Imperas Tools Overview

This document provides an overview of the
OVP and Imperas tools environment.

Imperas Software Limited

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

Author:	Imperas Software Limited
Version:	2.0.2
Filename:	Imperas_Tools_Overview.doc
Project:	Imperas Tools Overview
Last Saved:	January 22, 2021
Keywords:	Imperas Tools Overview

Copyright Notice

Right to Copy Documentation

Destination Control Statement

Disclaimer

Table of Contents

1 Preface

This document describes the Imperas Professional Tools and provides an introductory overview. There are specific documents that relate to the usage of each tool.

1.1 Notation

`Code` Text representing a code extract or command line
keyword A word with special meaning.

1.2 Related General Documents

The following document is part of the OVP and Imperas installations and can be found in the directory:

$IMPERAS_HOME/doc

- Imperas Installation and Getting Started

1.3 Related OVP Documents

The following documents are part of the OVP and Imperas installations and can be found in the directory:

$IMPERAS_HOME/doc/ovp

- Imperas Tools Overview (this document)

1.3.1 Platform related

- Writing Platforms and Modules in C User Guide
- Using OVP Models in SystemC TLM2.0 Platforms
- iGen Model Generator Introduction
- iGen Platform and Module Creation User Guide
- iGen Peripheral Template Generator User Guide
- Online (doxygen) documentation of iGen API from headers

1.3.2 Simulation related

- Simulation Control of Platforms and Modules User Guide
- Advanced Simulation Control of Platforms and Modules User Guide
- Control File User Guide
- Guide to Using Processor Models
- Online (doxygen) documentation of OP API from headers

1.3.3 Debugger related

- Debugging Applications with GDB User Guide
- eGui Eclipse User Guide
- Debugging Applications with Eclipse User Guide
- Debugging with ARM DS-5

- Debugging with INSIGHT User Guide

1.3.4 Writing Behavioral Peripheral Models

- Peripheral Modeling Guide
- BHM PPM Function Reference
- Online (doxygen) documentation of BHM/PPM API from headers

1.3.5 Writing CPU Models

- Processor Modeling Guide
- VMI Morph Time Function Reference
- VMI Run Time Function Reference
- VMI Memory Model Component (MMC) Function Reference
- VMI OS Support Function Reference
- VMI View APIs Function Reference
- Online (doxygen) documentation of VMI API functions from headers

1.4 Related Imperas Documents

The following documents for the Imperas Professional Tools are part of the Imperas installation and can be found in the directory:

$IMPERAS_HOME/doc/imperas

- Imperas Binary Interception Technology User Guide
- Imperas Custom Object Reader
- Imperas Multi-Processor Debugger (MPD) User Guide
- Online (doxygen) documentation of MPD Command Reference from headers
- Imperas CPU Helper Developers Guide
- Imperas VAP Tools User Guide
- Online (doxygen) documentation of VAP API functions from headers

2 Introduction

Imperas simulation technology enables very high performance simulation, debug and analysis on virtual platforms containing multiple processors and peripheral models.

This document gives an overview of the OVP tools and provides an introduction to the Imperas Professional Tools. The Imperas tools are a superset of the OVP simulation technologies, adding additional capabilities for advanced simulation, debug, software verification, analysis and profiling, and productivity.

2.1 The different tools

Imperas develops tools for software developers to use virtual platforms/prototypes (simulation models) to develop embedded software.

All the tools make use of the capabilities in the over 250 different processor, peripheral and platform models available from OVPworld.org.

Subsequent chapters provide an overview of each of these tools. They can be classified as follows:

2.1.1 Simulators

- Imperas ISS - Imperas Instruction Set Simulator - processors/memory
- Imperas CpuManager - Imperas professional simulator - full functionality
- OVPsim - OVP simulator with subset of full functionality
- harness.exe - control program to simulate platforms / modules

2.1.2 Debuggers

- Imperas MPD - Imperas Multi-Processor Debugger with command line
- eGui Eclipse based debugger - encapsulation of GDB/MPD in Eclipse

2.1.3 Productivity tools

- iGen - script based platform, module, peripheral model generator
- Imperas cpuGen - script based CPU model template generator

2.1.4 Verification Analysis Profiling (VAP) tools

- Imperas CpuHelper - tools that understand personalities of different processors
- Imperas OsHelper - tools that understand personalities of different operating systems
- Imperas VAP tools - advanced tools for code/instruction coverage, function/instruction profiling, memory/cache analysis, functional coverage, fault injection, instruction/function/task tracing, OS scheduler analysis, execution profiling, protocol verification, shared resource introspection

2.2 Prerequisites

To create virtual platforms and models the reader should be familiar with the C programming language and general debugging techniques. Familiarity with the Gnu debugger GDB would be beneficial.

2.3 Host platform support

All the Imperas tools are available on Windows and Linux operating systems.

To see specific hardware requirements and which versions of host operating systems are supported, please refer to the Imperas Installation and Getting Started Guide.

3 Imperas Simulation Tools

3.1 Imperas Instruction Set Simulator (ISS)

The Imperas ISS, *iss.exe*, is a standalone executable that performs the following tasks:

- Locate and loads CPU models from the library
- Load application code to run on the built-in platforms
- Modify the behavior of the platforms and models by changing run-time switches
- Load the OVPsim or CpuManager objects and simulate the platform
- Load a semihost library to allow application code to interact with host computer
- Invoke a GDB or MPD debugger to enable source code debug
- Invoke eGui to control the GDB/MPD debuggers using a GUI
- Report performance statistics when simulation is complete

The Imperas ISS can be used to simulate application code in bare metal environments by just loading up a cross compiled elf file and selecting a CPU variant. There are configuration options to locate memory and select other parameters.

As with all Imperas tools there is a command line argument '*--help'* that lists all the options available.

There are also options to see what CPU models are available:

```
> iss.exe --showlibraryprocessors
```

and what specific variants of those models can be simulated:

```
> iss.exe --processorname arm --showvariants
```

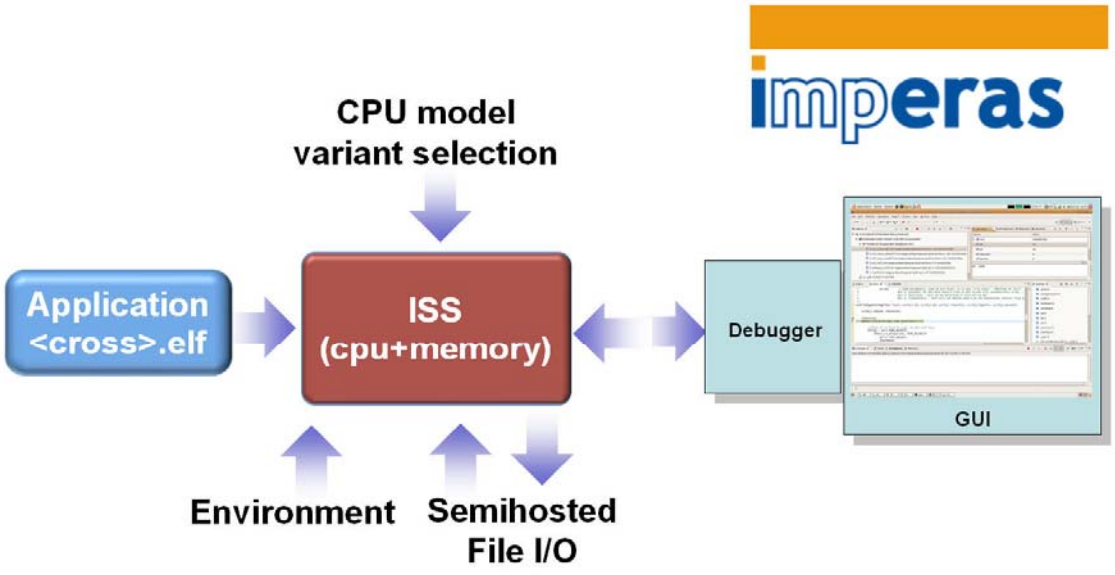
Note you need to specify which VLNV library vendor you want to see the variants from.

The simplest usage of the ISS is to select a processor and application:

```
> iss.exe --processorvendor arm.ovpworld.org \
       --processorname arm --variant Cortex-A72MPx1 \
        --program application.AARCH64.elf
```

The arguments *--processorvendor*, *--processorname*, *--variant* select which processor variant to use, and the *--program* defines the application to load.

The OVP and Imperas installations provide many examples of the ISS and it is used to explore the capabilities of the OVP provided CPU models and can be used for single processor, dual processor, and many processor examples. See *Demo/Processors* to run the ISS on the different CPU variants.



Imperas Instruction Set Simulator (ISS)

3.2 CpuManager simulation object

CpuManager is a run-time library. It is dynamically linked into a stand-alone executable program (Windows or Linux) which builds and simulates a platform. It can also be used to integrate Imperas simulation functionality into other simulation systems.

CpuManager is controlled from an API which consists of functions (prefixed by ‘op' (previously it was controlled from an API prefixed 'icm')) which fall into the following categories:

- Loading models of components (memories, processors, peripherals)
- Loading application code
- Modifying model behavior
- Overlaying extra functionality
- Loading extension libraries for analysis, verification and profiling
- Progressing the simulation
- Reporting results
- Stimulating and monitoring nets
- Modifying memory contents
- Acting when memory is modified

The OP API, fully supported by CpuManager, is described in the OVP documents:

- Writing Platforms and Modules in C User Guide
- Simulation Control of Platforms and Modules User Guide
- Advanced Simulation Control of Platforms and Modules User Guide
- Online (doxygen) documentation of OP API from headers

3.2.1 Imperas OEMsim simulation component

There is a version of CpuManager that is licensed to other simulator suppliers to enable their access to OVP models. It provides a subset of the simulation capabilities of CpuManager and has the additions of various APIs that enable efficient control and debug from within those other simulation environments. For more information, please contact info@imperas.com.

3.3 OVPsim

OVPsim is a basic version of CpuManager. OVPsim has been made available as part of the Imperas sponsored Open Virtual Platforms initiative. It is available for download from the www.OVPworld.org website.

Both OVPsim and CpuManager can simulate the same platforms, however many of the advanced features of CpuManager have not been implemented in OVPsim. For example, support for heterogeneous virtual platforms, many of the VMI API calls including extension libraries, and multi-processor debug are not available in OVPsim.

The environment variable *IMPERAS_RUNTIME* determines whether the OVPsim or CpuManager library is loaded into a program at runtime, by default, the OVPsim runtime is loaded.

3.4 harness.exe

The harness program is provided with Imperas and OVP installations, and is like the ISS, but instead of simulating just a processor/memory it can load and simulate a complete, defined platform without the user needing to write a complex controlling test bench or harness.

For many platforms and modules the *harness.exe* program will be sufficient to load and execute a simulation and it is not necessary to write your own C testbench / harness.

The simplest mode of invocation is to load a pre-compiled platform (or platform sub-module) and application for the processors to execute:

```
> harness.exe --modulefile module/model.so \
       --program application/application.OR1K.elf
```

The argument --*modulefile* <*filename*> states which shared object module to load.

The argument --*program* <*filename*> states which application program binary to load and run on the simulated processor(s) in the module.

There are a number of examples in *Examples/PlatformConstruction* that show different modules being created and executed with *harness.exe*.

4 Cross Compiler Tools

To run code on a processor that is not the host PC processor (x86) you need to cross compile your software to the target embedded processor's instruction set. This requires a set of cross compiler tools (the tool chain).

We expect that for your project you will obtain one of the various tool chains that are available for your specific processor architecture. There are many commercial vendors and open source solutions available.

Imperas and OVP do not provide commercial cross compiler tool chains.

As a convenience, to help you get started, Imperas provides many of the GNU cross compiler tool chains in a ready-to-use, pre-compiled, executable format via the download sections of the OVP and Imperas User websites. These are configured and compiled up from open source repositories and are provided 'as is'. These provided tool chains include compilers, linkers, and GDB debuggers.

5 Debuggers

5.1 Debugging applications running under an operating system

When applications run under the control of an operating system and are dynamically loaded, for example as processes under Linux, to be able to debug them they need some form of agent to run as a process under the operating system. This embedded agent controls the dynamically loaded user application and allows it to be debugged. The embedded agent communicates with a debugger outside of the system.

For example, using a GDB debugger, the embedded agent, gdbserver, controls the running user application and communicates to the GDB debugger program running outside the system being debugged. If the application and gdbserver are running under an OS running on a hardware target, the GDB will run on the host PC talking to the hardware using some form of debug 'pod' or through a TCP socket. If the target is a simulator, then the simulator will provide a TCP socket for the GDB to connect to using RSP - providing the same use model and experience as if the target was a hardware platform.

Eclipse or other GUI based environments can wrap the debugger to provide a graphical source code debug experience to debug applications running under an operating system on a virtual platform simulation using Imperas simulators.

5.2 Debugging bare metal software.

Bare metal software, hardware dependent software, firmware, and operating systems are all software that run directly on the hardware and are not separate software programs and are normally part of the one executable binary running directly on the platform. The simulator has direct control of the processors in the platform running the software and allows the debuggers to connect with it directly to debug the running software.

The OVPsim simulator can connect one GDB to one processor in the design being simulated.

The Imperas Professional Simulators (CpuManager) can connect a GDB debugger to each processor in the design, or can connect MPD to the simulator to control all processors.

5.2.1 GNU GDB

The GNU GDB debugger can be connected to one of the processors when running OVPsim or multiple processors can be connected when running CpuManager. The GDB can be used within an Eclipse CDT or Imperas eGui environment, or can just be connected to a terminal console.

Other debuggers that support the GDB RSP protocol may also be used depending on their level of support for RSP.

5.2.2 Imperas Multi-Processor Debugger (MPD)

The Imperas Multi-Processor Debugger (MPD) debugger is one of the Imperas Professional Tools and is part of the Imperas M*SDK product.

MPD allows simultaneous debugging of all the application processors, in a platform, including single core, multi-core and multi-threaded variants.

Additionally, peripheral models can be debugged at the same time as the application, letting the developer see the peripherals operating in the context of the platform and the application code.

The command set of the debugger is fully compatible with the GDB open source toolset, but adds commands to handle multiple processors and cores, and adds functionality to view peripheral, TLB and cache registers and to be notified when they change.

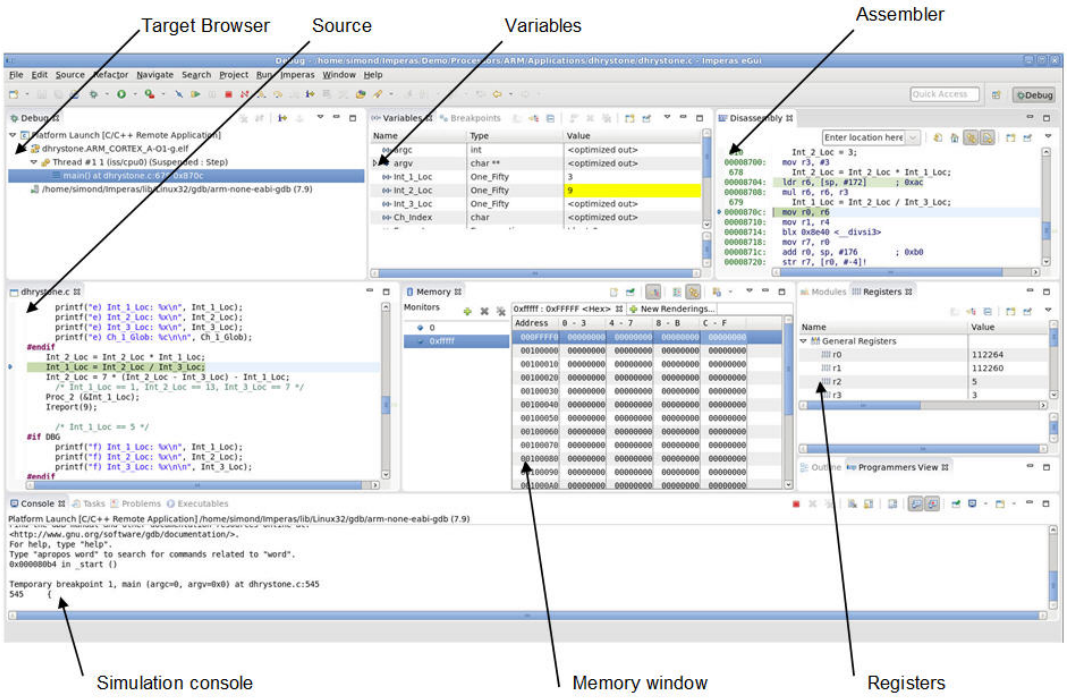
The MPD debugger is fully scriptable using the TCL language, and can be programmed to execute TCL routines in response to break- and watch-points. This allows intelligent debugging of platforms running VM systems, their drivers and their applications.

5.3 eGui

When using GDB or MPD you can run with the Imperas eGui, an Eclipse based GUI for controlling the simulation and debug of the embedded software.

eGui has the normal Eclipse CDT style perspectives and windows, plus several specific to Imperas and Multi-Processor debug.

eGui is provided as its own downloadable package. When you have installed it, try it out with the demos in Demo/Processors.


Target Browser
Source
Variables
Assembler
Simulation console
Memory window
Registers

6 Imperas iGen Wizard for Model Building (Productivity Tool)

The *iGen* productivity tool is used to build platforms, modules, intercept (extension) libraries, and behavioral/peripheral models by executing TCL (Tool Control Language) scripts and generating the C/C++/SystemC TLM2 files needed for the different tools.

Using TCL commands it is simple to create the C templates needed for model development and platform descriptions in C and C++/SystemC TLM2.0.

See the documents *iGen Model Generator Introduction*, *iGen Platform Generator User Guide*, and *iGen Peripheral Generator User Guide* for more information.

In OVP and Imperas installations there are over 50 Example directories with complete worked code, scripts, and detailed explanations in the User Guides. This makes it very easy to learn how to build and use the OVP technologies.

7 Imperas cpuGen Wizard for CPU Model Creation (Productivity Tool)

The *cpuGen* productivity tool eases the creation of processor models by executing TCL (Tool Control Language) scripts and generating a set of template files for a processor model.

Using TCL commands it is simple to create the C template files needed for processor model development. The templates files include all the recommended model structure and the implementation of the instruction decoder and other core aspects of the model; allowing the developer to concentrate on the task of describing the behavior required for each instruction.

See the document *Imperas CPU Model Generator Guide* for more information.

8 Imperas Verification, Analysis, Profiling (VAP) Tools for Software Development

Imperas simulators include advanced APIs to enable verification, analysis, and profiling to be accomplished completely un-intrusively. Native code can be written that uses these APIs and provided as a binary library plugin. The main capability is marketed under the SlipStreamer™ name and is based on a binary interception technology.

The Imperas installation includes a library of verification, analysis and profiling (VAP) tools that provide extensive visibility into the booting and running of the software on a virtual platform, in order to aid in identifying functional bugs, examining system activity and performance, and providing profiling and coverage information. These tools are fully described in the *Imperas VAP Tools User Guide* document.

In addition to the library of tools provided by Imperas, it is very easy for the user of the Imperas simulators to develop their own verification libraries as described in the *Imperas Binary Intercept Technology User Guide* document, which includes source examples that illustrate all of the basic techniques that are used to build verification plugins.

Example plugins provided as source include simple instruction, function, and memory tracing, programmers view monitoring, through to more complex examples including memory access checkers, share memory access checkers, malloc checkers etc. Also there are examples of cache and TLB monitors.

One of the usages of the Imperas verification technology is to improve the quality of software and the interfaces between models and software and hardware protocols. Functional coverage and assertions enable quality to be monitored and measured. The tools provide capabilities to develop software API and hardware protocol specific functional coverage and assertions.

All of the Imperas verification technology can be used with both processor and behavioral peripheral models and can work with bare metal or with running operating systems.

9 Imperas ipost Post Processor

The Imperas VAP Tools can produce a number of different data files, adhering to standard formats where possible. Where the files use industry standard formats they can be viewed using third party tools. In order to view some of the information contained within these data files without the use of a third party tool or where some post processing is required Imperas provides the *ipost.exe* program. This supports all of the formats generated by the VAP Tools and produces html output that may be viewed with any html browser program.

See the *Imperas Vap Tools User Guide* for more information on *ipost.exe*.

10 Imperas Library

The Imperas library is a collection of processor and peripheral models, Extendable Platform Kits™ (EPKs™), extension packages and analysis tools. All can be loaded by the Imperas simulators though there are some restrictions with OVPsim.

For information on all the components in the Imperas Library see the file:
$*IMPERAS_HOME*/doc/library/index.html
which is installed as part of the Imperas / OVP installation.

Alternatively you can look at the OVPworld website: http://www.ovpworld.org/library.

10.1 Processor Models

Within the library there are processor models of which many have been verified by the processor vendors themselves using their internal validation suites.

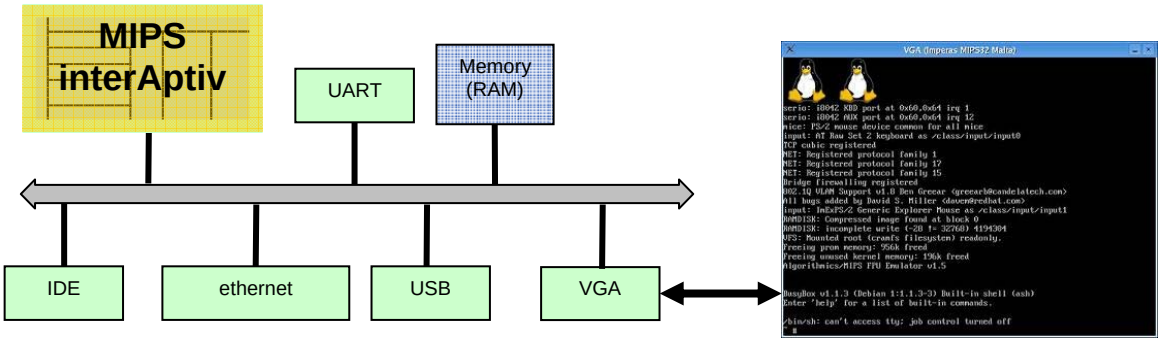
Currently there are models for ARM, MIPS, Xilinx, Renesas, Power Architecture, ARC, Altera and openCores families.

There are also processor models that are available from other model developers, though these are not maintained by Imperas.

10.2 Extendable Platform Kits

EPKs are virtual platforms (simulation models) of the target devices, including the processor model(s) plus peripheral models sufficient to boot an operating system or run bare metal applications. EPKs provide a base for you to extend and customize the functionality of the virtual platform, to closer reflect your own platform, by adding more component models, running different operating systems or adding additional applications. The platform and the peripheral models included in the EPKs are open source, so that you can easily add new models to the platform as well as modify the existing models.

The platforms available range from simple bare metal, processor-only platforms, through to multicore and to platforms that boot multicore operating systems such as the MIPS Malta platform that can run SMP Linux.



MIPS Malta EPK (Linux)

##