6.7.

4.6.6.2 Windows

As an alternative to using the setup script, *setup.bat*, or the pre-defined environment created by the installer the environment can be configured by setting environment variables explicitly.

This can be done in the MSYS shell or using the Windows System Control Panel.

4.6.6.2.1 MSYS Shell

Create an environment variable `IMPERAS_HOME` pointing to the root of the Imperas tree, for example:

```
$ set IMPERAS_HOME=<installDir>\Imperas
```

Create an environment variable `IMPERAS_ARCH` for the current architecture, for example:

```
$ set IMPERAS_ARCH=Windows64
```

Create an environment variable `IMPERAS_SHRSUF` for the current architecture, for example:

```
$ set IMPERAS_SHRSUF=dll
```

Create an environment variable `IMPERAS_VLNV` pointing to the root of the Imperas library, for example:

⁸ This is all one command; note the use of the Linux line continuation character ‘\’ at the end of the first 2 lines

```
$ set IMPERAS_VLNV=%IMPERAS_HOME%\lib\%IMPERAS_ARCH%\ImperasLib
```

Create an environment variable `IMPERAS_RUNTIME` specifying which library to load at runtime. If you have a license for the Imperas tools this should be:

```
$ set IMPERAS_RUNTIME=CpuManager
```

If you have a license for OVPSim then this may be left unset (as this is the default value) or set to:

```
$ set IMPERAS_RUNTIME=OVPSim
```

Add the Imperas executables directory to the search path, for example:

```
$ set PATH=%PATH%;%IMPERAS_HOME%\bin\%IMPERAS_ARCH%
```

Set the product personality:

```
$ set IMPERAS_PERSONALITY=CPUMAN_STANDARD
```

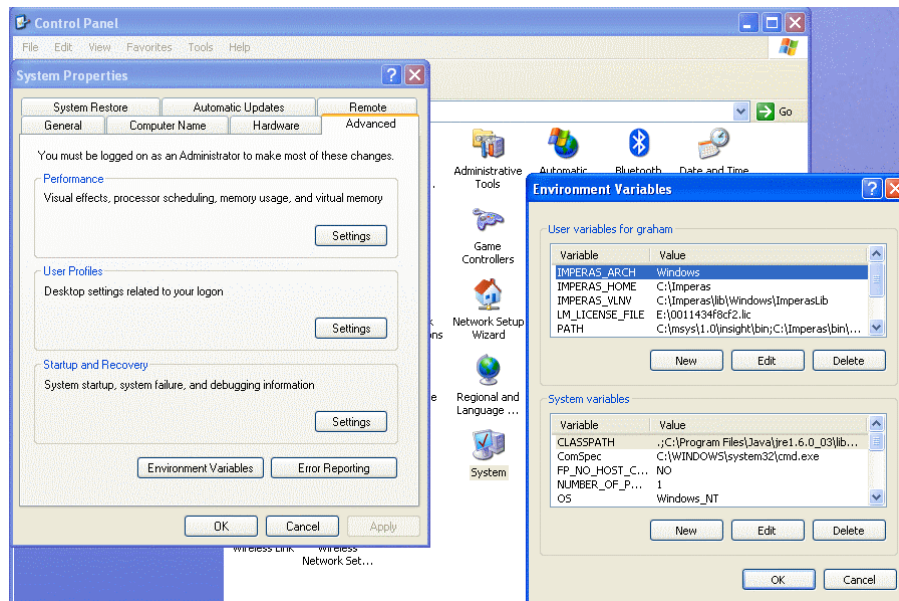
Next, you must also set up licensing for the program. This is described for both Linux and Windows installations in section 4.6.7.

You may also wish to install the preferred Windows development environment, MSYS. This is described in section 6 Windows Development Environment.

4.6.6.2.2 Windows System Menu

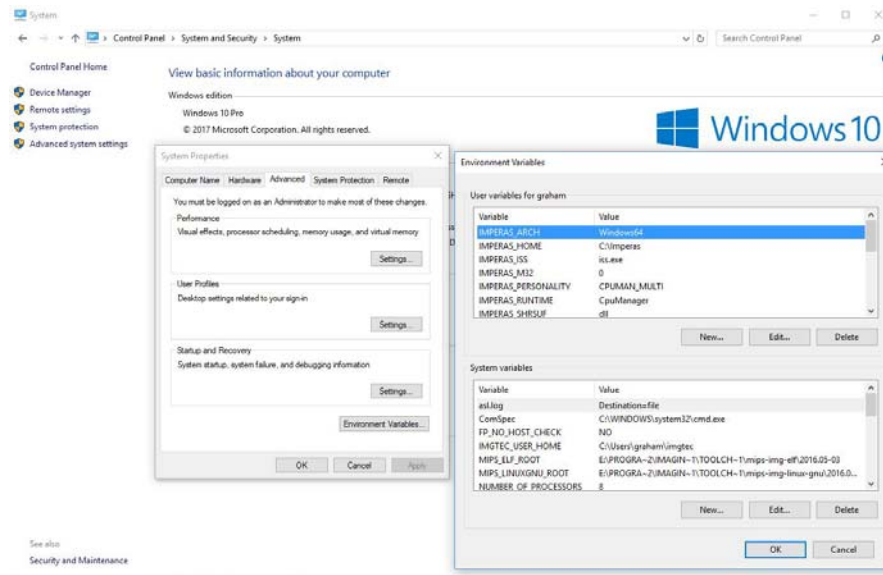
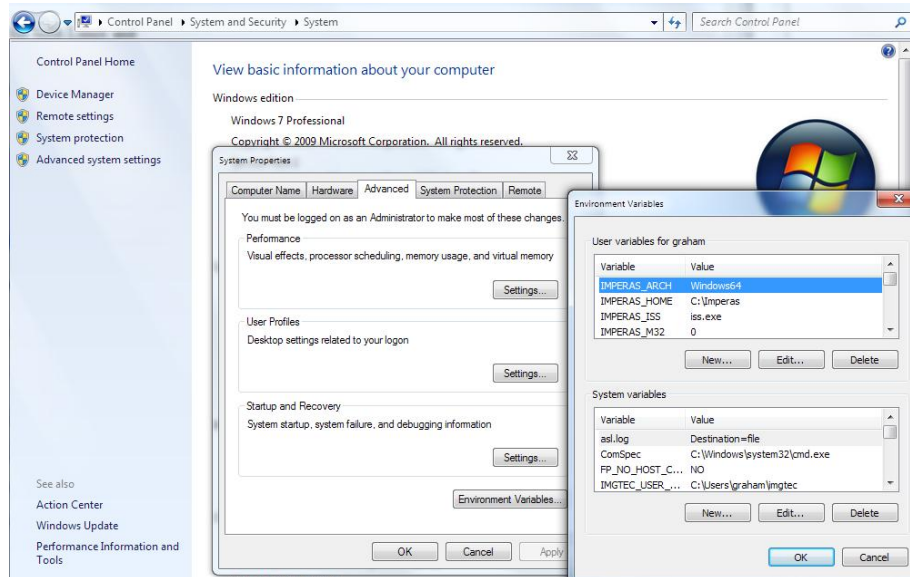
Environment variables on Windows XP can be modified by going to:

Start->Control Panel->System->Advanced->Environment Variables



Environment variables on Windows 7 and 10 can be modified by going to:

Start->Control Panel->System and Security->System->Advanced System Settings->Environment Variables



4.6.7 Internet Access Via a Proxy Server

If you are using a Demo license downloaded from the OVP website, this will require web access in order to run the simulator.

If this is the case and you use an Internet Proxy Server to access the web, you must set the environment variable, `IMPERAS_PROXY_SERVER`, in order to enable access to the internet.

For example (shown for a Linux host) if the Proxy Server is running on `myproxyserver.com` at port 3128

```
$ export IMPERAS_PROXY_SERVER="http://myproxyserver.com:3128"
```

A simple check to ensure this has worked correctly is as follows

```
$ export IMPERAS_PROXY_SERVER="http://myproxyserver.com:3128"
$ http_proxy=${IMPERAS_PROXY_SERVER} wget http://www.ovpworld.org
```

This should cause the `index.html` to download if all is successful

5 Setting up Licensing

The OVP and Imperas products are licensed using Flexera license software. In order to execute you will need a license file and for floating licenses you will need to run a license server.

Floating licenses to execute the Imperas Professional products are available by contacting Imperas at license@imperas.com.

In both cases the license file is bound to an individual computer through that computer's host ID. To obtain a license file you must provide the host ID and host name of one of your physical computers that will host the license.

NOTE

Running the Imperas license server within a virtual machine will not work correctly and is not permitted under the terms of the Imperas Software License Agreement.

5.1 OVPsim Node Locked License keys

OVPsim node locked license keys may be requested directly from <http://www.ovpworld.org/licensekey.php> by registered OVP users. Licenses for commercial use may be obtained from Imperas.

OVPsim licenses, from the OVPWorld website, are uncounted node-locked licenses and as such do not require a license server or a license daemon.

5.1.1 Obtaining the computer hostname and Host ID

To determine the host ID of a computer, use the `lmhostid` command of the `lmutil` utility program (`lmutil` on Linux or `lmutil.exe` on Windows).

If you have already gone through the installation process then the `lmutil` utility will be in the directory `IMPERAS_HOME/bin/IMPERAS_ARCH` which will already be on your executable path.

From a Linux shell or a Windows Command Prompt/MSYS shell do the following:

```
$ lmutil lmhostid
lmutil - Copyright (c) 1989-2012 Flexera Software LLC. All Rights Reserved.
The FlexNet host ID of this machine is "002486571114"
```

NOTE

If the returned host ID is "000000000000" then this may be because you are using an OS which uses Consistent Network Device Naming. Please see appendix F.5 for ways in which this problem can be resolved. A license cannot be generated for a NULL host ID.

The host name of the computer can be obtained with the `-hostname` argument applied to the `lmutil` utility program. From a Linux shell or a Windows Command Prompt/MSYS window do the following:

```
$ lmutil.exe lmhostid -hostname
lmutil - Copyright (c) 1989-2017 Flexera Software LLC. All Rights Reserved.
The FlexNet host ID of this machine is "HOSTNAME=user1-laptop"
```

5.1.2 Setup License File

OVPsim looks, by default, for the license file in:

```
$IMPERAS_HOME/OVPsim.lic
```

If the license file is installed in this location then there is no need to do anything. However, maintain a copy of the license file elsewhere as this will be deleted if uninstalling or overwritten when installing a new package containing licensing.

If it is installed in a different location the `IMPERASD_LICENSE_FILE`⁹ environment variable should be set to its location, as shown below.

```
IMPERASD_LICENSE_FILE=/home/user/Imperas_OVPsim.lic
```

OVPsim license key files are to a specific version (release) of OVPsim, with a license feature entry similar to that shown below.

```
FEATURE IMP_OVPSIM_20170919 imperasd 1.0 1-jul-2017 uncounted HOSTID=...
SIGN="..."
```

You will need to obtain a new license file when you install a new version of the simulator.

NOTE

When using an OVPsim node locked license the simulator product will also require access to the internet to allow for checking for product updates. Without internet access an error will be reported.

⁹ The `IMPERASD_LICENSE_FILE` environment variable is an alternative to `LM_LICENSE_FILE`, *either may be used*.

5.2 Imperas Floating Licenses

Floating licenses require that a license server is executed on a machine that will be used to serve licenses to clients. The license server, `lmgrd` (`lmgrd.exe` on Windows) is provided in the binary directory of an installation.

The license to run the Imperas products (and the OVPsim product in some circumstances) is provided as a floating license where one computer on the network is the license server and the same or another computer on the network requests (checks out) licenses from it to enable the product to run. The license server uses a license daemon.

5.2.1 Obtaining the computer hostname and Host ID

To determine the host ID of a computer, use the `lmhostid` command of the `lmutil` utility program (`lmutil` on Linux or `lmutil.exe` on Windows).

If you have already gone through the installation process then the `lmutil` utility will be in the directory `IMPERAS_HOME/bin/IMPERAS_ARCH` which will already be on your executable path.

From a Linux shell or a Windows Command Prompt/MSYS shell do the following:

```
$ lmutil lmhostid
lmutil - Copyright (c) 1989-2017 Flexera Software LLC. All Rights Reserved.
The FlexNet host ID of this machine is "002486571114"
```

The host name of the computer can be obtained with the `-hostname` argument applied to the `lmutil` utility program. From a Linux shell or a Windows Command Prompt/MSYS window do the following:

```
$ lmutil.exe lmhostid -hostname
lmutil - Copyright (c) 1989-2017 Flexera Software LLC. All Rights Reserved.
The FlexNet host ID of this machine is "HOSTNAME=user1-laptop"
```

5.2.2 Starting the License Server

The Imperas license daemon (`imperasd`) must be run on the ‘license server’ using the Flexera license tools. The FlexNet `lmutil` tool and the Imperas daemon are provided in the `$IMPERAS_HOME/bin/$IMPERAS_ARCH` directory. The daemon can be started, to run as a background task, using the following command on the license server:

On a Windows machine in an MSYS shell

```
$ $IMPERAS_HOME/bin/$IMPERAS_ARCH/lmgrd.exe -c license.lic -l license.log
```

On a Linux machine in a shell

```
$ $IMPERAS_HOME/bin/$IMPERAS_ARCH/lmgrd -c license.lic -l license.log
```

By adding `-z` to the command line it will run in the foreground

The license server can also be run on Windows using the `LMTOOLS` utility. Additional information for getting a license daemon running on a license server can be found in APPENDIX E, “Additional FlexNet Licensing Information”.

Once you have your license server running, the computer from which the tools are run needs the `IMPERASD_LICENSE_FILE` or `LM_LICENSE_FILE` environment variable to be set.

If the license server was set up to use a specific port, as shown by the following `SERVER` line taken from a license file,

```
SERVER server1 00F346829930 2700
```

it must be specified before the '@', for example:

```
IMPERASD_LICENSE_FILE=2700@server1
```

where `server1` is the host name of the license server and `2700` is the port number that the license server is using.

If the port is not set in the license file one of the default ports (27000-27009) will be used by the license server and the `IMPERASD_LICENSE_FILE` environment variable can be set to @ followed by the host name of the license server, for example:

```
IMPERASD_LICENSE_FILE=@server1
```

NOTE

The server name, `server1`, and the port number, `2700`, in the license file may be modified. If any other entry is modified it will invalidate the license file and the server will no longer serve licenses.

5.2.3 License Queuing

The license server supports license queuing for server based license daemons. This allows waiting for licenses becoming available which is useful for batch oriented execution.

In order to enable queuing, define the environment variable `IMPERAS_QUEUE_LICENSE` as shown

```
$ export IMPERAS_QUEUE_LICENSE=1
```

6 Windows Development Environment

6.1 Introduction

The development of platforms, processor and peripheral models on the Windows operating system has been validated in an environment using MSYS2 and MINGW. A default build environment is provided with both the Imperas tools and OVPsim installations that will allow models and platforms to be built in this environment.

Linux users should go to Section 7.

Although not directly supported Cygwin and MSVC can also be supported, see section Appendix for further information.

6.2 MSYS2 / MinGW Environment

The MSYS2 / MinGW environment is a shell and GNU toolchain to use under Windows. This should be installed on a Windows 32-bit or a Windows 64-bit host machine.

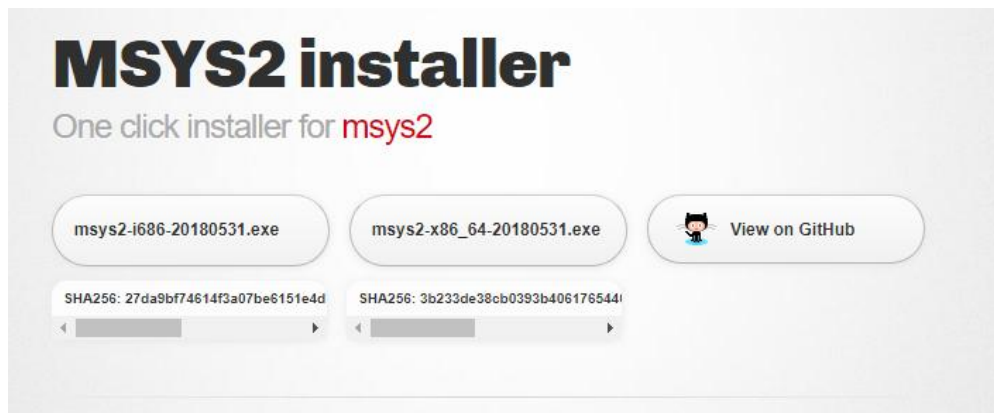
6.2.1 Obtaining MSYS2

MSYS2 can be obtained from <https://www.msys2.org/>
Any issues should be reported using the OVP Forum.

6.2.1.1 Download Installer

Download the installer from <https://www.msys2.org/> choosing either the 32-bit or 64-bit installer.

The tested version is the 20180531 release.

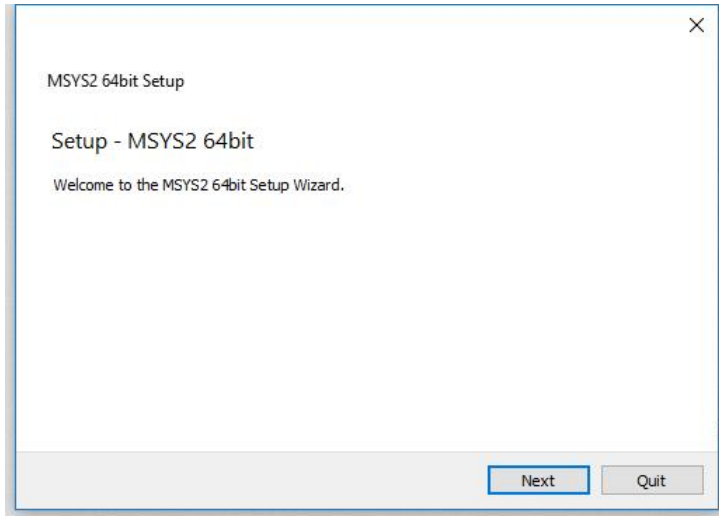


6.2.1.2 Installation of MSYS2

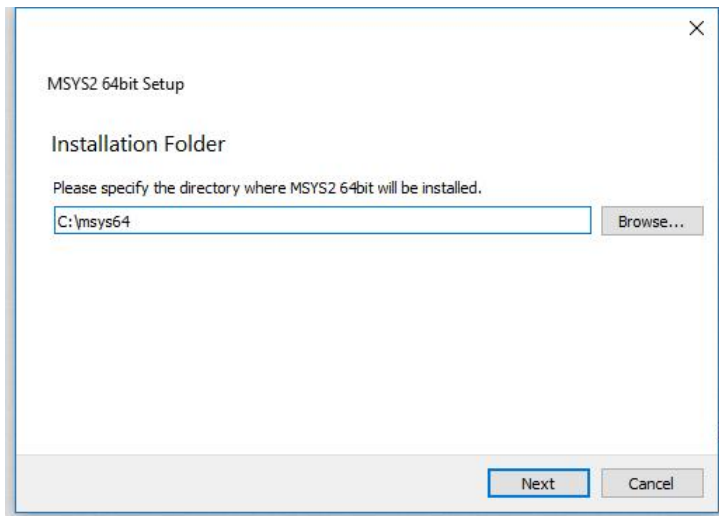
Execute the program that you downloaded.

```
msys2-x86_64-20180531.exe
```

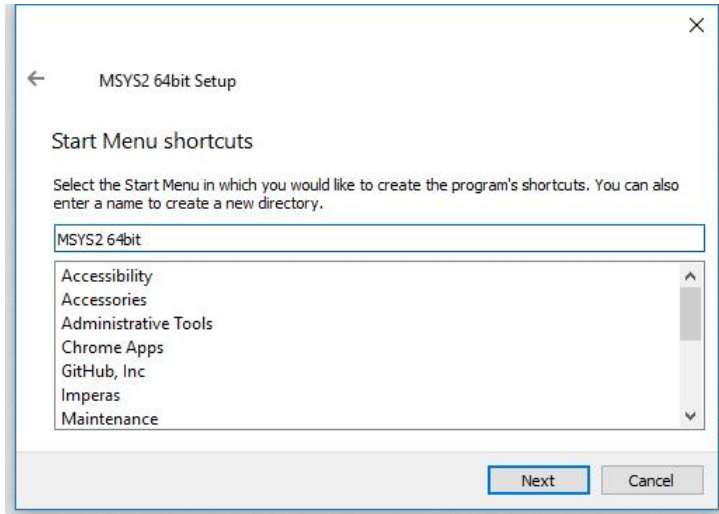
Select Next to Install



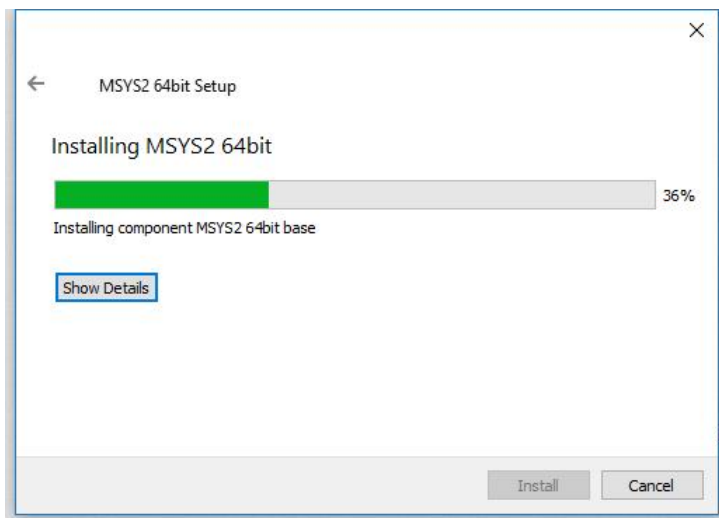
Select the Installation Folder



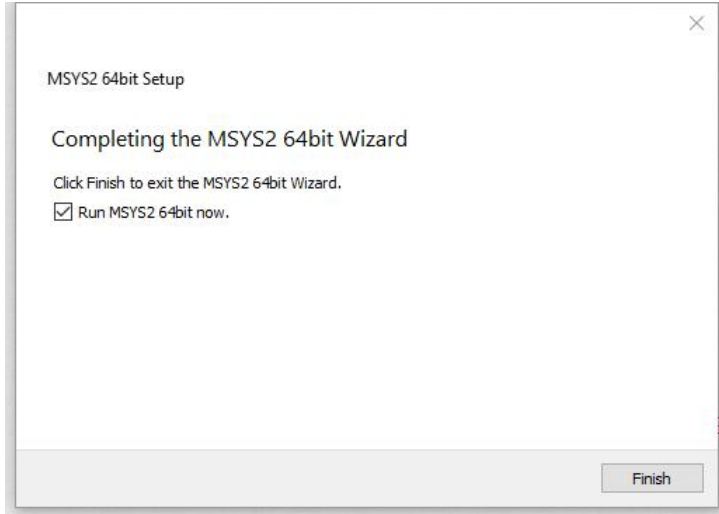
Select a name for the menu shortcut



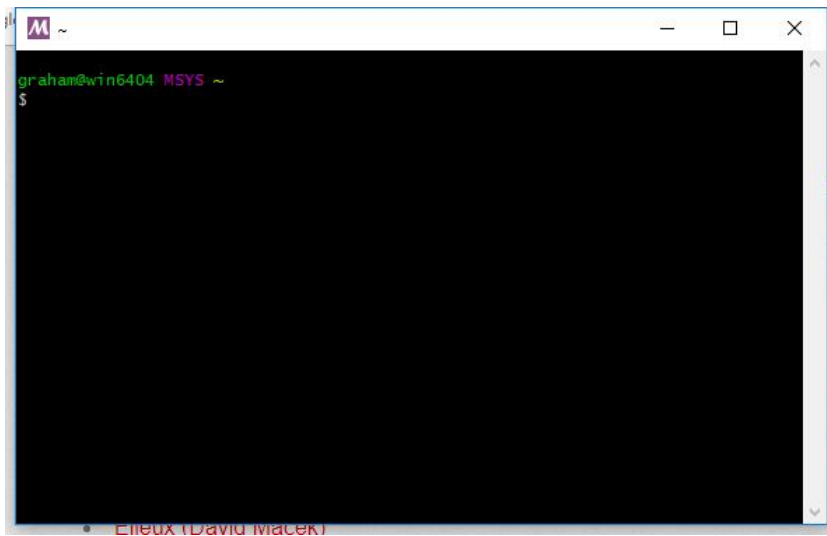
Install



Finish the installation

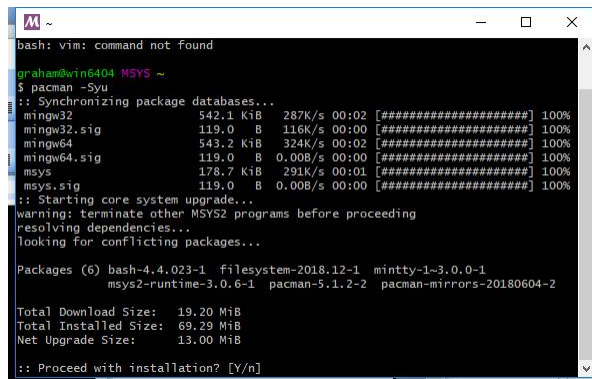


The MSYS shell should start



In the Msys Shell type the command

```
pacman -Syu
```



```
bash: vim: command not found
graham@win6404 MSYS ~
$ pacman -Syu
:: Synchronizing package databases...
mingw32             542.1 KiB   287K/s  00:02 [#####] 100%
mingw32.sig         119.0 B    116K/s  00:00 [#####] 100%
mingw64             543.2 KiB   324K/s  00:02 [#####] 100%
mingw64.sig         119.0 B    0.00B/s  00:00 [#####] 100%
msys                178.7 KiB   291K/s  00:01 [#####] 100%
msys.sig            119.0 B    0.00B/s  00:00 [#####] 100%
:: Starting core system upgrade...
warning: terminate other MSYS2 programs before proceeding
resolving dependencies...
looking for conflicting packages...

Packages (6) bash-4.4.023-1  filesystem-2018.12-1  mintty-1-3.0.0-1
               msys2-runtime-3.0.6-1  pacman-5.1.2-2  pacman-mirrors-20180604-2

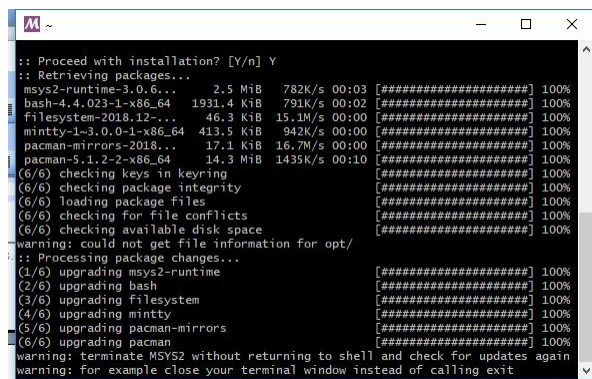
Total Download Size: 19.20 MiB
Total Installed Size: 69.29 MiB
Net Upgrade Size:     13.00 MiB

:: Proceed with installation? [Y/n]
```

Allow installation to proceed confirming ‘Y’

Terminate the shell when prompted with the messages

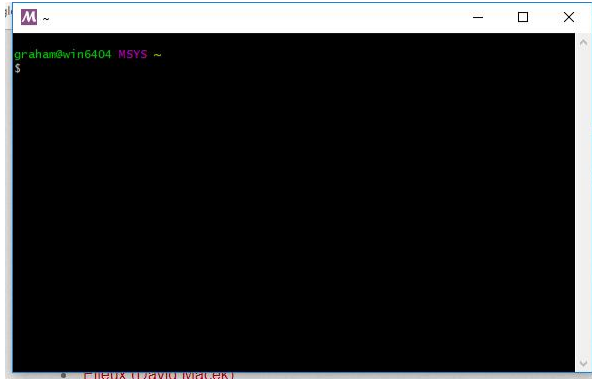
Warning: terminate MSYS2 without returning to shell and check for updates again
warning: for example close your terminal window instead of calling exit



```
:: Proceed with installation? [Y/n] Y
:: Retrieving packages...
msys2-runtime-3.0.6-1  2.5 MiB   782K/s  00:03 [#####] 100%
bash-4.4.023-1-x86_64 1931.4 KiB  791K/s  00:02 [#####] 100%
filesystem-2018.12-1  46.3 KiB   15.1M/s  00:00 [#####] 100%
mintty-1-3.0.0-1-x86_64 413.5 KiB  942K/s  00:00 [#####] 100%
pacman-mirrors-2018... 17.1 KiB   16.7M/s  00:00 [#####] 100%
pacman-5.1.2-2-x86_64 14.3 MiB  1435K/s  00:10 [#####] 100%
(6/6) checking keys in keyring [#####] 100%
(6/6) checking package integrity [#####] 100%
(6/6) loading package files [#####] 100%
(6/6) checking for file conflicts [#####] 100%
(6/6) checking available disk space [#####] 100%
warning: could not get file information for opt/
:: Processing package changes...
(1/6) upgrading msys2-runtime [#####] 100%
(2/6) upgrading bash [#####] 100%
(3/6) upgrading filesystem [#####] 100%
(4/6) upgrading mintty [#####] 100%
(5/6) upgrading pacman-mirrors [#####] 100%
(6/6) upgrading pacman [#####] 100%
warning: terminate MSYS2 without returning to shell and check for updates again
warning: for example close your terminal window instead of calling exit
```

Once the shell has been terminated, start a new shell by executing C:/msys64/msys2.exe.

This will start a new MSYS2 shell



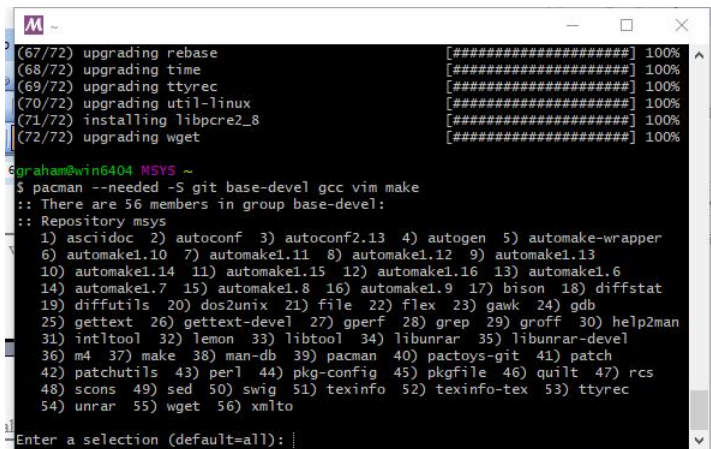
In this shell reissue the command

```
pacman -Syu
```

6.2.2 Install Utilities

Now install some common utilities.

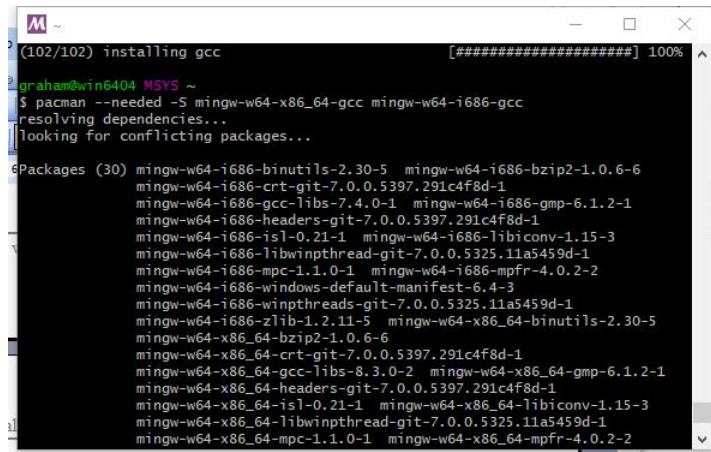
```
pacman --needed -S git base-devel gcc vim make zip unzip
```



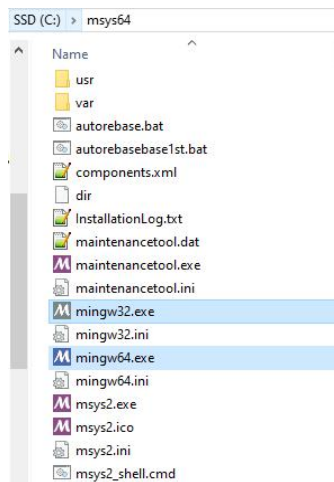
6.2.3 Install Host Toolchains

The following shows how to install both the 32-bit and 64-bit host toolchains. In practice only the one targeting your host hardware may be required.

```
pacman --needed -S mingw-w64-x86_64-gcc mingw-w64-i686-gcc
```



This will create a mingw32.exe and/or a mingw64.exe in C:\msys64 which should be used to start the MSYS shell with the correct environment to generate either 32-bit or 64-bit host executables.



These appear on the Windows toolbar or can be used to create a shortcut your desktop.

6.2.4 Environment

6.2.4.1 Inheritance Configuration

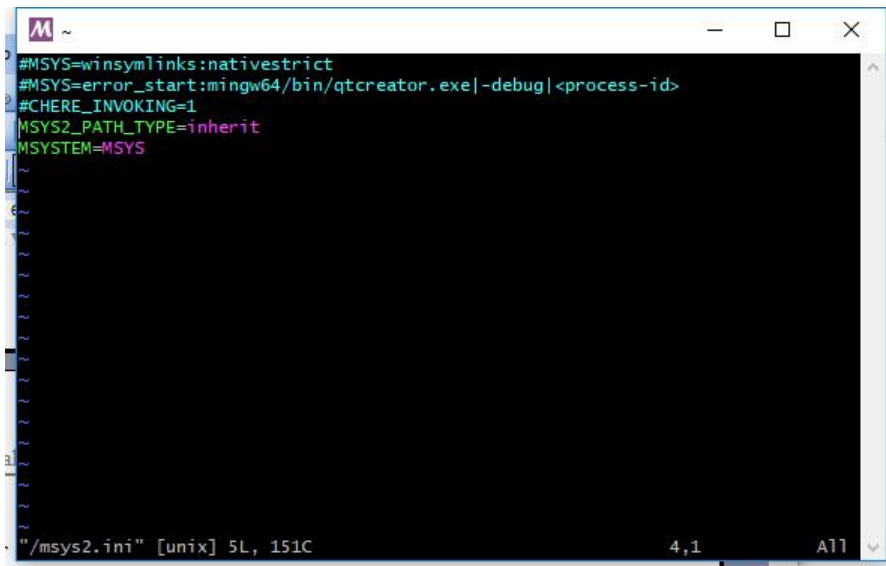
Depending upon the version and your requirements you will need to edit one or more the following files *msys2_shell.cmd*, *msys2.ini*, *mingw32.ini* and *mingw64.ini* to uncomment the following line

```
MSYS2_PATH_TYPE=inherit
```

This can be done within the MSYS2 shell using the command

```
vim /msys2_cmd.shell /msys2.ini /mingw32.ini /mingw64.ini
```

The screenshot below shows one of the files modified so that the line `MSYS2_PATH_TYPE` is no longer commented.



Alternatively, if using the start menu to start the MSYS2 shell, the properties on the start menu link can be edited to add *-use-full-path*

Open the links by right-click on the entry in the start menu and select more and then open file location. The properties on these shortcut files should be edited to add the additional command line argument.

6.2.4.2 Update make for OVP Scripts

Some OVP scripts make calls to `mingw32-make` instead of `make`. To accommodate this a copy and rename of `make.exe` should be made.

```
cd /c/msys64/usr/bin
cp make.exe mingw32-make.exe
```

Check that the OVP scripts using mingw32-make are working by creating a batch file “test.bat” containing

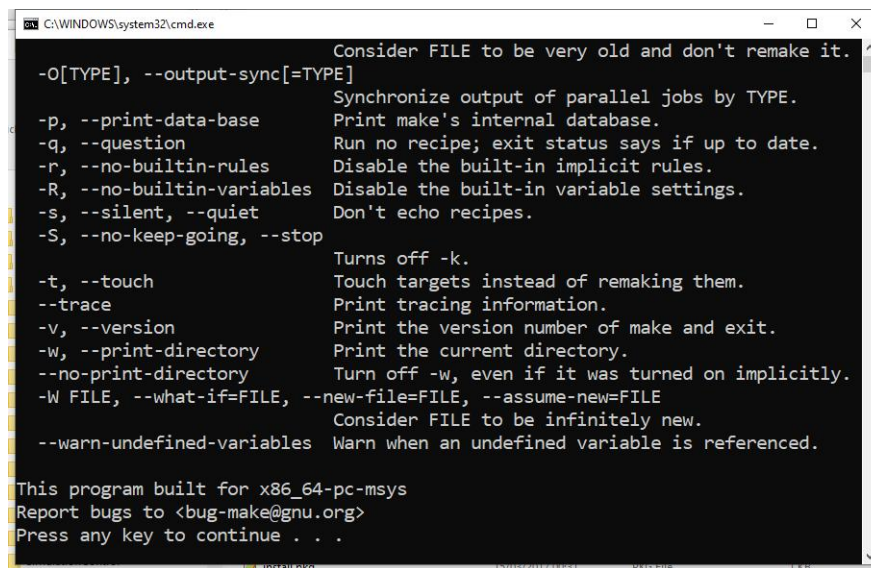
```
@echo off

;rem move into the Example Directory
set BATCHDIR=%~dp0%
cd /d %BATCHDIR%

mingw32-make -help

pause
```

In a Windows explorer double click the test.bat icon and ensure you get a cmd shell with the following help output



```
Consider FILE to be very old and don't remake it.
-O[TYPE], --output-sync[=TYPE]      Synchronize output of parallel jobs by TYPE.
-p, --print-data-base               Print make's internal database.
-q, --question                       Run no recipe; exit status says if up to date.
-r, --no-builtin-rules              Disable the built-in implicit rules.
-R, --no-builtin-variables          Disable the built-in variable settings.
-s, --silent, --quiet               Don't echo recipes.
-S, --no-keep-going, --stop         Turns off -k.
-t, --touch                         Touch targets instead of remaking them.
--trace                             Print tracing information.
-v, --version                       Print the version number of make and exit.
-w, --print-directory               Print the current directory.
--no-print-directory               Turn off -w, even if it was turned on implicitly.
-W FILE, --what-if=FILE, --new-file=FILE, --assume-new=FILE
                                   Consider FILE to be infinitely new.
--warn-undefined-variables          Warn when an undefined variable is referenced.

This program built for x86_64-pc-msys
Report bugs to <bug-make@gnu.org>
Press any key to continue . . .
```

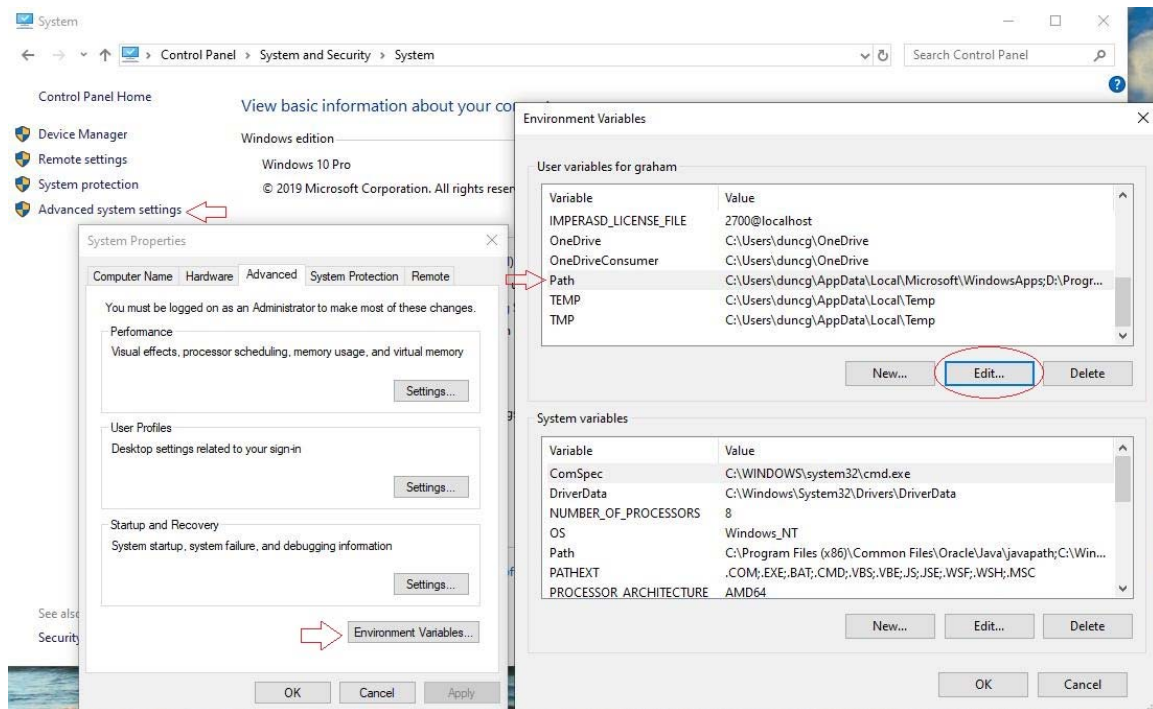
6.2.4.3 Update Windows PATH

Some OVP scripts build applications and platforms, using make and the native toolchains installed previously. In order for these to be found the entries must be made to the Windows PATH.

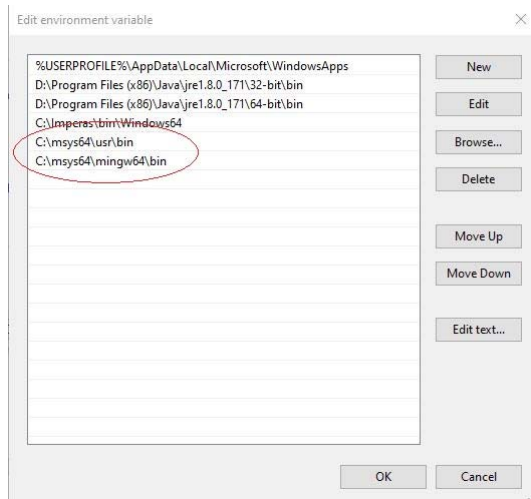
The make application is found in the MSYS installation at C:\msys64\usr\bin so this should be added to the PATH.

The native toolchains are found at either C:\msys64\mingw32\bin or C:\msys64\mingw64\bin. Only one of these should be added to the PATH.

To add to the Windows environment PATH open *Control Panel*, select *System and Security*, then select *System* and finally *Advanced System Setting* and *Environment Variables* as shown below



Edit the Path environment variable and browse to the directories *user/bin* and *mingw64/bin* or *mingw32/bin*, depending upon host, as shown



6.2.4.4 Stop PATH expansion

MSYS2 performs path expansion in scripts when executed. This can result in bad paths. As an example, the following script entry is taken from one of those provided in the harness directory of the Demo *Demo/Platforms/riscv_RV64_Virtio_Linux*

```
--override virtio/smartLoader/command=\"root=/dev/vda ro console=ttyS0\" \
```

When the shell script, RUN_Virtio_Linux.sh, that contains this line, is executed in an MSYS2 shell the Linux kernel does not boot as expected but instead a kernel panic is caused.

```
[ 1.120000] VFS: Cannot open root device "C:/msys64/dev/vda" or unknown-  
block(0,0): error -6  
[ 1.120000] Please append a correct "root=" boot option; here are the available  
partitions:  
[ 1.120000] fe00 65536 vda  
[ 1.120000] driver: virtio_blk  
[ 1.120000] Kernel panic - not syncing: VFS: Unable to mount root fs on  
unknown-block(0,0)
```

As can be seen in the above log output, this is because /dev/vda has been expanded in the shell to C:/msys64/dev/vda, where the MSYS2 installation is located at C:/msys64.

To stop this from happening the command below must be executed in the shell prior to executing any scripts.

```
$ export MSYS2_ARG_CONV_EXCL="*"
```


6.2.5 Completion and test of MSYS/MinGW installation

With any of the product packages (OVPsim, Imperas_SDK or Imperas_DEV) installed and the riscv.toolchain cross compiler package installed you can try running the HelloWorld example usingOP found in the installation under Imperas/Examples/HelloWorld.

Start an MSYS shell using either the mingw32 or mingw64 shortcut created previously.

Below we show a valid execution of the script example.sh in the MSYS shell

```
$ cd /c/Imperas/Examples/HelloWorld/usingOP
$ ./example.sh
make: Entering directory `/c/Imperas/Examples/HelloWorld/usingOP/application'
Compiling application.c
Linking application.RISCV32.elf
rm application.o
make: Leaving directory `/c/Imperas/Examples/HelloWorld/usingOP/application'
make: Entering directory `/c/Imperas/Examples/HelloWorld/usingOP/platform'
# Host Depending obj/Windows64/platform.d
make: Leaving directory `/c/Imperas/Examples/HelloWorld/usingOP/platform'
make: Entering directory `/c/Imperas/Examples/HelloWorld/usingOP/platform'
# Host Compiling Platform obj/Windows64/platform.o
# Host Linking Platform platform.Windows64.exe
# Host Linking Platform object model.dll
make: Leaving directory `/c/Imperas/Examples/HelloWorld/usingOP/platform'

CpuManagerMulti (64-Bit) v20190225.0 Open Virtual Platform simulator from
www.IMPERAS.com.
Copyright (c) 2005-2019 Imperas Software Ltd.  Contains Imperas Proprietary
Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

CpuManagerMulti started: Fri Mar 15 14:21:33 2019

Info (RISCV_ANSI) Attribute 'variant' not specified; defaulting to 'RV32I'
Info (OR_OF) Target 'test/CPU1' has object file read from
'application/application.RISCV32.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type          Offset      VirtAddr    PhysAddr    FileSiz    MemSiz
Flags Align
Info (OR_PD) LOAD          0x00000000 0x00010000 0x00010000 0x000039a4
0x000039a4 R-E 1000
Info (OR_PD) LOAD          0x00004000 0x00014000 0x00014000 0x00000854
0x000008d0 RW- 1000
Hello World
Info
Info -----
Info CPU 'test/CPU1' STATISTICS
Info Type                  : riscv
Info Nominal MIPS          : 100
Info Final program counter : 0x100ac
Info Simulated instructions: 1,815
Info Simulated MIPS        : run too short for meaningful result
```



```
Info -----
Info
Info -----
Info SIMULATION TIME STATISTICS
Info   Simulated time       : 0.00 seconds
Info   User time           : 0.00 seconds
Info   System time         : 0.00 seconds
Info   Elapsed time        : 0.00 seconds
Info -----

CpuManagerMulti finished: Fri Mar 15 14:21:33 2019

CpuManagerMulti (64-Bit) v20190225.0 Open Virtual Platform simulator from
www.IMPERAS.com.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.
```

Application Build

To build the application we must have the appropriate Cross Compiler toolchain installed. By default, this example sets `CROSS=RISCV32` and so we must have installed the package `riscv.toolchain`.

If this is missing from the installation expect to see an error of the form:

```
$ ./example.sh
Fatal (CHECK_PF) One or more of packages from line 'install riscv.toolchain'
are required, but are not installed or selected. Please check Installation and
Environment.
Info Exiting
```

Platform Build

To build the platform we use the standard Makefile, `Makefile.platform`, that is provided, with other standard Makefiles, in `IMPERAS_HOME/ImperasLib/buildutils`. This requires that we have the expected host gcc toolchains installed.

If this is missing from the host `PATH` environment variable expect to see an error of the form:

```
$ cd platform
$ make
# Host Depending obj/Windows64/platform.d
make: x86_64-w64-mingw32-gcc: Command not found
# Host Compiling Platform obj/Windows64/platform.o
make: x86_64-w64-mingw32-gcc: Command not found
make: *** [obj/Windows64/platform.o] Error 127
```

As you can see the platform cannot be built because the host GCC cannot be found (this is `x86_64-w64-mingw32-gcc` as we are running on a 64-Bit Windows host machine). We need to setup the `PATH` environment variable to include the host toolchain.

We can setup the `PATH` environment variable in the shell, shown below the addition of the bin directory for the 64-bit Host GCC to the `PATH`

```
export PATH=$PATH:/c/msys64/mingw64/bin
```

however, this is only for this shell so it is better to add into the system environment (see section *Setting Environment Variables* in an Appendix of this document).

We can now re-invoke the same make command and build the platform

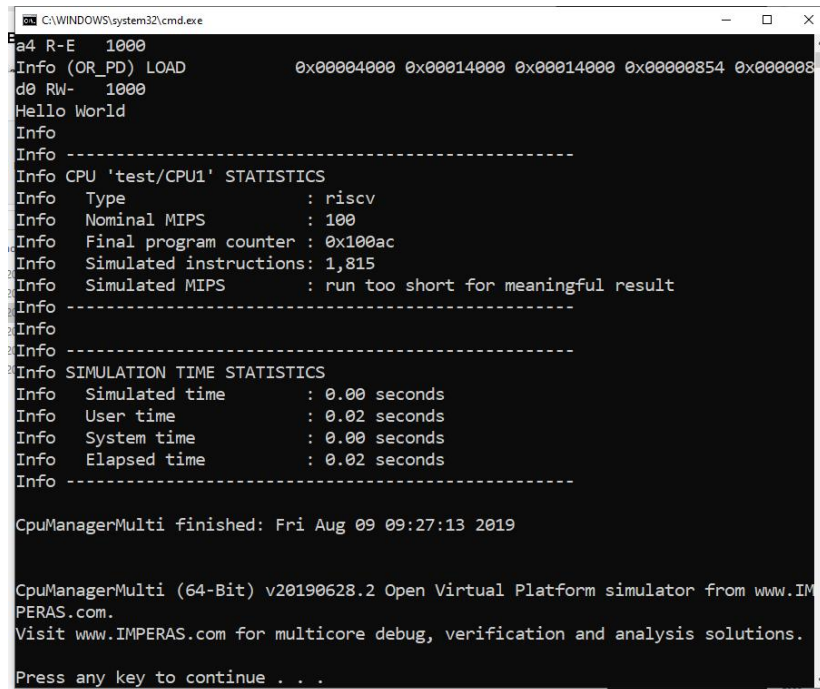
```
$ make
# Host Depending obj/Windows64/platform.d
# Host Compiling Platform obj/Windows64/platform.o
# Host Linking Platform platform.Windows64.exe
# Host Linking Platform object model.dll
```

6.2.5.1 Test Windows Batch File Execution

To test the execution from a Windows batch file you can try running the HelloWorld example usingOP found in the installation under Imperas/Examples/HelloWorld.

Open this directory in a Windows explorer and double-click on the script example.bat

A cmd shell should start and the example execute



```
C:\WINDOWS\system32\cmd.exe
a4 R-E 1000
Info (OR_PD) LOAD      0x00004000 0x00014000 0x00014000 0x00000854 0x00000854
d0 RW- 1000
Hello World
Info
Info -----
Info CPU 'test/CPU1' STATISTICS
Info Type           : riscv
Info Nominal MIPS   : 100
Info Final program counter : 0x100ac
Info Simulated instructions: 1,815
Info Simulated MIPS   : run too short for meaningful result
Info -----
Info
Info -----
Info SIMULATION TIME STATISTICS
Info Simulated time   : 0.00 seconds
Info User time        : 0.02 seconds
Info System time      : 0.00 seconds
Info Elapsed time     : 0.02 seconds
Info -----
CpuManagerMulti finished: Fri Aug 09 09:27:13 2019

CpuManagerMulti (64-Bit) v20190628.2 Open Virtual Platform simulator from www.IMPERAS.com.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.
Press any key to continue . . .
```

7 Additional Toolchain Packages

Additional Cross Compilers are provided as Linux or Windows installers.

These are available from the OVP website, www.OVPworld.org and the user area of the Imperas website¹⁰, <http://www.imperas.com/>

IMPORTANT

The Cross Compiler and Peripheral Simulation Engine toolchains are provided only as 32-bit native executables. This requires that a 64-bit host must provide a 32-bit compatibility mode in order that they can execute. The packages they are provided in are labeled as 64-bit packages, for example Linux64 only to indicate that they support a 64-bit Imperas or OVPsim installation.

7.1 Application Cross Compiler Toolchains

Imperas provides example pre-built toolchains for creating elf files from application source code to load and execute on the processors provided by OVP.

The toolchains are available to download at www.OVPworld.org and from the user area of the Imperas website, www.imperas.com.

NOTE

The Cross Compiler toolchains are provided to allow a user to quickly start generating application binary code that can be executed on OVP processor models. It is expected that after the initial experiments a cross compiler toolchain will be selected from a cross compiler supplier. Some processors and processor variants are not supported by the cross compiler toolchains available on the OVPWorld website and alternatives are indicated to download and install.

Provided as part of these cross compiler toolchains are Makefiles to provide a default application build environment for specific processor architectures or variants, for example RISCv32, MIPS32R6, ARM7TDMI, V850.

The installed toolchains and the Makefiles are found in the directory

`IMPERAS_HOME/lib/IMPERAS_ARCH/CrossCompiler`

The Makefiles are named using the format

`<Processor Type/Variant>.makefile.include`,

¹⁰ The Imperas website provides the same installers but also for 64-bit host computers.

for example, `MIPS32.Makefile.include`.

Each cross compiler toolchain should be installed into the current installation of Imperas Professional or OVPSim.

A Makefile created to build an application elf file should include an appropriate `<NAME>.makefile.include` and it is typical to use a variable `CROSS` to define the one to be used. The Makefile will use the lines shown below to do this

```
CROSS?=OR1K
-include
$(IMPERAS_HOME)/lib/$(IMPERAS_ARCH)/CrossCompiler/$(CROSS).makefile.include
```

The included Makefile defines a number of variables that should be used in the application Makefile build rules.

There are generic variables that are created by all the Makefiles

```
IMPERAS_CC
IMPERAS_LINK
```

And there are specific variables created by only this Makefile, for example

```
ARM7TDMI_CC
ARM7TDMI_LINK
```

The use of the specific named variables allow more than included Makefile to be used.

The `CROSS` variable can be used to define these in a generic Makefile as

```
$(CROSS)_CC
$(CROSS)_LINK
```

The final generic application Makefile would become

```
IMPERAS_HOME := $(shell getpath.exe "$(IMPERAS_HOME)")
include $(IMPERAS_HOME)/bin/Makefile.include

#
# Makefile for Cross Compiling an application for a target processor
# type
#

# Various Cross compiler setups, Default or1k
CROSS?=OR1K
-include
$(IMPERAS_HOME)/lib/$(IMPERAS_ARCH)/CrossCompiler/$(CROSS).makefile.inc
lude
ifeq ($(CROSS)_CC,)
    IMPERAS_ERROR := $(error "Please install the toolchain to support
$(CROSS)")
```

```
endif

SRC ?= application.c
EXE = $(patsubst %.c,%.${CROSS}.elf,$(SRC))

all: $(EXE)

%.${CROSS}.elf: %.o
    $(V) echo "Linking $@"
    $(V) $(IMPERAS_LINK) -o $@ $< $(IMPERAS_LDFLAGS) -lm

%.o: %.c
    $(V) echo "Compiling $<"
    $(V) $(IMPERAS_CC) -g -O0 -c -o $@ $<

clean:
    - rm -f *.elf *.o
```

By default, this Makefile will build an elf file `application.OR1K.elf` from a source file `application.c`.

To build for a different target we can specify the CROSS variable

```
$ make CROSS=ARM7TDMI
```

To build different source we can specify the SRC variable

```
$ make CROSS=MIPS32R2 SRC=hello.c
```

7.2 Peripheral Simulation Engine (PSE) Toolchain

The Peripheral Simulation Engine (PSE) is used to execute behavioral code that provides peripheral models in a virtual platform simulation.

A toolchain is provided to allow the behavioral code to be compiled for the PSE.

This toolchain should be installed into the current installation of Imperas tools or OVPsim and so will also appear in `IMPERAS_HOME/lib/IMPERAS_ARCH/CrossCompiler` and provides the toolchain *pse-elf* and a Makefile *PSE.makefile.include*.

The Makefile in a peripheral model source directory should include the pre-defined Makefile for building PSE behavioral code, `Makefile.pse`, as shown below

```
IMPERAS_HOME := $(shell getpath.exe "$(IMPERAS_HOME)")
include $(IMPERAS_HOME)/ImperasLib/buildutils/Makefile.pse
```

8 Getting Started

This chapter will look at the basics of using the product installed in the previous sections.

The following should be carried out in a Linux shell or on Windows in an MSYS shell; some scripts (.bat files) may be executed on Windows by double clicking on them in an explorer browser.

8.1 ISS Tutorial Video

Detailed information on the usage of the ISS may be found by watching the tutorial video available when logged into the OVPWorld website at

<http://www.ovpworld.org/using-the-imperas-instruction-set-simulator-iss>

8.2 Check Installation

This section provides some checks that the tools have been correctly installed.

8.2.1 Simulator Execution

To verify the installation after an OVPsim or an Imperas install one of the Demos that are downloaded in both installations can be executed. These Demos will work with both the OVPsim and the Imperas Simulators.

The setting of the environment variable `IMPERAS_RUNTIME` to *OVPsim* or *CpuManager* will select between OVPsim or the Imperas professional simulator respectively.

In the Processor demos we can execute an application cross compiled for a processor type using the ISS. The ISS allows the definition of a virtual platform containing processor(s) and memory, the loading of application elf file(s) and controlling the simulator execution. There is more information regarding the ISS usage in section 8.4.2 Running the simulation using the ISS.

Execute the Fibonacci benchmark on a single core RISC-V platform.

Change to the demo directory.

```
$ cd $IMPERAS_HOME/Demo/Processors/RISCV/riscv32/RV32IMAC/single_core/
```

Run one of the execution scripts, for example

```
$ Run_Fibonacci.sh
```

If Windows explorer is used, change to the directory and double-click the corresponding .bat file.

This should provide an output similar to that shown below (this is an OVPSim installation, for an Imperas installation the output will be the same but with a different banner):

```

IMPERAS Instruction Set Simulator (ISS)

OVPSim (64-Bit) v20180221.0 Open Virtual Platform simulator from www.OVPworld.org.
Copyright (c) 2005-2018 Imperas Software Ltd.  Contains Imperas Proprietary Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

OVPSim started: Tue Mar 20 09:52:14 2018

Info (OR_OF) Target 'iss/cpu0' has object file read from
'../../../../Applications/fibonacci/fibonacci.RISCV32-00-g.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type          Offset      VirtAddr   PhysAddr   FileSiz   MemSiz     Flags
Align
Info (OR_PD) LOAD          0x00000000 0x00010000 0x00010000 0x00016998 0x00016998 R-E
1000
Info (OR_PD) LOAD          0x00017000 0x00027000 0x00027000 0x000009c0 0x00000a24 RW-
1000
starting fib(39)...
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
....
fib(33) = 3524578
fib(34) = 5702887
fib(35) = 9227465
fib(36) = 14930352
fib(37) = 24157817
fib(38) = 39088169
finishing...
Info
Info -----
Info CPU 'iss/cpu0' STATISTICS
Info   Type                : riscv (RV32IMAC)
Info   Nominal MIPS         : 100
Info   Final program counter : 0x100ac
Info   Simulated instructions: 7,120,123,071
Info   Simulated MIPS       : 1397.8
Info -----
Info
Info -----
Info SIMULATION TIME STATISTICS
Info   Simulated time       : 71.20 seconds
Info   User time            : 5.09 seconds
Info   System time         : 0.00 seconds
Info   Elapsed time        : 5.11 seconds
Info   Real time ratio      : 13.94x faster
Info -----
OVPSim finished: Tue Mar 20 09:52:15 2018

OVPSim (64-Bit) v20180221.0 Open Virtual Platform simulator from www.OVPworld.org.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

```

NOTE

The Imperas ISS used in this example is provided in two different executable programs; `iss.exe` and `issdemo.exe`. The demo scripts use the environment variable `IMPERAS_ISS` that selects which to use. The environment variable is set up as part of the installation scripts. `iss.exe` uses the default OVPsim or Imperas simulator license. `issdemo.exe` does not require a license but does require access to the internet.

The `issdemo.exe` executable is only available to run for a specific number of days after the release is built.

If you see the following it indicates that `issdemo.exe` is being executed and the remaining days for which it will execute.

```
Info (PER_DDR) 77 Demo License days remain
```

If the number of days is exceeded the simulator will not run and the following message will be displayed.

```
Fatal (PER_TMO) License has expired.  
Please contact license@imperas.com to renew  
Info Exiting
```

In this circumstance either:

- change the environment variable `IMPERAS_ISS` to `iss.exe`, if you have a current and valid OVPsim or Imperas license
- update the version of product to a later version, a new release is typically made each quarter

8.2.2 iGen Installation

The iGen productivity tool `igen.exe` is used to simplify the generation of hardware definitions for virtual platforms (modules) and processor and peripheral model templates.

You can check that it is correctly installed by typing:

```
$ igen.exe --version
```

This should provide the version information, for example *20150901.0*


```
$ igen.exe --help
```

This should start iGen with the help output similar to that shown below:

flag	short argument	description
diagnostics		
--apropos	command	Show igen commands similar to the given argument
--help	-h	Print list of flags
--showcommands		Show all igen commands
input		
--batch	-b filename	Execute this tcl file
--batchargv	argument	Argument to --batch file
--checkmodels		Load and check models when writing a platform
--modellibrary	string	Processor VLNv library
--modelname	string	Processor VLNv name
--modelvendor	string	Processor VLNv vendor
...		

8.2.3 Harness Installation

A utility program `harness.exe` is provided to easily load a hardware definition (module) and to load and control the execution of applications on processors within the module.

You can check that it is correctly installed by typing:

```
$ harness.exe --version
```

This should provide the version information, for example 20150901.0

And an list of arguments supported (abbreviated below) may be obtained using the help argument as shown:

```
$ harness.exe --help
flag          short argument      description
control
  --callcommand      strings          Call a command in a plugin. eg.
des/plugin/cmd arg1 arg2
  --controlfile      -C filename      Read a control file
  --enabletools      [processors]    Load VAP tools
  --extlib           string          Add an extension library eg.
des/instance=/v/l/n/v
  --finishafter      -I string        Finish simulation after this many
instructions. Eg. 1000 or des/inst=1000
  --finishtime       -F string        Finish simulation at this time
debug
  --gdbconsole       [rootmodule]    Pop up gdb(s) in console window(s)
  --gdbgui           [rootmodule]    Start gdb debug in Eclipse (eGui)
  --port             strings         Open this port number to allow a
connection to a GDB using RSP
diagnostics
  --monitornets      [rootmodule]    Monitor nets
  --showbuses        [root]          Show all (static) bus connections
  --showbusses       [root]          Alias for showbuses
  --showcommands     [root]          Show commands that can be called
with --callcommand
  --showload         [root]          Show where each model is loaded from
harness
  --modulefile       string          Path to a module
  --modulelibrary     -L string       Module library (defaults to
'module')
  --modulename       -N string        Module name
  --modulevendor      -V string        Module vendor
  --moduleversion     -S string        Module version (defaults to '1.0')
  --parameter        name=value      Module model parameters
```

8.2.4 MPD Installation

MPD is only available as part of the Imperas professional installation.

MPD is the Multicore/Multi-Processor debug and interactive control program to allow the interactive debug of applications on all processor models and the debug and development of the behavioral code of peripheral models in a unified environment.

You can check that it is correctly installed by typing:

```
$ mpd.exe --version
```

This should provide the version information, for example *20150901.0*

And an list of arguments supported (abbreviated below) may be obtained using the help argument as shown:

```
$ mpd.exe --help
flag          short argument  description
mpd
--apropos      command        Show mpd TCL commands similar to the given
argument
--batch        -b filename    Execute tcl file(s) in batch mode
...<snip>...
--port         -p integer     Simulator port number to connect
--processordebugbasic string      Do not use gdb with this processor
--processorexex -e string      Define the executable for one or more
processors
--processorgdb -g string      Define the gdb path for one or more processors
--processorgdbflags -f string    Define special flags required by a gdb
--searchpath   string        Add a path to search for source files
--showcommands Show all mpd TCL commands
--simulatorcwd Change to the same working directory as the
simulator
```

The MPD debug console can be started from the simulator command line.

Execute and debug the Fibonacci benchmark on a single core OR1K platform using MPD

Change to the demo directory.

```
$ cd $IMPERAS_HOME/Demo/Processors/RISCV/riscv32/RV32IMAC/single_core/
```

Run the execution script with the additional command line argument *--mpdconsole*

```
$ Run_Fibonacci.sh --mpdconsole
```

The simulation will start up as previously (this is CpuManager the Imperas professional simulator to support MPD) but the additional output will be observed showing the connection to the MPD debugger:

```
CpuManagerMulti (64-Bit) v20180221.0 Open Virtual Platform simulator from
www.IMPERAS.com.
```

```
..
```

```
Info (GDBT_PORT) Host: HOST, Port: 51701
```

```
Info (GDBT_WAIT) Waiting for remote debugger to connect...
```

```
..
```

```
Info (GDBT_MPD) Client connected to platform
```

And a console will be started executing MPD

```
MPD (64-Bit) 20180221.0 Multiprocessor debugger from www.IMPERAS.com.
```

```
Copyright (c) 2005-2018 Imperas Software Ltd.
```

```
Contains Imperas Proprietary Information.
```

```
Licensed Software, All Rights Reserved.
```

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

```
Info (MPD_SCS) Connecting
```

```
Info (MPD_SC) Socket connected
```

```
Info (MPD_VC) Server is compatible
idebug (cpu0) >
```

This now allows us to perform debugging of the application in the simulation platform.

8.2.5 eGui Installation

The package eGui_Eclipse must be installed to provide the Eclipse GUI, eGui, that can be used with either OVPSim or Imperas professional simulations.

eGui can be used to control the virtual platform simulations and the debugging of cross compiled applications and peripheral model behavioral code (when used with MPD).

The eGui is invoked from the simulator command line using the arguments `--gdbgui` or `--mpdegui`.

Execute and debug the Fibonacci benchmark on a single core OR1K platform using eGui

Change to the demo directory.

```
$ cd $IMPERAS_HOME/Demo/Processors/RISCV/riscv32/RV32IMAC/single_core
```

Run the execution script with the additional command line argument `--gdbgui` for an OVPSim installation or `--mpdegui` if you are using an Imperas professional installation.

```
$ Run_Fibonacci.sh --mpdegui
```

If the eGui installation has not been completed correctly you may see the following error message. In this case please check that the eGui_Eclipse package has been installed into the same directory as the product installation

```
IMPERAS Instruction Set Simulator (ISS)
```

```
..

```

```
Info (GDBT_PORT) Host: HOST, Port: 51747
```

```
Fatal (DBC_FNF) EGUI was not found at 'C:\Imperas\bin\Windows64\egui.exe'. Please
check your installation.
```

```
Info Exiting
```

With a correct installation of eGui, the simulation will start up as previously (this is CpuManager the Imperas professional simulator to support MPD):

```
CpuManagerMulti (64-Bit) v20180221.0 Open Virtual Platform simulator from
www.IMPERAS.com.
```

```
..

```

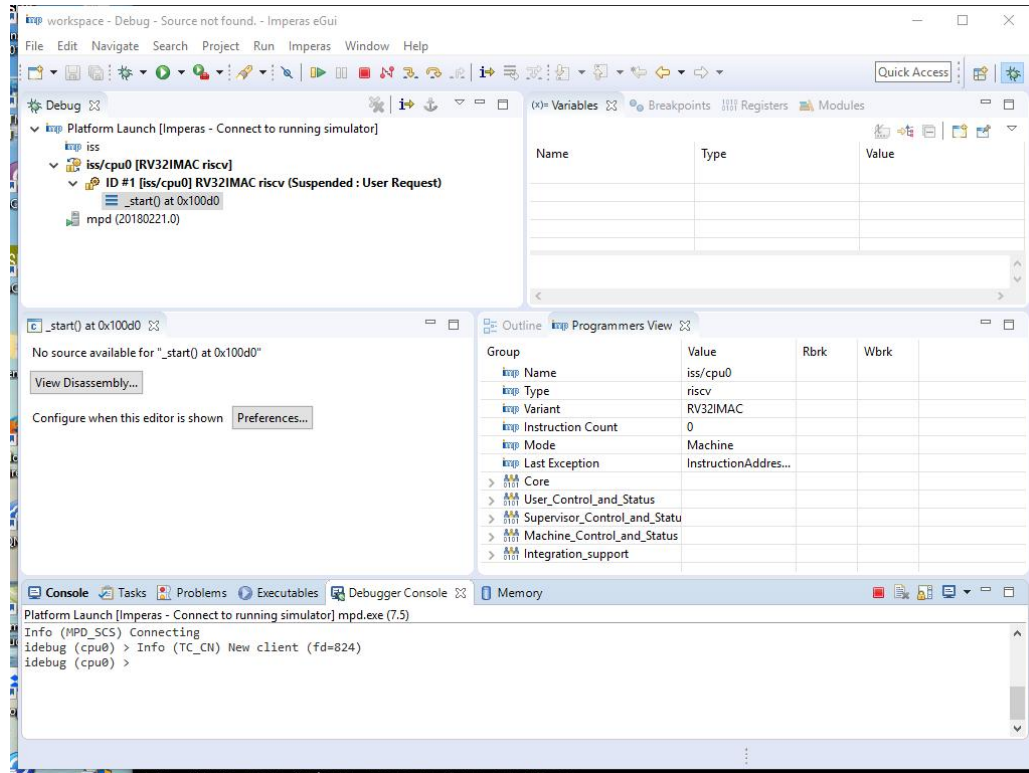
```
Info (GDBT_PORT) Host: HOST, Port: 51701
```

```
Info (GDBT_WAIT) Waiting for remote debugger to connect...
```

```
..<snip>..
```

Info (GDBT_MPD) Client connected to platform

And an Eclipse session will be started



This now allows us to perform debugging of the application in the simulation platform.

8.3 Build Environment

8.3.1 Introduction

Windows and Unix present a number of challenges to engineers in the different ways file structure paths are referenced, in particular the ‘\’ (backslash) versus ‘/’ (slash) differences, and the differences regarding the use of a ‘ ’ (space) in a name; in Unix a space is generally treated as a separator, whereas on Windows it can form part of a filename without the need to be escaped.

To help overcome these issues, a number of utilities and Makefiles have been created which will help in the building of Modules, Processors and Peripherals.

Look at the file Imperas/Examples/HelloWorld/usingOP/application/Makefile

The first lines of the Makefile:-

```
ifndef IMPERAS_HOME
    IMPERAS_ERROR := $(error "IMPERAS_HOME not defined, please setup Imperas/OVP
environment")
endif
IMPERAS_HOME := $(shell getpath.exe "$(IMPERAS_HOME)")
```

This line will modify a Windows long filename format, to the short filename format, so that in the Makefile, the default value of IMPERAS_HOME is modified from

```
IMPERAS_HOME = C:\Program Files\Imperas
to
IMPERAS_HOME = C:/PROGRA~1/Imperas
```

This removes the space separator which would otherwise cause a problem for `make`.

NOTE

It is recommended to install in a location that does not contain a space, for example the default installation for Windows is C:\Imperas

8.3.2 Standard Makefiles

There are a number of Makefiles provided in the directory `ImperasLib/buildutils` that should be used to build.

For individual components:

<code>Makefile.harness</code>	: for building an executable from an <code>OP_harness</code> C file that loads and configures the hardware module and controls the simulation
<code>Makefile.module</code>	: for building hierarchical hardware definition as native host shared objects from an iGen input file and/or C source files. These could be combinations of processor and peripheral models and include other modules.
<code>Makefile.pse</code>	: for building a <code>pse.pse</code> peripheral model behavioral executable from an iGen input file and/or C source files
<code>Makefile.host</code>	: for building native host model shared objects such as processor models or intercept and extension libraries.
<code>Makefile.TLM.igen</code>	: for generating a SystemC TLM2 platform definition file from an iGen input file
<code>Makefile.TLM.platform</code>	: for building an executable from a SystemC TLM2 platform definition file
<code>Makefile.platform</code>	: for building an executable from a platform definition file
<code>Makefile.extlib</code>	: for building native host model intercept and extension libraries that include iGen source files (use <code>Makefile.host</code> if C source files only).

For building a library:

`Makefile.library` : this may be used to build the entire VLNV component library

Each component directory should have a Makefile that includes one of the above standard Makefiles, for example a peripheral model Makefile would be

```
IMPERAS_HOME := $(shell getpath.exe "$(IMPERAS_HOME)")
include $(IMPERAS_HOME)/ImperasLib/buildutils/Makefile.pse
```

8.3.3 Build Output Selection

When a component is built using one of the standard Makefiles the target for the binary can be controlled by setting one of the following variables to 1.

<code>NOVLNV</code>	: Build into the current directory. This is typically only used for examples.
<code>VLNVROOT</code>	: Build into the VLNV library, the root of which is specified by this variable

SYSTEMVLNV : Build into the product VLNV. Care should be used when selecting this as it is modifying the installation. Typically, this will only be used if a patch update for a model is provided.

For example, in a directory containing peripheral model source code we can build the output binary in the current directory with the command

```
$ make NOVLNV=1
```

8.3.4 VLNV Component Library

Components and hardware definitions are generally stored in a VLNV¹¹ library. This comprises a source library and an output binary library from which they are used by the simulator.

In the examples, the components may be local and compiled into the same directory for ease of use.

It is recommended that changes are not made in the provided VLNV library. If new components are to be created or an existing component is to be modified for a different purpose it is recommended that a user VLNV library is created in which the component is given a new unique VLNV reference.

8.3.4.1 Creating a User Component Library

Create a new local VLNV library structure to contain our new models. It is always recommended to work outside of an OVP or Imperas product installation.

For example, this can be accomplished on a Linux host or in an MSYS shell on a Windows host using the following commands to create a new library at *myLocalLib/source*.

```
mkdir -p myLocalLib/source/vendor.com/processor/riscv/1.0/model
cp $IMPERAS_HOME/ImperasLib/source/Makefile myLocalLib/source
```

The Vendor entry should use a company specific name in place of *vendor.com* above.

We can now create new models within the library VLNV structure, some typical structures for peripheral models and for hardware modules are shown below. Other library types include, platform and processor.

Peripherals

PSE	vendor.com/peripheral/peripheralName/1.0/pse
Native	vendor.com/peripheral/peripheralName/1.0/model

¹¹ VLNV is Vendor Library Name Version and provides for a library directory structure of the form Vendor/Library/Name/Version with the source library located ImperasLib/source and the output binary library located at lib/\$IMPERAS_ARCH/ImperasLib

Hardware Definition Module

Module vendor.com/module/moduleName/1.0/module

The complete library is then built using the Makefile as shown in the previous section

8.3.4.2 Building VLNV Component Library

Used with VLNVSRC and VLNVROOT to specify the root of the source directory and the root of the output binary VLNV library directory respectively.

```
$ make -f $IMPERAS_HOME/ImperasLib/buildutils/Makefile.library \
    VLNVSRC=$IMPERAS_HOME/ImperasLib/source \
    VLNVROOT=$IMPERAS_VLNV
```

It is recommended to run this from the VLNVSRC directory, this can be achieved using the following

```
$ make -C $IMPERAS_HOME/ImperasLib/source \
    -f $IMPERAS_HOME/ImperasLib/buildutils/Makefile.library \
    VLNVSRC=$IMPERAS_HOME/ImperasLib/source \
    VLNVROOT=$IMPERAS_VLNV
```

NOTE

The build system requires that the output library, VLNVROOT, directory exists. If you are building into your own library directory it may be required to create this directory before invoking the Makefile.

This is also used to build your own library components in the following way. This assumes your own VLNV library with correct structure.

```
$ mkdir -p $HOME/lib/$IMPERAS_ARCH/MyLib
$ make -C $HOME/MyLib/source \
    -f $IMPERAS_HOME/ImperasLib/buildutils/Makefile.library \
    VLNVSRC=$HOME/MyLib/source \
    VLNVROOT=$HOME/lib/$IMPERAS_ARCH/MyLib
```

8.3.4.3 Selecting a User Component Library

8.3.4.3.1 Selecting for all simulations

By default the Imperas VLNV library is selected using the IMPERAS_VLNV environment variable, such as shown below

```
$ export IMPERAS_VLNV=$IMPERAS_HOME/lib/$IMPERAS_ARCH/ImperasLib
```

This allows all components within this library to be found and used in platforms.

However, when you are creating custom components, they may be created using separate source and binary libraries which are built using the Makefile described in the previous section.

The `IMPERAS_VLNV` environment variable is a list of libraries to be searched. The list uses a colon (':') as the separator on Linux and a semi-colon(';') as the separator on Windows.

Thus if a new library, `$HOME/lib/$IMPERAS_ARCH/myLib`, is created with custom components we can add this to the `IMPERAS_VLNV` search path,

For Linux, in a shell or shell script

```
$ export IMPERAS_VLNV=$IMPERAS_VLNV:$HOME/lib/$IMPERAS_ARCH/myLib
```

For Windows, in a batch file

```
$ set IMPERAS_VLNV=%IMPERAS_VLNV%;%HOME%\lib\%IMPERAS_ARCH%\myLib
```

For Windows, in an MSYS/MINGW shell

Note that a semi-colon is used as the separator but it is also required to quote the complete string so that it is not pre-processed by Windows but passed complete into the OVPsim or Imperas product.

```
$ export IMPERAS_VLNV="$IMPERAS_VLNV;$HOME/lib/$IMPERAS_ARCH/myLib"
```

8.3.4.3.2 *Selecting for a Specific Simulation*

To use a VLNV library only with a specific simulation the command line argument `--vlnvroot` can be added. This specifies an additional VLNV library to use for this simulation.

In the following the module specified and the components it references may be in the product VLNV library or in the library *myLib*.

```
$ harness.exe --modulevendor test.ovpword.org \  
--modulelibrary module \  
--modulename test \  
--moduleversion 1.0 \  
--program application/application.OR1K.elf \  
--vlnvroot $HOME/lib/$IMPERAS_ARCH/myLib
```

NOTE

This is the recommended approach to use so that it is apparent on the command line which libraries are being used in which simulations.

8.3.4.4 Generating Deprecated ICM API TLM Interface files

The VLNV library no longer provides the deprecated ICM API TLM2.0 interface files. If these are required, for use on an old project, they must be generated.

The Peripheral model TLM2.0 interface files are generated using the target `tlmpse` defined in `Makefile.ICM_TLM.igen`.

The Processor model TLM2.0 interface files must be individually generated using `iGen` to match the specific configuration of the processor models.

The next two sections show how to accomplish this.

8.3.4.4.1 Peripheral Models

Used with `VLNVSRC` to specify the root of the source directory of the VLNV library directory the `tlmpse` target will build the `tlm2.0` (ICM) TLM interface files.

```
$ make -f $IMPERAS_HOME/ImperasLib/buildutils/Makefile.ICM_TLM.igen \
      VLNVSRC=$IMPERAS_HOME/ImperasLib/source tlmpse
```

It is recommended to run this from the `VLNVSRC` directory, this can be achieved using the following

```
$ make -C $IMPERAS_HOME/ImperasLib/source \
      -f $IMPERAS_HOME/ImperasLib/buildutils/Makefile.ICM_TLM.igen \
      VLNVSRC=$IMPERAS_HOME/ImperasLib/source tlmpse
```

8.3.4.4.2 Processor Models

Each processor configuration requires a separate TLM interface file to match the specific configuration of, for example, bus ports and interrupt connections.

The following shows the generation of the ICM TLM2.0 interface file for the ARM processor configured for variant `ARMv5TEJ`.

```
$ mkdir -p ImperasLib/source/arm.ovpworld.org/processor/arm/1.0/tlm2.0
$ igen.exe -modelvendor arm.ovpworld.org --modellibrary processor --modelname
arm -setparameter variant=ARMv5TEJ \
    -icm \
    -writetlm
ImperasLib/source/arm.ovpworld.org/processor/arm/1.0/tlm2.0/arm_ARMv5TEJ.igen.h
pp \
    -userheader ImperasLib/fileheaders/refArmApache.txt
```

And the generation of the OP TLM2.0 interface file also for variant M5150.

```
$ igen.exe \  
-modelvendor mips.ovpworld.org --modellibrary processor --modelname mips32 \  
-setparameter variant=M5150 \  
-op \  
-writetlm mips32_M5150.igen.hpp \  
-userheader ImperasLib/fileheaders/refApache.txt
```

Additional configuration parameters can be assigned by adding further *-setparameter* arguments.

For example, generate the TLM2 interface for the ARM CortexA(MPx2 processor with additional interrupt lines.

```
$ igen.exe \  
-modelvendor arm.ovpworld.org --modellibrary processor --modelname arm \  
-setparameter variant=CortexA9MPx2 \  
-setparameter override_GICD_TYPER_ITLines=6\  
-op \  
-writetlm arm_Cortex-A9MPx2-GICD_TYPER_ITLine-6.igen.hpp \  
-userheader ImperasLib/fileheaders/refArmApache.txt
```

8.4 Hello World Example

There are many examples and demonstrations that are provided in an installation. They are in the directories `$IMPERAS_HOME/Examples` and `$IMPERAS_HOME/Demo`.

There are a number of different ways of building a hardware definition of a platform and controlling a simulation. In this section we will look at three different aspects using the provided 'hello world' examples.

1. Executing an application on a processor using the ISS, see 8.4.2
2. Creating our own hardware definition and executing with fixed harness, see 8.4.5
3. Creating our own harness, see 8.4.6

In previous examples we used the ISS to run the OR1K Fibonacci example. The ISS uses the model library and allows a binary cross-compiled ELF file to be run on a processor model variant without a specific hardware definition. It is the simplest way to run and debug software.

If you need to model hardware then there are several ways to create it.

Look in the `Examples/HelloWorld` directory:

```
$ ls $IMPERAS_HOME/Examples/HelloWorld
usingCPP  usingHarnessExe  usingHarnessInC  usingHarnessInCPP  usingOP
usingISS  usingSystemC
```

For the simplest, look at `usingISS`:

```
$ cp -r $IMPERAS_HOME/Examples/HelloWorld/usingISS .
$ cd usingISS
```

In this directory you should see an application directory with a file `application.c`, which can run on this platform, and `Makefile` for building the application.

8.4.1 Compiling an application

Compile the Application using the OR1K Cross Compiler toolchain.

```
$ cd application
$ make
```

Or

```
$ make -C application
```

If you see the following, the toolchain has not been installed or it has not been installed into the current release that is being executed. (Install the toolchain now.)

```
Makefile:12: *** "Please install the toolchain to support OR1K ". Stop.
```

A correct compilation will give you output something like

```
$ make
Compiling application.c
Linking application.OR1K.elf
rm application.o
```

To see exactly what is going on add *VERBOSE=1* to the command line

```
$ make CROSS=OR1K VERBOSE=1
echo "Compiling application.c"
Compiling application.c
C:/Imperas/lib/WINDOW~1/CROSSC~1/or32-elf/bin/or32-elf-gcc
-IC:/Imperas/ImpPublic/include/common
-IC:/Imperas/ImpPublic/include/target/common
-IC:/Imperas/ImpPublic/include/target/application
-IC:/Imperas/lib/Windows64/TargetLibraries/include/or32 -c -o application.o
application.c
echo "Linking application.OR1K.elf"
Linking application.OR1K.elf
C:/Imperas/lib/WINDOW~1/CROSSC~1/or32-elf/bin/or32-elf-gcc
-IC:/Imperas/ImpPublic/include/common
-IC:/Imperas/ImpPublic/include/target/common
-IC:/Imperas/ImpPublic/include/target/application
-IC:/Imperas/lib/Windows64/TargetLibraries/include/or32 -nostartfiles
-TC:/Imperas/lib/Windows64/TargetLibraries/or32-elf/or32-elf-lib.ld -LC:/Imperas/lib/Windows64/TargetLibraries/or32-elf
C:/Imperas/lib/Windows64/TargetLibraries/or32-elf/crt0.o -o
application.OR1K.elf application.o -limperas -lm
rm application.o
```

This cross compilation of the application for the OR1K processor, will produce a file called `application.OR1k.elf`.

Move back up a directory

```
$ cd ..
```

8.4.2 Running the simulation using the ISS

Run the compiled application using the ISS:

```
$ iss.exe --program application/application.OR1K.elf \
--processorvendor ovpworld.org \
--processorname orlk \
--variant generic \
--verbose
```

This should provide an output similar to that shown below:

```
OVPsim (32-Bit) v20150901.0 Open Virtual Platform simulator from
www.OVPworld.org.
Copyright (c) 2005-2015 Imperas Software Ltd.  Contains Imperas Proprietary
Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.
```

```

OVPSim started: Fri Sep 18 11:45:52 2015

Info (OP_AL) Found attribute symbol 'modelAttrs' in file
'C:/Imperas/lib/Windows32/ImperasLib/ovpworld.org/processor/orlk/1.0/model.dll'
Info (OP_AL) Found attribute symbol 'modelAttrs' in file
'C:/Imperas/lib/Windows32/ImperasLib/ovpworld.org/semihosting/orlkNewlib/1.0/model.dll'
Info (OR_OF) Target 'platform/cpul' has object file read from 'application.OR1K.elf'
Info (OR_PH) Program Headers:
Info (OR_PH) Type  Offset      VirtAddr  PhysAddr  FileSiz  MemSiz    Flags Align
Info (OR_PD) LOAD 0x00002000 0x00000000 0x00000000 0x0000dc48 0x0000dd64 RWE   2000
Hello World
Info
Info -----
Info CPU 'platform/cpul' STATISTICS
Info   Type                : orlk
Info   Nominal MIPS        : 100
Info   Final program counter : 0x1720
Info   Simulated instructions: 2,226
Info   Simulated MIPS      : run too short for meaningful result
Info -----
Info
Info -----
Info SIMULATION TIME STATISTICS
Info   Simulated time       : 0.00 seconds
Info   User time           : 0.02 seconds
Info   System time         : 0.00 seconds
Info   Elapsed time        : 0.02 seconds
Info -----

OVPSim finished: Fri Sep 18 11:45:52 2015

OVPSim (32-Bit) v20150901.0 Open Virtual Platform simulator from www.OVPworld.org.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

```

NOTE

The 'Simulated MIPS' figure provides an indication of how many millions of instructions the simulator was able to run on your host machine per second. You will get the message 'run too short for meaningful result' if the simulated time is less than a second.

8.4.3 Execute example using provided script

For convenience we provide `example.bat` or `example.sh` that performs all the required actions, for example compile the application and runs the simulation:

```

$ ./example.sh
...

```

8.4.4 Creating a Virtual Platform Tutorial Video

Detailed information on how to create a virtual platform description of a hardware platform may be found by watching the tutorial video available when logged into the OVPWorld website at

<http://www.ovpworld.org/creating-virtual-platforms>

8.4.5 Creating and Simulating with a Platform/Module

The previous example showed compiling an application and then using the ISS to run it on a processor. In the next example we will create a simple hardware definition, a module, and use the provided harness program to load the hardware definition, load the application elf file and control the simulation.

```
$ cp -r $IMPERAS_HOME/Examples/HelloWorld/usingHarnessExe .
$ cd usingHarnessExe
```

Compile the application in the `application` directory:

```
$ make -C application
```

then look in the `module` directory

```
$ cd module
$ ls
```

We will see there is `module.op.tcl` and `Makefile`. The `module.op.tcl` is the definition of the hardware using iGen input script commands.

The provided Makefile, includes the standard Makefile *Makefile.module*, will run iGen on the input file:

```
$ make NOVLNV=1
# iGen Create OP MODULE module
# Host Depending obj/Windows64/module.d
# Host Compiling Module obj/Windows64/module.o
# Host Linking Module object model.dll
```

and if you now list the directory, you will see several `.c` and `.h` files created by iGen. These are normal C files that make use of the standard OVP OP API that create platforms. For examples of platforms/modules written directly in OP, please refer to:

Writing_Platforms_and_Modules_in_C_User_Guide.pdf

There are many examples of using this approach in the `Examples/PlatformConstruction` directory and there are 2 comprehensive documents in `doc/ovp` that describe the use of iGen for platforms/modules:

iGen_Model_Generator_Introduction.pdf

iGen_Platform_and_Module_Creation_User_Guide.pdf

Also, please look at the iGen section in `doc/ovp/index.html`. iGen is a platform and model building wizard/productivity tool to make it easy to create platforms/models.

The running of the Makefile has created the `module/model.dll` that is a binary representation of the input `.c/.h` that was created by iGen from the input `.tcl` file. This `module/model.dll` can be used in other platforms or modules (as part of a system/subsystem hierarchy), or it can be simulated directly using the provided `harness.exe` program:


```
$ cd ..
$ harness.exe --modulefile module/model.dll
               --program application/application.OR1K.elf
...
OVPsim started: Wed Mar 23 12:00:23 2016

Hello World using harness.exe

OVPsim finished: Wed Mar 23 12:00:23 2016
...
```

You can see what control options are available in the harness.exe by using the help option:

```
$ harness.exe --help
```

Try out some such as

```
--trace
--showbuses
--showlibrarymodules
--showoverrides
--gdbconsole
```

Again there is a script to compile the application, module and run the harness.exe:

```
$ ./example.sh
```

In this example the hardware module definition is in a local directory. The harness can also load modules from the VLNV library by specifying the arguments *--modulename*, *--modulelibrary*, *--modulename* and *--moduleversion*.

For example

```
$ harness.exe \
  --modulevendor    test.ovpworld.org \
  --modulelibrary   module \
  --modulename      test \
  --moduleversion   1.0
```

The harness will load the module found in the VLNV library that satisfies the vendor and name. This command line will essentially load a shared object found at

```
lib/$IMPERAS_ARCH/ImperasLib/test.ovpworld.org/module/test/1.0
```

It is possible to omit any of *--modulevendor*, *--modulelibrary* or *--moduleversion* if the defined arguments still generate a unique item in the VLNV library.

8.4.6 Writing your own test harness

The previous example showed creating a platform/module and using the provided simulation test harness to run and control the simulation. You can write your own test harness directly using three OP API to load and simulate.

The OVP OP API provides functions to control a simulation. There are examples of this in the `Examples/SimulationControl` directory and there are two documents which provide common use cases

[Simulation_Control_of_Platforms_and_Modules_User_Guide.pdf](#)

[Advanced_Simulation_Control_of_Platforms_and_Modules_User_Guide.pdf](#)

```
$ cp -r $IMPERAS_HOME/Examples/HelloWorld/usingHarnessInC .
$ cd usingHarnessInC
```

This has an `application` directory and `module` directory as in the previous examples. It also has a `harness` directory:

```
$ cd harness
$ ls
harness.c    Makefile
```

The `harness.c` is a regular C program that uses calls to the OVP OP API to control the loading and simulation of a module. Please refer to the documents mentioned above for information on the OP API calls etc.

As in previous examples, for convenience a script it provided to compile and run the complete example:

```
$ ./example.sh
...
# Compiling application.c
...
# iGen Create OP MODULE module
...
# Host Linking Harness harness.Windows64.exe
...
OVPSim started: Wed Mar 23 12:21:06 2016

Hello World using C Harness

OVPSim finished: Wed Mar 23 12:21:06 2016
```

9 Understanding Semihosting Support

9.1 In Imperas and OVP simulations

The term "semihosting" can mean several things. In the case of OVP simulations it refers to specific behavior that is provided by the host system rather than the simulation platform or an operating system running on the simulation platform.

For example, in an embedded system, to get output on a terminal the platform would typically include a hardware device, such as a UART, and the application software would provide a driver to initialize the device and provide low level functions that can be used by library functions, for example `printf`, to output characters to the terminal. This adds complexity to the software and the platform that is not required if we simply wanted to cross compile and run a bare metal C application on a specific processor variant.

In this example semihosting would be used to provide behavior for the low level functions without having to add anything to the platform or application code. A semihost library replaces behavior of the low level function in the cross compiled application with a native host function provided within a semihosting intercept library.

The semihost library is native code that is loaded as a shared object by the simulator. It is defined in the instantiation of a processor instance in a platform.

9.2 Replacing function and/or instruction behavior

Different mechanisms may be used to support semihosting. This may be replacing a function's behavior or using one or more¹² specific instructions to control semihost operations.

A semihost library may replace a function in its entirety to provide that function itself, for example the function `_write`, which sends a character from a buffer to a hardware device output register, may be intercepted and replaced by a semihost function that takes the character from the buffer and displays it on the host stdout.

A semihost library may replace a specific instruction, typically a software interrupt instruction, which is a way of the application program requesting an action from a host operating system, but that in this case is intercepted and the function performed

9.3 Specific to a Cross Compiler and C Library

The way a semihost library is intended to work is specific to a processor type and also to the cross compiler toolchain and C library used during the application compilation and linkage. It is, therefore, important that the correct semihost library is included with the processor instantiation in a platform for the cross compiler and C library used to build the application that will execute on the processor.

¹² The actual number of instruction types that can be used is limited in a semihost library

Example mips32 cross compiler toolchain uses mips32Newlib semihost library. The C library used in the application compilation in this case is specified by a linker script, `mipssim-hosted.ld`. This controls the C library included. The semihosting is performed at the low level function level, as defined in the `.intercept` table of the semihost file `mips32Newlib.c` in the VLNV library.

Example ARM Linaro cross compiler toolchain used armAngel semihost library. The C library used in the application compilation in this case is specified by a specs file, `aprofile-ve.specs`. This controls the C library included. The semihosting is performed at the instruction level, as defined in the `armOSOperation` function of the semihost file `armAngel.c` in the VLNV library.

9.4 Used to terminate the simulation

The semihost library may also detect other events in the execution of application software, for example a call to `exit` or a jump-to-self.

An embedded processor would never stop executing instructions, unless put into a specific mode. In order to stop an application 'running off the end' designers typically put spin loops in areas of code that are used to capture unwanted events from which the program cannot recover including the `exit` function code.

The execution of a spin loop could cause the application to run forever and make the simulation appear to lock-up so the semihost library is used to detect spin loops or calls to `exit` and terminate the simulation.

9.5 Caution using with EPKs and non-baremetal platforms

The semihost library replaces functions and/or instructions behavior so it is not the case that a semihost library should always be used. In an EPK or other non-baremetal platform care should be taken to avoid the situation that code is not executed when expected because a symbol has happened to match one defined in the semihost library and has caused the functions normal behavior to be replaced by that of the semihost library.

10 Understanding Simulation Time Statistics

At the end of simulation, if the verbose flag has been used, some statistics will be reported by the simulator. This provides information to indicate how many instructions were executed on the OVP Processor model and the equivalent simulated Millions of Instructions Per Second (MIPS) that were executed. This can be used as an indication of how well the simulator performed on the host machine running the particular application.

The typical simulation statistics is shown below. Most entries are self explanatory but are described in the following sections

```
Info -----
Info CPU 'platform/cpu0' STATISTICS
Info   Type                : arm (Cortex-A9UP)
Info   Nominal MIPS        : 100
Info   Final program counter : 0x4b8
Info   Simulated instructions: 22,400,008,761
Info   Simulated MIPS       : 5720.7
Info -----
Info
Info -----
Info SIMULATION TIME STATISTICS
Info   Simulated time       : 224.00 seconds
Info   User time            : 3.92 seconds
Info   System time         : 0.00 seconds
Info   Elapsed time        : 3.93 seconds
Info   Real time ratio      : 56.98x faster
Info -----
```

OVP Fast processor model configuration information

```
Info   Type                : arm (Cortex-A9UP)
Info   Nominal MIPS        : 100
```

The model type and, if selected, the configuration variant are displayed. The nominal MIPS for the processor model, by default, is 100 MIPS. This has an effect when running with 'wallclock' enabled or when in a multi-processor system. With `-wallclock` enabled it is used to control the maximum execution performance i.e. only 100 million instructions will be executed in a real time 1 second. In a multi-processor system (without wallclock enabled) it provides an execution ratio between processors e.g. a 200 MIPS processor will execute twice the instructions as a 100 MIPS processor in a simulation unit of time.

Program execution information

```
Info   Final program counter : 0x4b8
Info   Simulated instructions: 22,400,008,761
```

'Simulated instructions' will vary depending upon the application being executed, this count indicates the number of simulated processor instructions for the processors in the

platform. The final program counter can be useful as an easy way of showing the application program executes in a deterministic way over a number of simulation runs.

Simulated MIPS

```
Info    Simulated MIPS          : 5720.7
```

The 'Simulated MIPS' will be a measure of the number of 'Simulated instructions' over the host elapsed time and is an indication of the performance of the simulator.

There is an overhead at the start of simulation with Just-In-Time code morphing simulators and so the Simulated MIPS are not calculated if the simulated run time is below a threshold. If you see the following output you are typically running too few instructions i.e. your application is too short.

```
Info    Simulated MIPS          : run too short for meaningful result
```

Simulation time statistics

The simulated time is the time it would have taken to run the application based upon the MIPS configuration of the processor and the number of instructions the application took to complete.

```
Info    Simulated time          : 224.00 seconds
```

The time fields are real time information from the native host machine relating to the simulator execution.

```
Info    User time                : 3.92 seconds
Info    System time              : 0.00 seconds
Info    Elapsed time            : 3.93 seconds
```

The "real time ratio" is an indication of how much quicker the simulator was able to execute the processor instructions than the real silicon based upon the OVP Fast processor model MIPS rate configuration and the time the simulator took to execute the instructions on the native host machine.

```
Info    Real time ratio          : 56.98x faster
```

11 Understanding the operation of a code morphing simulator

11.1 Instruction Fetch

A code morphing simulator operates in two phases to provide the behavior of a target processor executing a sequence of instructions.

1. Morphing cross compiled target instructions (for example ARM Cortex-A9, MIPS 1074Kc, V850) to a sequence of native host instructions
2. Executing the sequence of native host instructions to provide the behavior expected of the target

In 1 we are executing in 'morph time' using the translations specified by the OVP processor model. During the translation process, each instruction is read from the virtual platform memory using a debug (artifact) transaction. This access must have no effect on the system and must be implemented as a 'back door' access that bypasses any behavior of the system; including timing or delays that may be in a SystemC TLM2 virtual platform implementation.

The code morpher will continue reading instructions and morphing code until it has constructed an entire 'code block'. This may be a few or many instructions.

Once a code block is created, the block is executed. We are now in 2, in 'run time', for each target instruction behavior, a) the instruction is reread from memory - but this time as a 'real' transaction that will have an effect on the system that the fetch would normally have and b) the native code is executed to create the behavior.

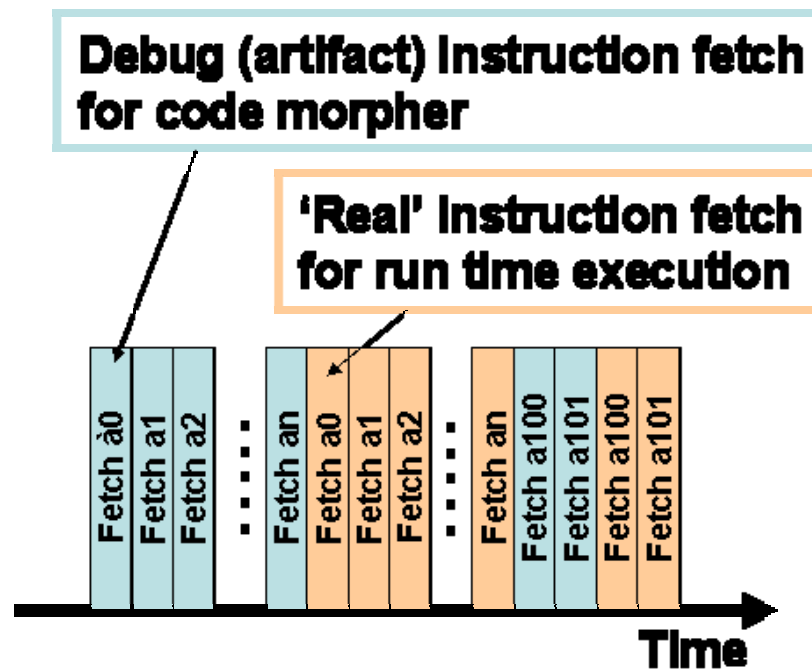


Figure 1: Real and Artifact accesses over time

11.2 SystemC Interface Transaction Types

In SystemC TLM2 interface code is generated for each OVP fast processor model.

There is also generic OVP processor interface code that is provided in the installed component source library.

The component VLVN source library is found at `IMPERAS_HOME/ImperasLib/source`, with the V/L/N/V of the TLM2 support models

`ovpworld.org/modelSupport/tlmProcessor/1.0` containing the interface file

11.2.1 OP API (Current)

`tlm/tlmProcessor.hpp`
`tlm/tlmBusPort.hpp`

In this file you will find the function `read` that will perform a TLM2 read (fetch) transaction by calling either `socket->b_transport()` for a 'real' access or `socket->transport_dbg()` for a debug or simulation artifact access.

11.2.2 ICM API (Deprecated)

`tlm2.0/tlmProcessor.cpp`

In this file you will find the function `icmCpuMasterPort::readUpCall` that will perform a TLM2 read (fetch) transaction by calling either `socket->b_transport()` for a 'real' access or `socket->transport_dbg()` for a debug or simulation artifact access.

IMPORTANT NOTE

The SystemC virtual platform must provide **both** *debug* and *real* interface connections.

APPENDIX A

Installation Packages, Products and Licensing Features

A.1 Installation Packages

There are three main product installation packages available to download via the Imperas or OVP World websites.

1. Imperas_SDK
2. Imperas_DEV
3. OVPSim

In addition to the product installation packages above are a number of installers providing specific demonstrations and example toolchains to be used in compilation of peripheral models and cross compilation of processor application code.

A.1.1 Imperas_SDK package

The Imperas_SDK package is available from the Imperas website and provides the M*SDK and M*TST product features.

These include

1. Verification, Analysis and Profiling (VAP) tools
2. Multicore debugger for debugging of applications running on processor models and peripheral models in a unified environment
3. M*SIM and Imperas ISS professional simulators
4. Imperas Generator Wizard (iGen) for generating virtual platforms and model (processor and peripheral) templates
5. Model Library
6. Examples and Demonstrations

The setting of the IMPERAS_PERSONALITY environment variable to CPUMAN_MULTI is required to select the product features provided in the SDK product.

A.1.2 Imperas_DEV package

The Imperas_DEV package is available from the Imperas website and provides the Developer installations; C*DEV, S*DEV¹³ and M*DEV which provide different simulator features. The package provides the DEV product features.

These include

1. M*SIM and Imperas ISS professional simulators

¹³ The 'Standard' Developer simulator configuration provides similar capabilities to the OVPSim simulator.

2. Imperas Generator Wizard (iGen) for generating virtual platforms and model templates
3. Model Library
4. Examples and Demonstrations

The setting of the `IMPERAS_PERSONALITY` environment variable is used to select the simulator features, between those provided by Controller (C*DEV), Standard (S*DEV) Multi (M*DEV) or ISS products.

A.1.3 OVPsim package

The OVPsim package is available from the OVP World website. The package provides the product features.

These include

1. OVPsim reference simulator
2. Imperas Generator Wizard (iGen) for generating virtual platforms and model templates
3. Model Library
4. Examples and Demonstrations

A.2 License Features

A.2.1 Executable Programs

The following table shows the executable programs that are provided in the product packages.

Any specific license feature required to execute in conjunction with a simulator license is as detailed in the following section.

Product Name	Package	License Feature	Simulator / Personality
mpd.exe	Imperas_SDK	(simulator)	Not required
igen.exe	All	(simulator)	Any / Any
ipost.exe	Imperas_SDK	IMP_IPOST	not applicable
cpuGen	Imperas_SDK	(simulator) + IMP_CPUGEN	Any / Any

A.2.2 CpuManager Simulator and its Personalities

The CpuManager simulator runtime can be configured to provide different levels of simulation and feature support. The configuration of the simulator is controlled with the `IMPERAS_PERSONALITY` environment variable. The simulator has NO default personality and the environment variable must be set before running the product. The license features that the simulator will checkout are controlled by the personality.

Simulator Personality	Product	License Feature
CPUMAN_CONTROLLER	C*DEV	IMP_CPUMAN_CONTROLLER
CPUMAN_STANDARD	S*DEV	IMP_CPUMAN_STANDARD
CPUMAN_MULTI	M*DEV	IMP_CPUMAN_MULTI
CPUMAN_MULTI	M*SDK	IMP_CPUMAN_MULTI and IMP_VAP (if required)
CPUMAN_MULTI	M*TST	Interactive Usage IMP_CPUMAN_MULTI and IMP_MPD, IMP_VAP (as required) Batch Usage IMP_CPUMAN_MULTI_BATCH and IMP_VAP_BATCH (as required)
CPUMAN_ISS	ISS	IMP_CPUMAN_ISS
OVPSIM	OVPsim	IMP_OVPSIM

A.2.3 CpuManager Simulator and Non-Interactive (batch) Usage

The CpuManager simulator supports interactive and non-interactive modes of operation; each are separately licensed.

The non-interactive usage is controlled using the `IMPERAS_BATCH_LICENSE` environment variable.

If the environment variable is set it causes the simulator to operate in different modes defined by the value:

- 0 use interactive license (default normal operation)
- 1 use non-interactive (batch) license
- 2 use non-interactive (batch) license or use interactive license if the batch license is not available

If the environment variable is not set the interactive mode is used.

If batch (non-interactive) mode is being used, then a `BATCH` version of the license feature is required. For example, using personality `CPUMAN_MULTI` in batch mode would require the license feature `IMP_CPUMAN_MULTI_BATCH` instead of the normal license feature `IMP_CPUMAN_MULTI` used in interactive (standard) operation. If the Imperas VAP tools are being used in batch mode, the license key `IMP_VAP_BATCH` will be used in preference to `IMP_VAP`.

A.2.4 *OVPsim Simulator*

The OVPsim simulator does not use the `IMPERAS_PERSONALITY` environment variable and provides fixed simulation support, equivalent of the Developer S*DEV product.

APPENDIX B

Obtaining an Imperas License

To obtain an

Imperas license send an email request to **license_support@imperas.com**,
 OVPsim license visit the OVPworld.org license page, or send an email to
license@ovpworld.org

The email sent must contain the following information:

Name	of the person responsible for the Imperas/OVP Tools
e-mail address	
Phone number	
Company Address	
Host id ¹⁴	of the machine running the license server
Host Name	
Type (Linux, Windows)	

1.1.1 ¹⁴ This is the FlexLM information obtained using the Imutil(.exe) that is installed with all packages. See section 4.6.7 Internet Access Via a Proxy Server

If you are using a Demo license downloaded from the OVP website, this will require web access in order to run the simulator.

If this is the case and you use an Internet Proxy Server to access the web, you must set the environment variable, IMPERAS_PROXY_SERVER, in order to enable access to the internet.

For example (shown for a Linux host) if the Proxy Server is running on myproxyserver.com at port 3128

```
$ export IMPERAS_PROXY_SERVER="http://myproxyserver.com:3128"
```

A simple check to ensure this has worked correctly is as follows

```
$ export IMPERAS_PROXY_SERVER="http://myproxyserver.com:3128"
$ http_proxy=${IMPERAS_PROXY_SERVER} wget http://www.ovpworld.org
```

This should cause the index.html to download if all is successful
 Setting up Licensing

APPENDIX C

Accessing Imperas User Area

C.1 Initial Login at Imperas User

Login at www.imperas.com and select the "User Login" link. At this point you will use the same username and password that is used on the OVP World website.



Imperas User Login

Please login with your www.imperas.com or www.OVPworld.org user data:

Username

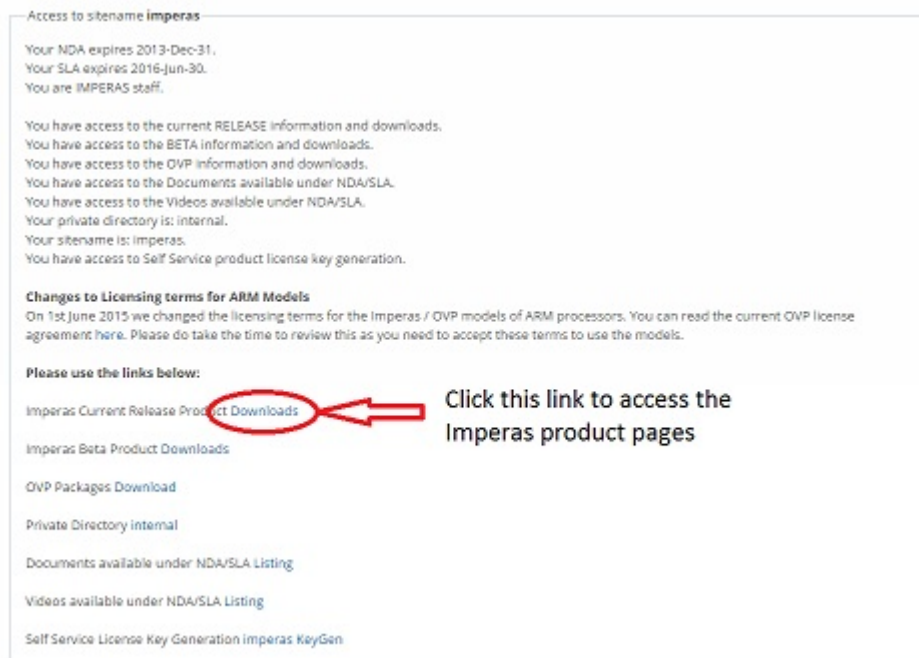
Password

Add OVP World Username Here

Login using www.imperas.com or www.OVPworld.org Username/Password

C.2 Selecting Imperas Product download

The Imperas products are found at the “Downloads” link for the Imperas Current Release Product.



Access to sitename **imperas**

Your NDA expires 2013-Dec-31.
Your SLA expires 2016-Jun-30.
You are IMPERAS staff.

You have access to the current RELEASE information and downloads.
You have access to the BETA information and downloads.
You have access to the OVP information and downloads.
You have access to the Documents available under NDA/SLA.
You have access to the Videos available under NDA/SLA.
Your private directory is: internal.
Your sitename is: imperas.
You have access to Self Service product license key generation.

Changes to Licensing terms for ARM Models
On 1st June 2015 we changed the licensing terms for the Imperas / OVP models of ARM processors. You can read the current OVP license agreement [here](#). Please do take the time to review this as you need to accept these terms to use the models.

Please use the links below:

[Imperas Current Release Product Downloads](#)

[Imperas Beta Product Downloads](#)

[OVP Packages Download](#)

[Private Directory internal](#)

[Documents available under NDA/SLA Listing](#)

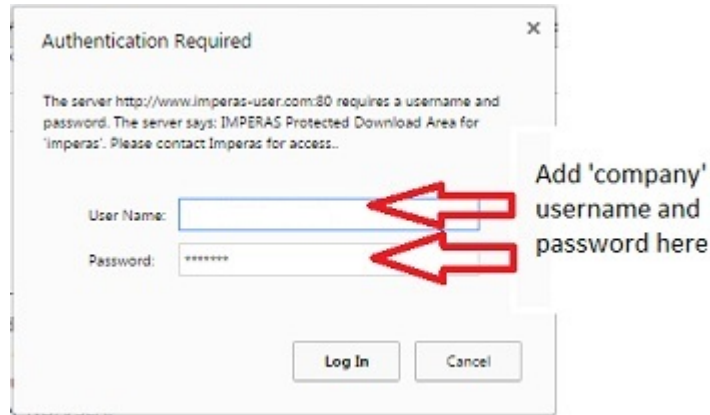
[Videos available under NDA/SLA Listing](#)

[Self Service License Key Generation imperas KeyGen](#)

Click this link to access the Imperas product pages

C.3 Logging into Download area

When any of the links are selected you will get a popup in which the company username and password should be entered. This is common to all users of a company, university etc. and is provided by Imperas.



C.4 Selecting Files

You are now in a page listing all the available Imperas product packages. Select the link for the package that you wish to download.

Windows (Windows64) (17 available)	
Downloads	Size
Imperas_FlexLM.20150901.0.Windows64.exe	4.71 MB
Imperas_DEV.20150901.0.Windows64.exe	99.8 MB
Imperas_RenesasUPD70F3441.20150901.0.Windows64.exe	5.42 MB
Imperas_SDK.20150901.0.Windows64.exe	116.55 MB
Imperas_hetero_Nucleus_arm_Linux_mips_jcm.20150901.0.Windows64.exe	21.1 MB
Imperas_hetero_armxt_mips32x3_jcm.20150901.0.Windows64.exe	7.41 MB

APPENDIX D

Setting Environment Variables

D.1 Opening System properties on Windows XP

In order to set an environment variable using Windows XP follow these steps

Select ‘Control Panel’

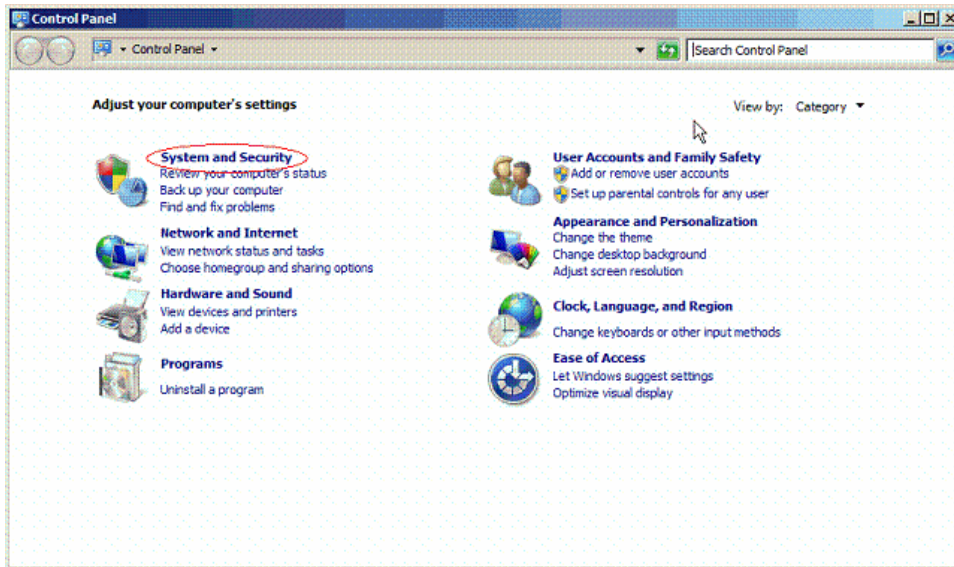


Select ‘System’ to bring up the ‘System properties’ dialogue box.

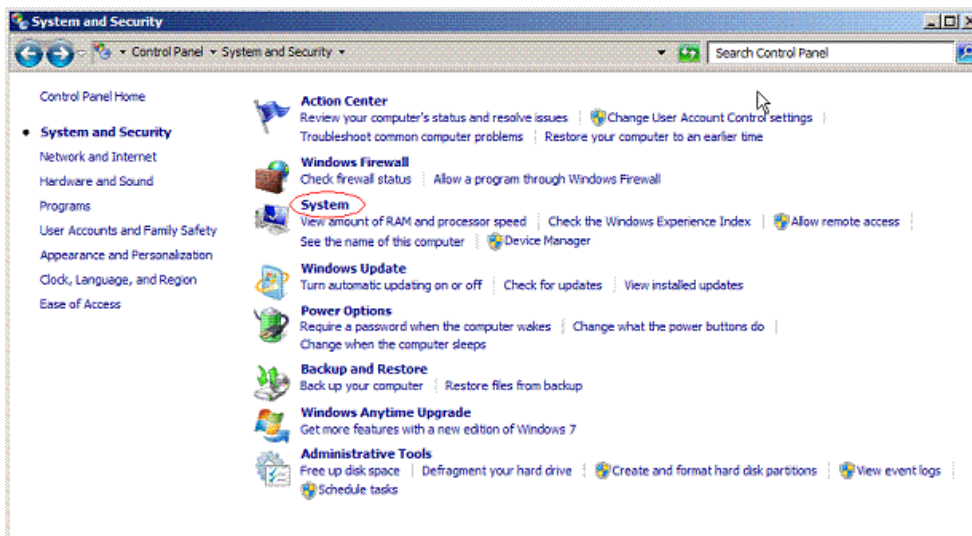


D.2 Opening System properties on Windows 7 and 10

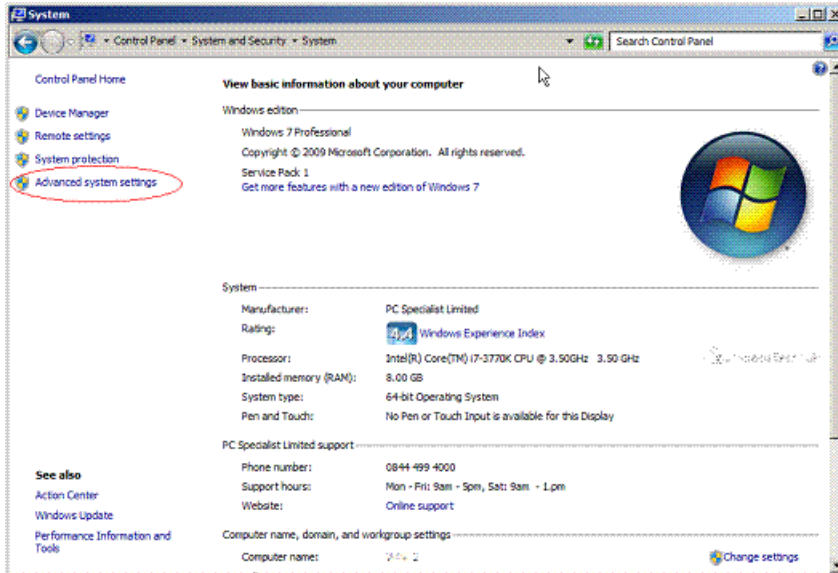
Select ‘Control Panel’ and then select ‘System and Security’



Now select 'System'



Finally select ‘Advanced system settings’



D.3 Modifying Environment Variables in Windows

Select ‘Advanced’ tab and then ‘Environment Variable’



APPENDIX E

Additional FlexNet Licensing Information

E.1 Locating the license server software

The license server and daemon are provided as part of the OVPsim or Imperas releases or may be downloaded from the Imperas and OVPWorld websites.

After installing the OVPsim or Imperas packages the license server, `lmgrd`, and the Imperas license daemon, `imperasd`, can be found in the binary installation directory.

On a Windows machine the server will be found

```
$IMPERAS_HOME/bin/$IMPERAS_ARCH/lmgrd.exe
```

On a Linux machine the server will be found

```
$IMPERAS_HOME/bin/$IMPERAS_ARCH/lmgrd
```

The daemon, `imperasd`, is found in the same directory.

E.2 Types of Licenses

There are several different types of license files. A license can be node-locked, which means the tools can only run on a single computer, or it can be *floating* in which case one computer acts as a license server, which runs a license daemon, and the tools may run on any computer that can contact the server over the local area network to check out a license.

In the case of a node-locked license, you will send Imperas/OVPworld the host name and host ID of the computer that you will run the tools on. In the case of a floating license you will need to choose one computer to be the license server and send the host name and host ID of that computer to Imperas/OVPworld to obtain your license. Note the license server may be the same computer you run the tools on.

In addition a license may be *uncounted*, which means that any number of copies of the program may run simultaneously, or it may be *counted* which means that only a specified number of instances of the program may be active at one time.

11.2.3 E.2.1 Uncounted Node-locked Licenses

An uncounted node-locked license is the simplest sort of license. It does not require a license server or a license daemon to be running. You can tell your license is node-locked

if it has only uncounted FEATURE lines and comment lines (which start with a #) in it. For example:

```
#### OVPSim non-commercial license keys
#### Generated on 2011-04-30
#### Hostname1 PC1
FEATURE IMP_OVPSIM_20130630 imperasd 1.0 25-aug-2013 uncounted \
    HOSTID=002119426114 SIGN="0874 1401 CA64 337B FC3B 9CE6 9D29 \
    CFD3 F89E F4C1 D090 9084 CE9A E2A8 D8BA 0F6F BEF9 CC19 ACF4 \
    A7F9 6834 145A 0B11 0BED AD50 6672 47B2 1758 7B24 65D2"
```

You can tell if a license feature is uncounted by looking for the word `uncounted` following the expiration date of the feature.

If a license contains only node-locked, uncounted features then you only need to set the environment variable `IMPERASD_LICENSE_FILE` to point at it, e.g.:

```
IMPERASD_LICENSE_FILE=~/.Imperas/license.lic
```

See the respective installation instructions for Windows and Linux for examples of how to set an environment variable.

11.2.4 E.2.2 Floating Licenses

If your license is a floating license then you will need a license daemon program running on a license server. With a floating license any computer that can find the license server on the local area network can run the tools.

An example of a floating license looks like:

```
SERVER server1 00F346829930
VENDOR imperasd /path-to-daemon/imperasd
USE_SERVER
##### Feature List #####
FEATURE IMP_SIMULATOR imperasd 1.0 25-aug-2013 5 TS_OK SIGN="5763\
    07C3 196A 858A 5455 A9EF 541E D2D4 6A75 1ED5 B3BF AE94 F141 \
    1A7A BE50 CD76 3466 A503 3035 4464 9788 4911 AEC5 F73B 18E7 \
    92AC 27CF B8A4 9FF3 0DE4 "
FEATURE IMP_CPUMANAGER imperasd 1.0 25-aug-2013 5 TS_OK SIGN="10D9 \
    9297 4903 02EE 90C9 95BC 3371 9B72 11D4 F9E2 056F BBF2 AA5F \
    4DBE 681B 0193 A355 B1B2 1638 6BBF 01CD A006 5E73 56C1 FD5D \
    EAB6 B2E0 70C3 893F 9755"
```

Here we see `SERVER`, `VENDOR` and `USE_SERVER` lines in addition to the `FEATURE` lines we saw in an uncounted node-locked license. Also, after the expiration date on the `FEATURE` lines we see 5 instead of `uncounted` indicating this is a counted license. When either of these is true a license server will be required.

The format of the `SERVER` line is as follows:

```
SERVER <host name> <host ID> {<port>}
```

`<host name>` should be the host name of the license server. This can be edited if it is not

correct.

<host ID> is the host id of the license server. This cannot be changed without invalidating the license file. The license server must be the computer with this host ID. {<port>} is an optional integer value which specifies the port the server communicates through. It should usually be left blank, in which case a port in the default range of 27000 to 27009 will be used. If it is specified in the license file then the same value must be specified on the `IMPERASD_LICENSE_FILE` environment variable on the computer running the target program.

The format of the `VENDOR` line is as follows:

```
VENDOR <vendor_daemon_name> <path to vendor daemon file>
```

<vendor_daemon_name> should be `imperasd`. <path to vendor daemon file> should be edited to be the path of the `imperasd` (`imperasd.exe` on Windows) file located in the directory `$IMPERAS_HOME/bin/$IMPERAS_ARCH` for example:

```
VENDOR imperasd /usr/Imperas/bin/Linux32/imperasd
or
VENDOR imperasd C:\Imperas\bin\Windows32\imperasd.exe
```

E.3 Starting the License Server

The Imperas license daemon (`imperasd`) must be run on the license server using the Flexera license tools. The FlexNet `lmutil` tool and the Imperas daemon are provided in the `$IMPERAS_HOME/bin/$IMPERAS_ARCH` directory. The daemon can be started using the following command on the license server:

On a Windows machine in an MSYS shell

```
$ $IMPERAS_HOME/bin/$IMPERAS_ARCH/lmgrd.exe -c license.lic -l license.log
```

On a Linux machine in a shell

```
$ $IMPERAS_HOME/bin/$IMPERAS_ARCH/lmgrd -c license.lic -l license.log
```

Where `license.lic` is the name of the license file which must be able to be read from the license server and `license.log` is the name of a file where the license manager will write informational and debug messages.

The license server may also be started in a foreground mode by adding the `-z` switch

```
$ $IMPERAS_HOME/bin/$IMPERAS_ARCH/lmgrd -z -c license.lic
```

If the license server is a different computer than you installed the tools on, you may want to copy the license manager tools and the daemon file from

`$IMPERAS_HOME/bin/$IMPERAS_ARCH` to the license server. The license tools all start with “lm” and the daemon file is named either `imperasd` or `imperasd.exe`.

The `lmgrd` command will need to be re-run every time the license server is rebooted.

On Linux it can be placed in a boot script file (e.g. in `/etc/rc.local`).

On Windows it can be started as a service each time the computer is rebooted. The command `lmtools` will start a GUI program where you can configure the windows `lmgrd` service from the `Config Services` tab.

E.4 Configuring the Host Computer

On the computer that will be running the tools you simply have to set the `IMPERASD_LICENSE_FILE` environment variable to point to the license server,

```
IMPERASD_LICENSE_FILE=@server1
```

where `server1` is the hostname of the license server. The command `hostname` may be used on both Windows and Linux to find out the host name of a particular computer. This example assumes that the default port (27000-27009) is being used by the license server. If the license server was set up to use a different port it must be specified before the ‘@’, for example:

```
IMPERASD_LICENSE_FILE=9999@server1
```

Here, 9999 is the port number that the license server is using.

See the respective installation instructions for Windows and Linux for examples of how to set an environment variable.

E.5 Other License File Configurations

Flexera license software has many different ways to configure it. You may already have other FlexNet licenses running on your computers. By default we have configured things to be as independent of other installations as possible, but your particular situation may require alternative configurations.

Also, Flexera license software supports many different options to control how licenses are managed than are discussed here. If you need to use any of those additional capabilities please consult the Flexera license software End Users Manual.

Searching the internet for “FlexLM End Users Manual” or “FlexNet End User Manual” should locate a copy of this manual.

E.6 The daemon options file

Use of the licenses can be customized by using a daemon options file.

To make Flexera license software use an options file, either specify the full path to the file in the license file or name the file `imperasd.opt` and place it in the same directory as the license file.

Each line of the options file controls one option. The most commonly used options are:

```
EXCLUDE feature[:keyword=value] type {name | group_name}
```

Exclude a user or host from using a license feature.

where `featurename` is the name of the feature as specified in the license file, `type` is normally `USER`, `HOST` or `GROUP`, and `name` is the name of the user or group to exclude.

```
INCLUDE feature[:keyword=value] type {name | group_name}
```

This option is the same as `EXCLUDE` but allows a user to use a license feature, but note, if any `INCLUDE` lines are used then one must be provided for each user who will use the license.

```
GROUP group_name user_list
```

Defines a name (`group_name`) for a group of users, so it can be used in other options like include & exclude. `username list` is just a list of names separated by a space.

```
RESERVE num_lic feature[:keyword=value] type {name | group_name}
```

Reserve licenses for a specific user, group or host.

where `num_lic` is the number of licenses to be reserved, and the other variables are the same as for `EXCLUDE`.

For a list of other options please consult the FlexLM End Users Manual.

Searching the internet for “FlexLM End Users Manual” or “FlexNet End User Manual” should locate a copy of this manual.

E.7 License Administration tools

There are several utilities that aid administration of the licenses and this section briefly describes the most frequently used.

`lmutil lmstat`

allows users to see:

- which daemons are running
- which users are using which features

Options:

- a - Display all information.
- A - List all active licenses.
- c - Use the given license file.
- f - List all users of featurename.
- S - List all users of a vendor's features (for example, `arcd`)
- s - Display status of clients running on given host.

`lmutil lmdown` gracefully shutdown all daemons.

Use `-c` to shutdown a specific license. Providing no arguments shuts down the license(s) specified by the environment variable `LM_LICENSE_FILE`. Use of `lmdown` should be protected to prevent unintentional loss of licenses.

`lmutil lmremove` most commonly used to remove licenses that are still checked out after a node crashes.

`lmremove` remove all instances of the specified feature by the given user on host.

`lmutil lmreread` make the license daemon reread the given license file.

This will start any new daemons that have been added and cause currently running daemons to read their license files again.

Use either `lmutil lmreread imperasd` or `lmutil lmreread -c <licensefile>`.

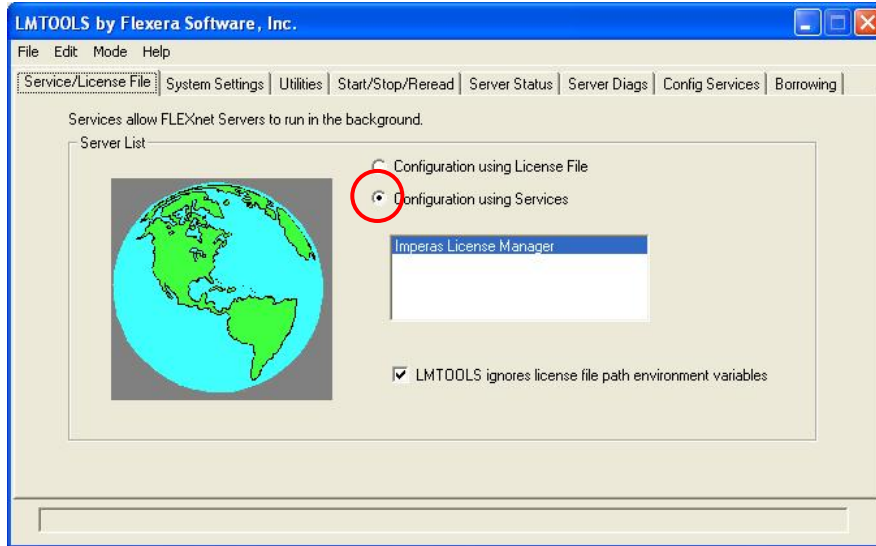
E.8 License Server Manager as a Windows Service using LMTOOLS

To configure a license server manager (`lmgrd`) as a service, you must have Administrator privileges. The service will run under the `LocalSystem` account. This account is required to run this utility as a service.

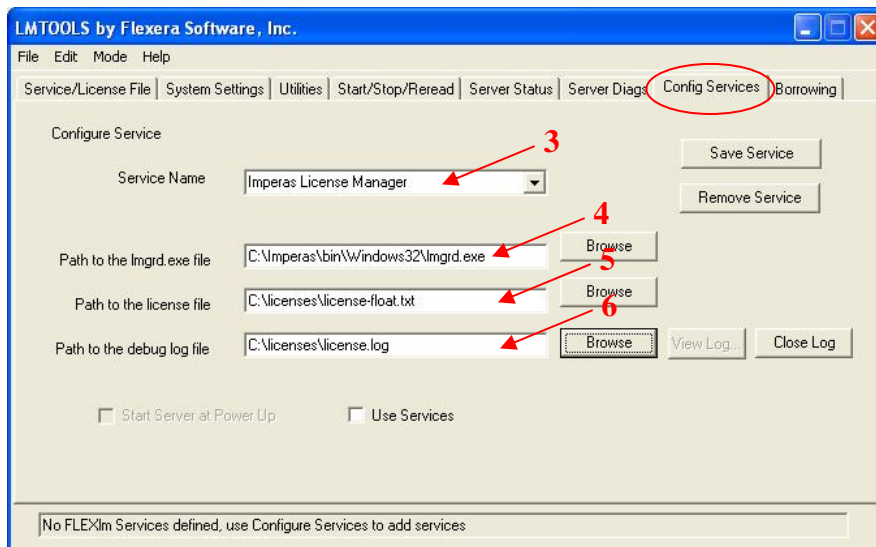
To configure a license server as a service:

1. Run the `lmtools` utility, this is found in `Imperas/bin/Windows32`

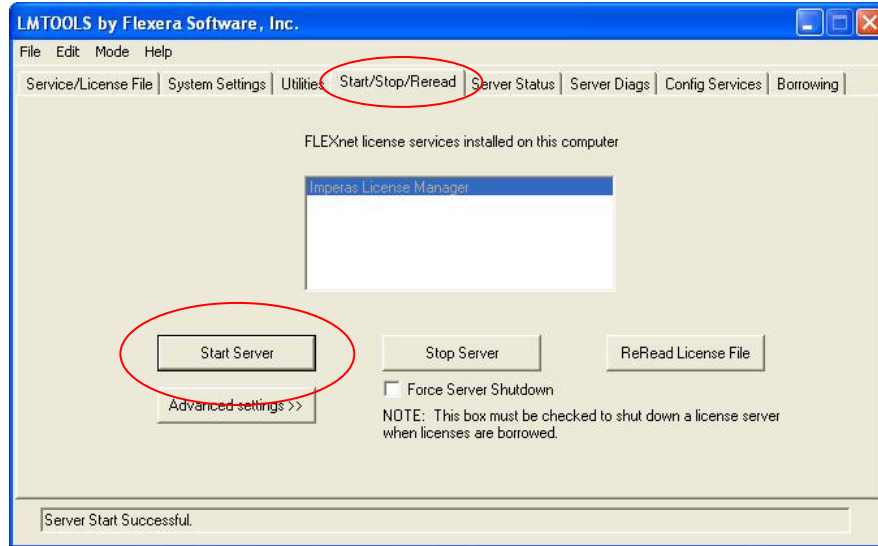
2. Click the **Configuration using Services** button, and then click the **Config Services** tab.



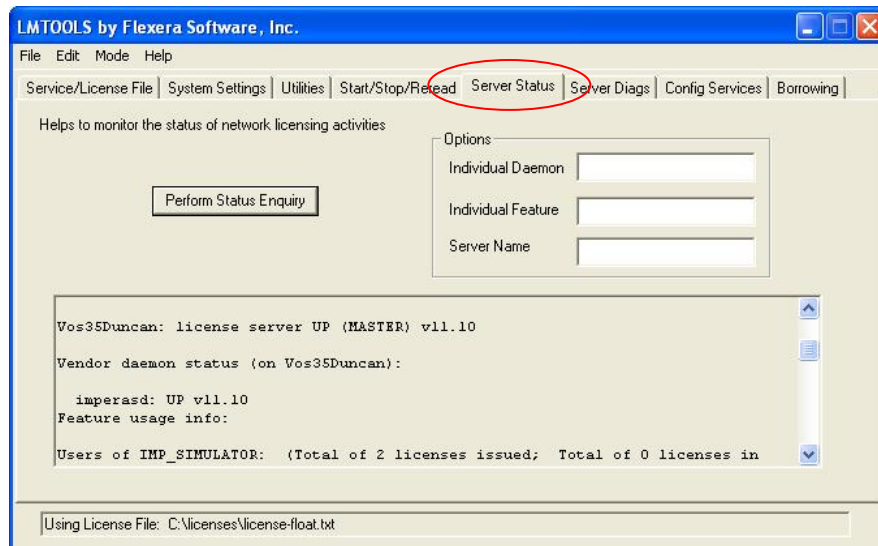
3. In the **Service Name**, type the name of the service that you want to define, for example, *Imperas License Manager*. If you leave this field blank, the service will be given a default name.
4. In the **Path to the lmgrd.exe** file field, enter or browse to lmgrd.exe for this license server.
5. In the **Path to the license file** field, enter or browse to the license file for this license server.
6. In the **Path to the debug log file**, enter or browse to the debug log file that this license server writes.



7. Start the license server



8. Check the license server status



APPENDIX F

Some Common Problems

F.1 Segmentation Fault when Native Debug of an OVP Platform

When an platform is being debugged the simulation may stop with the message

“Program received signal SIGSEGV, Segmentation fault.”

This will have occurred in the call to `opSessionInit()` when the licensing is initialized. The latest version of the Flexera license software licensing code utilized by the OVP and Imperas products contains a feature that causes a segmentation fault when attempting to run under a debugger.

This has no effect on the execution of the OVP virtual platform and it is possible to continue the simulation and debugging the platform beyond this point. Simply continue the simulation.

The following is the contents of an example `.gdbinint` script that may be used to ignore this issue

```
handle SIGSEGV nostop
handle SIGSEGV noprint
```

The OVP and Imperas simulators also may use of the floating point exceptions during simulation, these may be ignored when debugging by adding the following

```
handle SIGFPE nostop
handle SIGFPE noprint
```

Finally, additional options that we have found useful are

```
set break pending on
set backtrace limit 100
```

F.2 Simulator Reports Internal Abort (ASRT)

The simulator will only report an assertion when it has detected that the internal state is no longer valid. This can be caused by a bug in the simulator, which should be reported to Imperas including a test case¹⁵ that can be used to reproduce the problem at the Imperas site.

However, often these problems are caused when the simulator is being used incorrectly.

Asynchronous Calls to a running simulator

One such error is making asynchronous calls to a running simulator. There must be no asynchronous calls made to the running simulator. If the simulation is to be interrupted the procedure that is described in the ‘OVPsim and CpuManager User Guide’ must be followed.

If this procedure is not followed the asynchronous calls can cause the simulator to be interrupted during a system state update and hence the state can become corrupted.

When this occurs it is typical to see the following assertions reported by the simulator and for the simulation to terminate.

> Internal Abort (ASRT)

>

/home/release/build/20130630.6/Imperas/SimCommon/source/morph/codeBlock.c:1407

> : Assertion failure : bad native address bounds

> Internal Abort (ASRT)

> /home/release/build/20130630.6/Imperas/SimCommon/source/morph/xRef.c:690

> : Assertion failure : resolved cross-reference found

F.3 Building on Windows using mingw32-make

F.3.1 Error in Makefile.pse when building peripherals

There is a known bug in the version of GNU make 3.80. This version is sometimes installed as part of the MinGW installation as mingw32-make and will result in the error shown below when attempting to build a peripheral model

```
/c/Imperas/Examples/Models/Peripherals/creatingDMAC/1.registers
$ mingw32-make NOVLNV=1
```

¹⁵ A test case should include the virtual platform source code, the application code being executed and clear instructions of how to reproduce the problem. All these files should be sent as a zip or tar file. It is useful to extract the files and check in a clean directory that the instructions sent can re-produce the problem.

```
C:/Imperas/ImperasLib/buildutils/Makefile.pse:39: *** missing `endif'. Stop.  
  
/c/Imperas/Examples/Models/Peripherals/creatingDMAC/1.registers  
$
```

You can check the version of mingw32-make that you have installed with the command “mingw32-make --version”

```
$ mingw32-make --version  
GNU Make 3.80  
Copyright (C) 2002 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions.  
There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A  
PARTICULAR PURPOSE.  
  
$
```

F.4 Licensing

F.4.1 Feature Not Supported

If you see a message similar to the following:

```
Error (LIC_NE) License not available. FLEXLM reports:  
  
License server system does not support this feature.  
Feature:      IMP_OVPSIM  
License path: C:\Imperas\bin\Windows\OVPSim.lic;  
FlexNet Licensing error:-18,147  
For further information, refer to the FlexNet Licensing End User Guide,  
available at "www.macrovision.com".  
  
License server system does not support this feature.  
Feature:      IMP_OVPSIM_20130630  
License path: C:\Imperas\bin\Windows\OVPSim.lic;  
FlexNet Licensing error:-18,147  
For further information, refer to the FlexNet Licensing End User Guide,  
available at "www.macrovision.com".  
  
Fatal (LIC_FE) If you have a license from OVPworld, put it in  
$IMPERAS_HOME/OVPSim.lic  
Alternatively, set IMPERASD_LICENSE_FILE environment variable to your  
license file or server location.  
If you don't have a key, click on download at www.OVPworld.org  
Info Exiting
```

You do not have the correct license FEATURE available in your license file.

In the above output you can see that the simulator first attempts to obtain the license feature IMP_OVPSIM and then attempts to obtain the feature IMP_OVPSIM_20130630.

The feature IMP_OVPSIM may be obtained by contacting Imperas; this is a license that can be used with any version of the OVPSim simulator.

The feature `IMP_OVPSIM_20130630` is contained in licenses generated from the OVP World license page automatically. It is generated for the current product version and can only be used with that product version.

This license error could be caused by one of the following:

1. The license file has not been obtained
 - a. Check the license file does include this `FEATURE`
2. The incorrect license file is being used or the license file has been corrupted
 - a. Check the `IMPERASD_LICENSE_FILE` variable is pointing to the correct license file or license server for floating licenses
 - b. Check the license file has not been corrupted. Use a hex editor to ensure only valid ASCII characters and `<CR>`, `<LF>` are present
3. The incorrect simulator is being used
 - a. Check that `IMPERAS_RUNTIME` has been set correctly
If your license(s) are supplied by Imperas for the Imperas tools, `IMPERAS_RUNTIME` should be set to `CpuManager` so that the correct Imperas licenses are used.
4. The simulator personality is not set correctly
 - a. Check that `IMPERAS_PERSONALITY` has been set correctly. This variable select the product you have purchased and selects the license feature that will be requested. See section 4.

F.4.2 Long Delay (30 second) at Simulator Start Up

If you experience a long delay when you start one of the Imperas products, this is most likely due to the product trying to request a license to run, and there being a problem in connecting to a defined license server.

The `IMPERASD_LICENSE_FILE` variable is set to a list of servers. If one of these servers does not exist, the license software will wait for (typically) 30 seconds (to give a server over a network time to respond) before continuing to the next server.

If you experience a delay at start up check that all the servers listed in your `IMPERASD_LICENSE_FILE` variable list really exist and are functioning correctly. Remove any non-existent servers from the list.

F.4.3 Cannot Access Date Server

If you see an error message such as :

```
Error (LIC_SRVNS) License not available. FLEXLM reports:
Bad Status connecting to internet date server.
Feature: IMP_OVPSIM_20120313
FlexNet Licensing error:+97,210
Fatal (LIC_SRVNS) Exiting:
Info Exiting
```

When using a version locked license, the simulator will access a date server. This error will occur if the simulator is not able to connect to the internet.

You may see this message if you are using an internet proxy server. In this case you need to inform the Imperas products of the proxy server using the `IMPERAS_PROXY_SERVER` environment variable. This is set to an `<address>:<port>` pair for the proxy server access.

For example

```
export IMPERAS_PROXY_SERVER=192.168.2.1:1234
```

F.5 Ethernet Adapter Naming/Hostid is zero

If you see an error such as:

```
Invalid host.
The hostid of this system does not match the hostid
specified in the license file.
Feature: IMP_OVPSIM_20100528.0
Hostid: 112233445566
License path: /home/Imperas.20130630/OVPsim.lic
FlexNet Licensing error:-9,57. System Error: 19 "(null)"
For further information, refer to the FlexNet Licensing End User Guide,
available at "www.macrovision.com".
```

This problem occurs when the license system checks the `hostid` in the license file against the `hostid` of the license server. The `hostid` is the MAC address of the server's ethernet device named `eth0`.

If the network interface is enabled on a different device, e.g. `eth1` the license system does not read a valid MAC address and a NULL host Id is reported. This may be caused because on modern operating systems Consistent Network Device Naming may be used. See the following link for further information

<http://fedoraproject.org/wiki/Features/ConsistentNetworkDeviceNaming>

The solution is to enable the network interface on `eth0` or to rename the network interface to `eth0`.

There are a number of ways this can be achieved, the following was taken from <http://www.science.uva.nl/research/air/wiki/LogicalInterfaceNames>

F.5.1 How to change the default 'ens33' network device to old 'eth0'

The following URL explains the full procedure

<http://unix.stackexchange.com/questions/81834/how-can-i-change-the-default-ens33-network-device-to-old-eth0-on-fedora-19>

The minimum set of stages that were required when performed are:

1. Edit `/etc/default/grub`. Note you need root privileges to edit this file.
2. At the end of `GRUB_CMDLINE_LINUX` line append `net.ifnames=0 biosdevname=0`
3. Save the file
4. type the command `grub2-mkconfig -o /boot/grub2/grub.cfg`
5. type the command `reboot`

F.5.2 How to reorder or rename logical interface names in Linux

On Linux systems the start-up order of network interfaces is unpredictable. It is usually consistent between reboots, but can change after an upgrade to a new kernel or the addition or replacement of a network card. For example, what used to be `eth0` could become `eth1` or `eth2`.

While there is some logic to which network interface gets which name, Linux documentation states that this may change, and users or programs should not assume a particular order. This is inconvenient, in particular if your management interface is at `eth1` at one node in a cluster and at `eth2` in another node of the same cluster (which we have experienced).

There are ways to achieve consistency. They can be divided in four methods:

1. Order the network interfaces based on physical properties of the NIC. (e.g. the physical location in the machine)
2. Order the network interfaces based on the MAC address of the NIC.
3. Order the network interfaces based on the driver of the NIC.
4. Order the network interfaces based on the physical location of the NIC in the computer

So you have to pick a method that suits your system. Imperas recommends to use `ifrename` (based on physical properties, especially useful if you often change network cards in your hosts) or writing a `udev` rule (based on the MAC address).

Note: Linux kernels up to 2.4 did only probe for the first Ethernet card, ignoring other NICs. We assume you use a 2.6 or higher kernel or already fixed this, for example by specifying `ether=0,0,eth1` as kernel parameter[∞].

F.5.3 Based on the physical properties

Perhaps the most elegant way to name the Ethernet NIC is to do so based on their physical properties, like link speed and port type.

F.5.3.1 Using the ifrename tool

`ifrename` is a tool specifically designed to (re)name network interfaces based on characteristics like MAC address (wildcards supported), bus information, and hardware settings. It uses a control file `/etc/iftab` to specify rules about how the interfaces will be named.

```
# Example /etc/iftab file
eth2          mac 08:00:09:DE:82:0E
eth3          driver wavelan interrupt 15 baseaddress 0x390
eth4          driver pnet32 businfo 0000:02:05.0
# wildcard name: pick the lowest available name of air0, air1, air2, etc.
air*          mac 00:07:0E:* arp 1
```

F.5.3.2 Using the ethtool and ip programs

It is possible to check the NIC properties using the `ethtool` program, and to change the name using the `ip` program:

```
if ethtool eth0 | grep -q "Port: FIBRE"; then
    ip link set dev eth0 name not_eth0
    ip link set dev eth1 name eth0
    ip link set dev not_eth0 name eth1
fi
```

The disadvantage of `ethtool` is that it can only be run by root, even when you're only using it to query for information.

F.5.4 Based on the MAC address

Secondly, it is also possible to name the network interface based on the MAC address of each NIC. The advantage is that it is possible to use this method if you have two NICs which use the same driver (unlike the next method: based on driver).

First, you must determine the MAC address of your interfaces. You can do this locally on a machine running

```
ifconfig -a
```

The MAC address is listed as `hwaddr` (hardware address). Alternatively, you can even determine MAC addresses remotely using `ping` and `/sbin/arp`.

There are three ways to map the MAC address to the logical interface name. Either by using the `udev` rules, with the `get-mac-address.sh` script, or by using the `nameif` program.

The `udev` method should work on all recent Linux distributions, and is recommended. The `get-mac-address.sh` script and the `nameif` program are known to work with Debian, while on Red Hat, you can change the interface configuration file.

F.5.4.1 Using udev rules

`udev` replaced `devfs` in Linux 2.6. First make sure that your Linux system has `udev` installed, rather than `devfs`. If you have a `/etc/udev` directory, but not `/etc/devfs` directory, you are probably fine. If not, be aware that changing your kernel from `devfs` to `udev` is possible, but is not just a matter of adding a new module. Perhaps for now, another method is easier for you.

Now that you have `udev`, it is rather simple. You only need to create a `udev` rule mapping the MAC address to the interface name. Store this in a file inside the `/etc/udev/rules.d/` directory:

```
KERNEL=="eth?", SYSFS{address}=="00:37:e9:17:64:af", NAME="eth0" # MAC
of first NIC in lowercase
KERNEL=="eth?", SYSFS{address}=="00:21:e9:17:64:b5", NAME="eth1" # MAC
of second NIC in lowercase
```

Most distributions already come with an example config file for you.

E.g. `/etc/udev/rules.d/network-devices.rules`
or

`/etc/udev/rules.d/010_netinterfaces.rules`.

More information can be found

at http://www.reactivated.net/writing_udev_rules.html or <http://www.debianhelp.co.uk/udev.htm>. (Thanks to Casey Scott and Ron Hermesen for the pointers.)

Another possible rule is:

```
SUBSYSTEM=="net", DRIVERS=="*", ATTRS{address}=="00:16:3e:00:02:00",
NAME="eth0"
SUBSYSTEM=="net", DRIVERS=="*", ATTRS{address}=="00:16:3e:00:02:01",
NAME="eth1"
```

I'm not sure about the difference between these rules. Information is welcome (please leave a comment below)

F.5.4.2 Using the interface configuration file

If you run a Red-Hat-based distribution, you can simply add the MAC address in the interface configuration file `/etc/sysconfig/network-scripts/ifcfg-eth0`:

```
DEVICE=eth0
HWADDR=00:37:e9:17:64:af
```

You can give it any `DEVICE` name you want, like `DEVICE=ethmngmt`, as long as you remember to rename the config file:

```
/etc/sysconfig/network-scripts/ifcfg-ethmngmt
```

Source: <http://forums.serverwatch.com/showthread.php?t=18476>

F.5.4.3 Using the `get-mac-address.sh` script

Another solution is to use the `get-mac-address.sh` script to map interface names by MAC address. On Debian, this script is distributed as part of the `ifupdown` package (in `/usr/share/doc/ifupdown/examples/get-mac-address.sh`). Copy this script to a saner place (e.g. `/usr/local/bin`), and you can setup `/etc/network/interfaces` in this manner:

```
auto lo eth0 eth1

iface lo inet loopback

mapping eth0 eth1
    script /usr/local/bin/get-mac-address.sh
    map 00:37:E9:17:64:AF netA
    map 00:21:E9:17:64:B5 netB

iface netA inet static
    address etc...

iface netB inet static
    address etc...
```

Source: <https://www.gelato.unsw.edu.au/archives/gelato-technical/2004-February/000334.html>

The disadvantage of this method is that defines a mapping, rather than changing the actual logical interface name.

F.5.4.4 Using the `nameif` program

Alternative to the `get-mac-address.sh` script, you can also use the slightly more convenient `nameif` program, which is distributed as part of the `net-tools` package on Debian.

The advantage of `nameif` is that you can specify the interface names in the `/etc/mactab` file:

```
ethmngnt 00:37:E9:17:64:AF
ethwireless 00:21:E9:17:64:B5
```

It is not possible to rename an interface to a name of an existing interface. So you can't

rename `eth1` to `eth0` as long as `eth0` still exists. It is possible to still swap the names `eth0` and `eth1` by using a temporary name (e.g. first rename `eth1` to `ethfoo`, then `eth0` to `eth1` and finally `ethfoo` to `eth0`). Note that this method may lead to problems if you use common names such as `eth0` and `eth1`. If you upgrade a kernel, the names may be different than you expected, and you may rename a NIC to `eth0` while `eth0` still exists, leading to name collisions. Therefore, it is recommended to use other names like `ethmgmnt`, `ethwired`, `ethwireless` and `eth10ge`, as shown in the example above.

F.5.5 Based on the driver

Warning: This only works if the driver is available as a loadable module. Not if it is built into the kernel.

This is a relative easy method, since it does not rely on external scripts. The idea is to just load the kernel module for your `eth0` interface before the modules for other network cards.

First of all, you must determine which driver is used for each network card. Linux does have a system to load the appropriate driver automatically, based on the PCI ID of the network card. There is no single command to simply get the driver (and other information like the link speed) based on just the interface name in Linux. Look for kernel messages:

```
dmesg | grep eth
```

This should give you a the driver name. You can verify if the name indeed does exist and is loaded:

```
lsmod
```

Note that running `modprobe tg3` en then `modprobe e1000` does start them in the correct order, with the correct interface names. This is a good check if this approach (using the driver to decide the interface name) can work.

F.5.5.5 Red Hat

In Red Hat, if the driver is called `tg3` (the Tigon driver), you specify the network name by adding this entry in `/etc/modules.conf`:

```
~alias eth0 tg3
```

F.5.5.1 Debian

On a Debian system, `/etc/modules.conf` is generated automatically and should not be edited directly. Instead, create a file in the subdirectory `/etc/modules/` (do not use `/etc/modprobe.d/`). For example, create the file `/etc/modutils/interfaces` and add the appropriate modules.

```
alias eth0 tg3
alias eth1 e1000
```

Next, update `/etc/modules.conf` by running:

```
update-modules
```

In some scenarios the kernel did already load the modules for the drivers, even before `/etc/modules.conf` was read. The result was that in effect, the specification in `/etc/modules.conf` was ignored, and this method did not work. As an alternative, it is possible to also list the drivers, in the appropriate order, in `/etc/modules` (thus not `/etc/modules.conf`):

```
tg3
e1000
```

The result will be that the `tg3` driver is loaded before the `e1000` kernel. Since `/etc/modules` only exists for Debian, this will not work for other distributions.

F.5.6 Based on the physical location in the computer

Warning: This only works if the driver is built into the kernel, not as a loadable module.

Note: It is relatively hard to get this to work, and we encountered problems with it. The other methods are recommended.

It is possible to name the network interfaces based on the interrupt (IRQ) and memory address. This should work if you have network cards in PCI busses, and it involves appending the proper parameters to the "ether=" or "netdev=" kernel parameters.

Detect the PCI slot of the devices using

```
lspci -v
```

This is reported to fail sometimes for certain cards. Now, write down the IRQ and IO (memory) address of each network card, and use this information to specify the interface name in your LILO or GRUB configuration file.

For LILO, you add a line to the appropriate boot configuration. For example:

```
append="netdev=irq=21,io=0x2040,name=eth0  
netdev=irq=20,io=0x3000,name=eth1 netdev=irq18,io=0x2000,name=eth2"
```

Under grub, it can just be listed as parameter. e.g.:

```
kernel /boot/vmlinuz netdev=irq=24,name=eth0
```

F.5.7 Ubuntu directly changing logical names

The following information is taken from the page found at <https://help.ubuntu.com/10.04/serverguide/network-configuration.html>

The Ethernet interface logical names are configured in the file

```
/etc/udev/rules.d/70-persistent-net.rules
```

To control which interface receives a particular logical name, find the line matching the interfaces physical MAC address and modify the value of `NAME=ethX` to the desired logical name. Reboot the system to commit the changes.

F.5.8 More Information

<http://www.tldp.org/HOWTO/Ethernet-HOWTO-8.html>

<http://www.tldp.org/HOWTO/BootPrompt-HOWTO-11.html>

F.6 What to try if license server (lmgrd) fails to run

When you try and run the license server

on Ubuntu you get an error

```
./lmgrd: No such file or directory
```

On CentOS you get an error

```
-bash: ./lmgrd: /lib/ld-lsb.so.3: bad ELF interpreter: No such file or  
directory
```

The license manager daemon (the version used by OVP/Imperas) is not officially supported on Ubuntu or CentOS, it is supported on the RedHat Enterprise Linux distribution. It can, however, be used on the Ubuntu and CentOS Linux operating system distributions.

The `LSB` support package is required, if it is not already present it may be obtained using the command

On Ubuntu

```
sudo apt-get install lsb
```

On CentOS

```
yum -y install redhat-lsb
```

On Debian

```
$bash: no such file or directory: ./lmgrd
```

The lsb library naming is not conventional and so may not be found by the license server. Setting a link between the expected name and the actual may resolve the issue

```
sudo ln -s /lib64/ld-linux-x86-64.so.2 /lib64/ld-lsb-x86-64.so.3
```

APPENDIX G

Abort Detected in Program

The Imperas and OVP products contain a function to trap aborts. This may trap aborts caused anywhere in the execution flow i.e. it may not be specifically within the Imperas product but in a platform or shared object loaded by the Imperas product.

The Imperas/OVP simulators load shared objects for processor models, peripheral models and intercept (including semihost) libraries so it is possible that a segmentation fault or abort can be caused in the loaded code.

This may appear to be a fault in the simulator but it is more likely to be a fault in one of the models or platforms that are loaded by the simulator. The following can be used to help find the problem in the software that you have created.

An abort will be reported in the following manner. For an example, an abort has been caused in the virtual platform when run using the command line

```
platform.Linux32.exe -program dhrystone.ARM_CORTEX_A15.elf
```

The abort is reported during the run as follows

```
SystemC 2.3.0-ASI --- Jun  6 2013 16:33:50
Copyright (c) 1996-2012 by all Contributors,
ALL RIGHTS RESERVED

CpuManager (32-Bit) v20190306.0 Open Virtual Platform simulator from
www.IMPERAS.com.
Copyright (c) 2005-2013 Imperas Software Ltd.  Contains Imperas Proprietary
Information.
Licensed Software, All Rights Reserved.
Visit www.IMPERAS.com for multicore debug, verification and analysis solutions.

<snip>

Internal Abort (ABRT) Imperas/Common/source/targetOS/tosSignal.c:656 : Abort
reached.
Uncaught Exception (SIGSEGV) at 0x5b5d30

This could be due to an error in your native code (for example the platform,
semihost library etc.) or an error in the simulator.
Use the following approaches to try to find the source of the error:
1. Rerun the simulation under a debugger.
2. Set environment variable IMPERAS_BACKTRACE=1 to generate a backtrace (Linux
only).
3. Set environment variable IMPERAS_LOOP_ON_EXCEPTION=1 to cause the simulator
to enter a wait loop on exception, so that a debugger can be attached.
```


If you believe the error originates within the simulator, please contact Imperas support (support@imperas.com)
Info Exiting

To aid in tracking down the cause of the abort there are two features, explained in the abort output

G.1 Obtaining Backtrace

A backtrace can be generated (on Linux only) which may help you determine if the error originates in the code that you have added. The following shows the same platform run with the IMPERAS_BACKTRACE=1 set on the command line, i.e.

```
IMPERAS_BACKTRACE=1 platform.Linux32.exe dhrystone.ARM_CORTEX_A15.elf
```

And this results in the following extra information

```
*** backtrace 19 stack frames:
-----
/home/graham/Imperas/bin/Linux32/libCpuManager.so(+0x8923b) [0x2e523b]
/home/graham/Imperas/bin/Linux32/libCpuManager.so(+0x892e5) [0x2e52e5]
/home/graham/Imperas/bin/Linux32/libCpuManager.so(+0x8932a) [0x2e532a]
/home/graham/Imperas/bin/Linux32/libCpuManager.so(+0x8930c) [0x2e530c]
/home/graham/Imperas/bin/Linux32/libCpuManager.so(+0x1db4eb) [0x4374eb]
/home/graham/Imperas/bin/Linux32/libCpuManager.so(+0x1db64b) [0x43764b]
[0xf1e40c]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(bindingEv+0x49) [0x18ff19]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(elab_doneEv+0x1f) [0x19012f]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(elabEv+0x14c) [0x174bfc]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(initEb+0x30) [0x176b60]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(_7scimeE+0x2d) [0x176c2d]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(20sc_stvpole+0xdc) [0x1778ec]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(_startEv+0x73) [0x177b83]
platform.Linux32.exe(sc_main+0x1d5) [0x805109f]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(sc_elab+0xa4) [0x163b94]
/user/systemc-2.3.0/lib-linux/libsystemc-2.3.0.so(main+0x32) [0x163a02]
/lib/libc.so.6(__libc_start_main+0xe6) [0xad1bb6]
platform.Linux32.exe(__gxx_person_v0+0x159) [0x8050841]

Internal Abort (ABRT) Imperas/Common/source/targetOS/tosSignal.c:630 : Abort
reached.
Uncaught Exception (SIGSEGV) at 0x18fd30

Info Exiting
```

G.2 Connecting a Debugger to a running simulation

The simulator can be forced to wait at exit for a debugger to be connected.

The following shows the same platform run with the environment variable IMPERAS_LOOP_ON_EXCEPTION=1 set on the command line, i.e.

```
IMPERAS_LOOP_ON_EXCEPTION=1 platform.Linux32.exe dhrystone.ARM_CORTEx_A15.elf
```

This simulation will not finish. In a separate shell you can now start a suitable host debugger and examine the running processes on the host for the one running your platform executable, `platform.Linux32.exe` in this case.

APPENDIX H

Other Windows Development Environments

H.1 Using a Cygwin Environment

H.1.1 Use MINGW GNU Toolset

Although, the MSYS environment is recommended it is also possible to make use of the Cygwin environment.

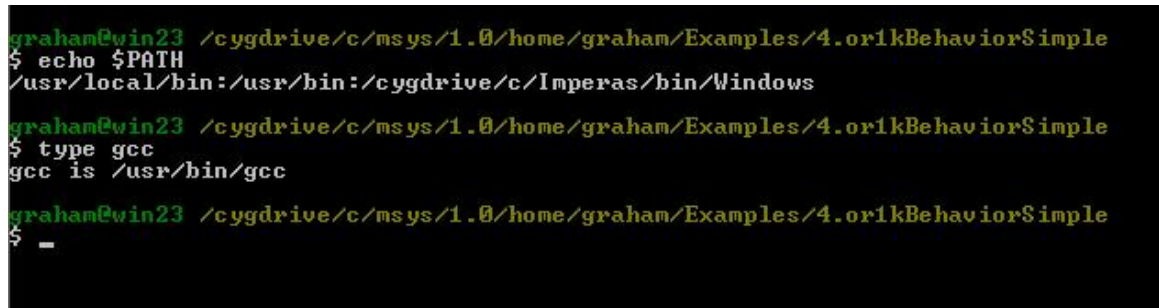
If an installation of cygwin is used as the environment it must be ensured that the MINGW GNU tools are used during the build of any application or DLL.

Using the MINGW GNU toolset provides a native Windows executable.

Using the Cygwin GNU toolset generates executables and DLLs that will reference and be reliant on the Cygwin DLLs. This will create executables and DLLs that are not portable and therefore not compatible in other environments. There are compatibility issues even between different versions of Cygwin.

H.2.2 Verify use of MINGW GNU Toolset

To verify that the correct environment is being used check the version of gcc that is found on the path in a cygwin shell. The following illustrates that the incorrect version of gcc would be used is from the Cygwin install.



```
graham@win23 /cygdrive/c/msys/1.0/home/graham/Examples/4.or1kBehaviorSimple
$ echo $PATH
/usr/local/bin:/usr/bin:/cygdrive/c/Imperas/bin/Windows

graham@win23 /cygdrive/c/msys/1.0/home/graham/Examples/4.or1kBehaviorSimple
$ type gcc
gcc is /usr/bin/gcc

graham@win23 /cygdrive/c/msys/1.0/home/graham/Examples/4.or1kBehaviorSimple
$ _
```

We should modify the path so that the MINGW GNU tools appear before the Cygwin tools.

```

graham@win23 /cygdrive/c/msys/1.0/home/graham/Examples/4.or1kBehavior$imple
$ export PATH=/cygdrive/c/msys/1.0/mingw/bin/:$PATH

graham@win23 /cygdrive/c/msys/1.0/home/graham/Examples/4.or1kBehavior$imple
$ type gcc
gcc is /cygdrive/c/msys/1.0/mingw/bin/gcc

graham@win23 /cygdrive/c/msys/1.0/home/graham/Examples/4.or1kBehavior$imple
$ _

```

We can see now that the GNU tools are being found from the MINGW installation before the Cygwin.

Note that this installation of MINGW has been made under an MSYS installation.

Note that when creating Makefiles or scripts to build executables and DLLs it is often useful to use MINGW specific names of the GNU tool executables. These appear in the mingw/bin directory as, for example, mingw-gcc.exe. By doing this you will guarantee that you do not inadvertently pick up the Cygwin tools. There are mingw-* named copies of all the GNU tools.

```

92160 Jan 18 2006 g++.exe
89600 Jan 18 2006 gcc.exe
16031 Jan 18 2006 gccbug
95232 Jan 18 2006 gcj.exe
77824 Jan 18 2006 gcjh.exe
26112 Jan 18 2006 gcov.exe
3025408 Jan 18 2006 gij.exe
625174 Aug 25 2006 gprof.exe
86016 Jan 18 2006 grepjar.exe
70656 Jan 18 2006 jar.exe
96256 Jan 18 2006 jcf-dump.exe
3028480 Jan 18 2006 jv-convert.exe
957440 Jan 18 2006 jv-scan.exe
806237 Aug 25 2006 ld.exe
92160 Jan 18 2006 mingw32-c++.exe
92160 Jan 18 2006 mingw32-g++.exe
89600 Jan 18 2006 mingw32-gcc-3.4.5
89600 Jan 18 2006 mingw32-gcc.exe
95232 Jan 18 2006 mingw32-gcj.exe
77824 Jan 18 2006 mingw32-gcjh.exe

```

H.3.3 Cannot Build OVPsim Examples

The Mingw gnu tools are native Windows tools and as such are not expecting Cygwin path names.

There is a utility `cygpath` that will convert a Cygwin path into a native windows path. This may be used to give the correct path for the MINGW GNU tools.

```
cygpath -w $(pwd)
```

```
graham@win23 /cygdrive/c/msys/1.0/home/graham/peripheralSemiHost
$ pwd
/cygdrive/c/msys/1.0/home/graham/peripheralSemiHost

graham@win23 /cygdrive/c/msys/1.0/home/graham/peripheralSemiHost
$ cygpath -w $(pwd)
c:\msys\1.0\home\graham\peripheralSemiHost

graham@win23 /cygdrive/c/msys/1.0/home/graham/peripheralSemiHost
$ -
```

Unfortunately this cannot be used in conjunction with the Makefile system provided with the OVPsim examples because the Windows native path contains a ‘.’ which is a special character when used in Makefiles!

The OVPsim examples are all verified for use in the MSYS environment.

H.4 Building Virtual platforms with Microsoft MSVC

The Microsoft Visual C++ environment may be used to build the virtual platform executable for a Windows host.

An example is provided in the Platform Examples section to show the compilation of a virtual platform using MSVC.

H.4.1 Running the Example

Take a copy of the example:

```
$ cp -r $IMPERAS_HOME/Examples/PlatformsICM/simple_MSVC_compile .
```

H.4.1.1 Compiling the CpuManager Platform

The test platform can be compiled to produce an executable, `platform.${IMPERAS_ARCH}.exe`, using the MSVC environment.

To compile using MSVC the setup program must be invoked that is provided with MSVC to start a shell with an appropriate environment. This is typically called a ‘Visual Studio 2008 Command Prompt’

A script in the platform directory, `compile.msvc.bat`, contains the following command line that is used to build using `nmake` and the Makefile provided.

```
$ nmake -nologo -s -f nmakefile
```

Note that for completeness this example can also be built in a Linux shell or in a MinGW shell on Windows by using the following command in the `platform` directory:

```
$ make -C platform
```

H.4.1.2 Creating an Application Executable

A test case must be created using the processor tool chain. Because the OR1K processor is supported by Imperas tools and shipped as an example, there is already an encapsulated tool chain that you can use to compile test cases for it.

Within the `platform` directory is a simple assembler test, `application/asmtest.S`, which simply performs a few instructions and exits. The application can be compiled using the following command in the `platform` directory:

```
$ make -C application
```

The result is an ELF format file for the OR1K called `asmtest.OR1K.elf`.

H.4.1.3 Running the Simulation

Having compiled the test platform and application, you are now ready to run a simulation. Do this by running the following in the `example` directory:

```
platform/platform.${IMPERAS_ARCH}.exe --program application/asmtest.OR1K.elf
```

You should see the output:

```
Processor 'cpul' terminated at 'exit', address 0x10000bc
```

This message is printed by the `imperasExit` semihosting library as the processor executes the first instruction at `exit` in the application.

##