OVP Guide to Using Processor Models

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

Author:	Imperas Software Limited
Version:	0.6
Filename:	OVP_Guide_To_Using_Processor_Models.doc
Project:	OVP Guide to Using Processor Models
Last Saved:	Monday, 13 January 2020

Copyright Notice

Copyright © 2020 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

This document describes the Imperas OVP Fast Processor Models and how they are used. It gives an overview of using a processor model in different simulation environments. It refers to other documents needed for more detailed information.

Part of this document lists the different processor models available and their configuration options and any specific usage constraints and requirements.

1.1 Notation

`Code` Example code

1.2 Glossary

Attribute: The term *attribute* has been replaced by *parameter* in the context of a processor model. It is a name/value pair associated with a processor instance. The value can be a string, integer or floating point number.

Bare Metal: A platform with no operating system, often just a single processor with a fully populated memory space. Often the platform is used with a basic semihost library and referred to as a *user-mode* simulation.

Binary interception: The monitoring by the simulator of the execution of application code by a processor model so that the simulator can change its behavior without having to modify the application code. Binary interception is used for Semihosting, tracing and analysis.

CpuManager: Imperas Professional Product simulator that implements all of the OVP APIs.

Core: Autonomous execution unit, usually defined by having its own program counter (PC).

Cycle Accurate Model: A model that represents the implementation details of a processor including its pipelines, and cycle by cycle state changes. (as opposed to Instruction Accurate)

Debug target: Execution unit that is recognized by a debugger. Usually equivalent to a *Core*, but might not be.

DMI: Direct Memory Interface. (In TLM2.0) circumvention of the transaction mechanism, giving a processor model direct access to a memory model. When enabled this can speed up a simulation by orders of magnitude, but loses the ability to observe or analyze bus transactions.

Instance: Copy of the part of a model that holds its state. OVP simulators can load a model once, then simultaneously simulate several instances without interference between them.

Instruction Accurate Model: A model that represents the functionality of a processors instruction execution without regard to artifacts like pipelines. Only instruction boundaries are visible. (as opposed to Cycle Accurate).

MIPS: Million Instructions Per Second. A measure of processor speed (not to be confused with MIPS Technologies, Inc, a processor IP vendor).

Model: Software simulation model of a processor or processor family.

Multicore: A processor containing more than one core.

OVPsim: Simulator that implements a subset of the OVP APIs, available from the OVPworld website.

Platform: Software model of a complete circuit comprising processors, memory, buses and peripherals. The simulation is accurate enough for software development but not for accurately predicting system performance..

Processor: Indivisible device provided by a silicon vendor or licensor. Can be supplied as RTL, layout or finished silicon.

Quantum: (In multiprocessor simulation) A time period in which each processor in turn simulates a certain number of instructions. Simulated time is advanced only at the end of a quantum, so this is limit of timing accuracy of the simulation.
The quantum is usually fixed for the duration of a simulation, but can be changed (at the start of a new quantum).

Semihosting: Interception by the simulator of calls in the application to I/O functions and the passing of the calls to the host operating system.

SlipStreamer™: Imperas marketing name for binary interception.

Sparse memory: Simulated memory is created by the OVP simulator as it is used; unused regions are not allocated. Therefore the simulator can create a model of a memory larger than that of the host computer.

TLB: Translation Look-aside Buffer. Part of a processor's VM controller.

Variant: A configuration setting of a model to represent a specific vendor processor, for example to configure the generic MIPS model to be MIPS 24KEc.

Virtual Platform: as Platform.

VM: Virtual Memory or Virtual Memory controller. Hardware that allows a processor to simultaneously execute several programs without interfering with each other.

1.3 Recommended Reading

Imperas simulation technology is based on just-in-time (JIT) compiler technology. The following book provides a good introduction to the concepts involved:

Virtual Machines, by James E. Smith, Ravi Nair
ISBN 1-55860-910-5
Publisher: Morgan Kaufmann/Elsevier

2 References

Subject	API	Document
Processor modeling	VMI	OVP Processor Modeling Guide OVP VMI Morph Time Function Reference OVP VMI Run Time Function Reference
Platform construction Debugger integration	OP	iGen Platform and Module Creation User Guide 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
Peripheral modeling	PPM BHM	OVP Peripheral Modeling Guide OVP BHM PPM Function Reference
TLM2.0 integration		OVPsim using OVP Models in SystemC TLM2.0
GDB Integration		OVPsim Debugging Applications with GDB OVPsim Debugging Applications with Eclipse
Cache modeling	VMI	OVP VMI Memory Model Component Function Reference
Multicore Debugger		Imperas Debugger User Guide
Verification, Analysis and Profiling (VAP)		Imperas VAP Tools User Guide Imperas Control File User Guide
	VMI	Imperas Binary Intercept Technology User Guide

2.1 Open Virtual Platforms API Acronyms

VMI Virtual Machine Interface (for processors)
OP OVP Platforms (for simulation control and platforms)
BHM BeHavioral Modeling (for peripherals)
PPM Peripheral Programming Modeling (for peripherals)
VAP Verification Analysis Profiling

3 Introduction

The OVP simulation technology from Open Virtual Platforms (OVP) and Imperas Software Limited enables very high performance simulation, debug and analysis of virtual platforms containing multiple processor and peripheral models. The OVP technology is extensible, provides the ability to create new models of processors and other platform components by writing C/C++ code that uses application programming interfaces (APIs) and libraries supplied as part of OVP.

Processor models developed using this technology can be used with both the Imperas Professional Tools simulation products and the OVPsim simulator.

3.1 SystemC TLM2.0

If you are a SystemC user, you should use this document in conjunction with the document 'OVPsim using OVP Models in SystemC TLM2.0', and follow the SystemC references.

3.2 C platforms, OVPsim, and CpuManager

If you are not a SystemC user, you will construct your platform using the C API and should use this document in conjunction with the documents referenced above.

4 Processor Model Overview

4.1 What is a processor model?

An OVP processor model is a shared object file (.so on Linux, .dll on Windows) and an optional C++ header for use with SystemC. The shared object can be loaded by an OVP compliant simulator.

The OVP processor models use the VMI API to convert the target processor code into native code for efficient simulation.

The Imperas OVP Fast Processor Models are written in C and are compiled with GCC. Both the binaries and source trees are provided as part of standard OVP installations.

The models are used directly as shared objects.

4.1.1 How a processor model works

The OVP model uses the VMI API to:

a) build an instruction decoder to decode the instruction set of the target processor;
b) provide disassembly output;
c) generate native code for simulation, without explicit reference to the native instruction set.

The model is only concerned with code translation; the simulator implements the just-in-time translation algorithm, memory allocation, re-translation of modified code and other features required for simulation.
For details of the VMI API, see references in section 2.

It is important to note that the simulator is very separate from the model - the model is written in C/C++ and the simulator is required to load and execute that model.

4.2 What is modeled and what isn't?

4.2.1 Instruction Accurate

An OVP processor model is instruction accurate; the outcome of a simulated program executing in a single thread will correctly match the hardware being modeled.

4.2.2 Pipeline and closely coupled Cache

The processor pipeline is *not* modeled so bus accesses will not always occur in the correct sequence. For this reason, OVP users do not generally model and use caches (though OVP has all the capabilities to model both closely coupled L1, and shared L2, L3, … caches).

Some OVP processor models do have models of L1 caches.

Cache-control registers are present, so software that controls a cache will execute correctly.

4.2.3 Cache Ratio Monitor
Some OVP processor models have a cache ratio monitor (CRM) which models a cache in sufficient detail to determine the approximate ratio of hits to misses. A CRM cannot be used to check cache coherency.

4.2.4 Speed
An OVP model does *not* accurately model processor speed – you need cycle accurate and MHz clocking to do this. OVP models have 'mips' ratings to specify the number of instructions to be executed in a measure of elapsed real time. Simulated time can be used to compare software implementations, but will not give an accurate indication of speed in the real hardware.

4.2.5 Virtual Memory
The model can use the VMI API to model Virtual Memory (VM) as simple fixed mappings or as complex hardware such as used in the cache modeling reference of Translation Lookaside Buffer (TLB) hardware; including exceptions and co-processors required to make a comprehensive model of a modern virtual memory equipped processor.

4.2.6 Standalone & L2, L3, etc Cache models
Stand-alone cache models are available in OVPsim. They can model any cache algorithm, but are slow and cannot be controlled by registers built into a processor. To write cache models see the cache modeling reference of section 2.

4.2.7 Time
OVP simulators have a concept of time. Simulation time is advanced at the end of each simulation quantum (which is global across the simulator), so that by choosing the length of time simulated in each quantum and the number of instructions executed by each processor in a quantum (by the MIPS rating), each processor is given a number of instructions to execute.

To set the MIPS rating of a model and the quantum of the simulator refer to the API reference in section 2.

Peripheral models can cause events to happen after a specified delay, accurate to a microsecond. However, it should be noted that time does not normally advance during a quantum, so peripheral registers that return time-related data might not give accurate results.

4.2.8 JTAG
OVP does not implement JTAG or include JTAG functionality in its models.

4.3 How is a model used?

A model cannot be used directly; it must be loaded by a simulator, e.g. OVPsim. The simulator runs the model together with any other models, either automatically, or under control of the API function calls.

4.4 Can a platform have multiple processors?

Yes.

An OVP simulator can load many instances of identical or different processor models, using separate, partly or fully shared memory schemes.

4.5 How are multiprocessor systems simulated?

OVP processors in multiprocessor platforms do not execute simultaneously; for efficiency each processor advances a certain number of instructions in turn. Increasing the number of instructions run on one processor in one turn (quantum) will reduce the time spent by the simulator switching context, so will improve simulation speed. However, it will also increase the chance that interactions between processors will be inaccurate with respect to the timing, especially if they communicate through shared memory. This mechanism is used in C, and C++, and SystemC TLM.0 platforms.

4.5.1 How many instructions per quantum?

The default behavior of the simulator is to run 100,000 instructions on each processor in turn. Performance is affected by the application and platform, but increasing the number of instructions beyond 100,000 has no measurable effect. The quantum can be reduced for better accuracy but below 1000 performance will suffer. It is possible to change the quantum size during a simulation, but only at the end of a quantum.

4.6 Do OVP models work in transaction-based platforms?

OVP models work successfully in transaction based simulation; SystemC TLM2.0 is an example of this. OVP models have a SystemC TLM2.0 interface, which allows them to be directly instantiated in a SystemC TLM2.0 platform. See references to SystemC TLM2.0 platforms in section 2.

4.6.1 Real and Artifact Transactions

An OVP processor model uses dynamic code translation. One of the few side effects of this technology is that the first time a block of code is executed, the simulator reads the program as it is translated into native code, prior to simulating execution of the code in the normal way. Each program read generates a bus transaction that would not be present in real hardware. Transaction based simulators allow this type of read to be marked as a *simulation artifact* or *debug transaction* so it can be ignored by parts of the model which might be disturbed. A debug transaction must carry the correct data so that the instructions can be understood, but the transaction should not be included in performance or cache analysis for example.

4.7 How are different bus architectures represented?

OVP allows the modeling of complex addressing systems and can correctly represent shared memory, aliasing and missing regions. Bus transactions can represent accesses of different numbers of bytes so that complex peripherals behave correctly. However OVP simulators are not cycle-accurate so do not simulate the details of a system's underlying bus architecture. For example, changing a bus implementation from AMBA to PCI will not normally show any differences in system performance.

5 Instancing OVP processor models in a platform

Imperas OVP Fast Processor Models can be instanced in many types of platforms.

5.1 SystemC TLM2.0 platforms

Each Imperas OVP Fast Processor Model is shipped with a TLM2.0 interface. Model instances should be added to the platform code, the interface header files included in the TLM2.0 platform, and the platform recompiled including the OVP headers and library. The processor models will then be automatically loaded at run-time. For details of using OVP models in SystemC see references in section 2.

5.1.1 Memory and SystemC TLM2.0 DMI

OVP models implement the SystemC TLM2.0 Direct Memory Interface (DMI). This allows a processor to negotiate a direct connection to a memory, subsequently circumventing the TLM2.0 bus transaction mechanism. The effect is that only the very first TLM2.0 bus transaction will occur - subsequent accesses will be invisible to TLM2.0, giving a very significant simulation speed-up. This feature is turned on by default, but can be disabled. See the reference to TLM2.0 in section 2.

5.2 C platforms

Instancing an OVP Fast Processor Model in a C platform is covered in detail in the platform construction document referenced in section 2, “References”.

6 Processor instance configuration

OVP processor models have many features which are controlled from the simulation environment by *parameters.* Some parameter names are standard to all processor models; some are specific to a particular processor. A parameter is a name/value pair. The name describes what the parameter controls, the value can be a string, integer or floating point number. Each model instance has its own parameters which must be *set* by the simulated platform and are then *read* by the model. See the model specific information documents for lists of parameters defined by each processor model.

parameter name	meaning	type	values
variant	select the model variant	string	model-specific
endian	processor endian	string	big, little
mips	Speed (10^6 instructions per sec)	real	positive number

Table 1 Standard Processor Parameters

6.1 Variant

An OVP processor shared object usually includes model configurations of more than one core from the same family.

For example a single ARM model could be configured to be one of 20 ‘variants’, e.g. an ARM7TDMI, ARM926EJ-S or ARM Cortex-A9UP, etc.

The different configurations are referred to as *variants* and can be selected by setting the *variant* parameter on the model instance.

6.2 Endian

Processors that can be configured big or little endian read their *endian* parameter. Some processor models can automatically set their endian to that specified by the application program being loaded.

6.3 mips (clock frequency)

The parameter *mips* is used to set the instruction frequency of a particular processor instance. The specified parameter value sets the number of million instructions executed per second. OVP processor models have a default speed of 100 MIPS.

6.4 Optional Features

Processors are supplied by their vendors with optional features; e.g. DSP instruction sets, floating point unit or TLB, etc. The OVP models have similar options which are selected by the models parameters. See the Model Specific Information documents for lists of parameters defined by each processor model.

7 Connecting processors to platform components

When a processor has been instanced and configured, it must be connected to other components in the platform. Components have bus ports which are connected together with buses and net ports connected using nets.

Most processor models have the following ports:

port name	port type	function
INSTRUCTION	bus master	fetch instructions for execution
DATA	bus master	read and write data
reset	net input	processor reset
intXXX	net input	hardware interrupt(s)

7.1 Bus Connections

Most processor models have separate instruction and data bus connections. If the silicon implementation has a single bus connection, instruction and data can both be connected to the same bus. Note that multiplexing connections by using multiple ports to common buses and nets has no effect on behavior or simulation performance.

7.2 Interrupts

A processor's external interrupt inputs modeled as nets in OVP. They are generally edge triggered and must be written with their inactive value followed by their active value, to produce and interrupt. Refer to vendor's data to determine if an input is level or edge sensitive and if the sense is positive or negative.

7.2.1 Vectored interrupts

Unlike in real hardware, an OVP net can carry an integer value. Some processor models use this value to encode the interrupt priority or level.

7.3 Processor Reset

Most OVP processor models include a reset pin. Refer to vendor's data to determine the sense of the reset input.

7.3.1 Implicit reset

A processor model performs an implicit reset at time zero, so there is no need to connect or drive the reset pin. However, if required, the processor can be forced into reset by writing a '1' to the reset, and held in reset until a '0' is written.

7.3.2 Reset will override the program entry point

After an explicit reset the processor will start code execution from the reset vector and will not use the program entry point provided in the application program.

7.3.3 Instruction accounting

When in the reset state the processor is idle. Time still moves forward and as such the instruction counts increase within each simulation quantum.

8 Processor Debug

It is necessary to be able to use industry-standard debug tools to debug application code that is running on the processors modeled in a virtual platform. OVP processor models can interface to a debugger in two ways: by RSP socket or through an API.

8.1 RSP Integration

OVPsim and the Imperas CpuManager have a built-in debug interface which uses the GNU GDB Remote Serial Protocol (RSP).

Note that this is in the simulator and not the processor model, meaning that all OVP processor models can be debugged using GDB (assuming that a GDB is available for that specific processor).

Depending on the capabilities of the specific simulator, a debugger or debuggers can be connected to one or more of the processor model instances present in a platform. The Imperas professional tools include a multicore debugger which uses an enhanced version of RSP to allow simultaneous debug of all the cores in a platform. For details see the references in section 2.

Supported connection	Simulator	
	OVPsim	**CpuManager**
Single GDB	yes	yes
Multiple GDBs	no	yes
Multicore debugger	no	yes

8.1.1 Use model

An executable is required, containing the platform definition (using the C API) and the OVP simulator. This executable can be used for debug or non-debug execution; the debug interface is enabled by a call to the C API or by using a control file. After starting, the executable waits for a socket connection. A debugger (in a separate process) connects to this socket. The debugger then communicates with and controls the simulator using the bidirectional socket connection.

8.1.1.1 Port number selection

Each initial debugger connection is made via a single port number (the connection is then handed off to a port allocated by the host operating system). The initial port number can be set, or if the value is zero, it will be chosen by the operating system.

8.1.1.2 Selecting the debug target

Single processor debug (OVPsim)	Single target processor must be specified in platform
Multiple GDB connections (CpuManager)	Processors connected to GDBs in order of processor construction
Multiprocessor Debug (Imperas Professional Tools)	All processors can be debugged simultaneously.

8.1.2 Enabling the debugger port

8.1.2.1 Using OVPsim and C API

Modify the call to opRootModuleNew:

To open a terminal and connect to the GDB debugger

```
mi = opRootModuleNew(0, 0, OP_PARAMS(OP_PARAM_BOOL_SET(OP_FP_GDBCONSOLE, 1)));
```

To open a port to which a GDB debugger can be connected. Using 0 allows the simulator to open the next available port or a specific port number can be provided.

```
mi = opRootModuleNew(0, 0, OP_PARAMS(OP_PARAM_UNS32_SET(OP_FP_REMOTEDEBUGPORT,
0)));
```

8.1.2.2 Using CpuManager and a control file

Add this to the control file:

```
-gdbconsole
```

or

```
-port <port number>
```

8.1.2.3 Using the Imperas Simulator's Command line

```
-gdbconsole
```

or

```
-port <port number>
```

8.2 API integration

Users wishing to integrate a third-party debugger or have their tools/programs control the models and/or simulation must use the C API. This allows complete control of the processors in a platform and provides all the required features including:

- interrogation of processors and their configuration
- run
- stop (interrupt simulation)
- single-step

- disassembly
- add/delete breakpoints
- add/delete watchpoints
- read/write registers
- read/write memory
- request callback on change of simulator state

8.2.1 Use model

The C API can be used both to construct the platform and separately, to debug it. For instance, a platform definition might be supplied as a shared object which is selected and loaded at run-time into a system which includes OVPsim and a debugger.

8.2.2 Using OVPsim in a threaded program

OVP models can be run in 'quick-threads' (SystemC uses this mechanism) or in a single POSIX thread or process. The C API simulation functions *opProcessorSimulate* and *opRootModuleSimulate* are not designed to be thread safe: do not make multiple calls to these functions from asynchronous POSIX threads.

It is possible to run the simulator in a separate thread from the debugger, provided certain conditions are met. This allows a certain level of responsiveness from the debugger during simulation. Refer to section 2 for details of debugger integration using the C API interface.

8.3 Debugging multicore processors

A platform using one single core processor requires one debugger. If a platform uses a multicore processor, each core will appear as a debug target (note that some silicon vendors allow the user to choose the number of cores in a processor). The C API provides functions to traverse the processor hierarchy, identifying each core and allows selecting one or more as debug targets.

9 Tracing, Interception, Semihosting, and Analysis

The same software program compiled to a binary that would be used in the silicon may be used in the OVP simulation. Symbolic and debug information must be available to allow interception, semihosting and analysis tools to work correctly.

9.1 Instruction Tracing

To help the user debug an application the OVP simulator can produce a trace of each instruction as it is executed. This is a feature of the simulator so does not have to be included in each processor model. Tracing can be turned on or off per processor model instance at any time before or during simulation. Tracing information includes address being executed, disassembly of the instruction and (in CpuManager) reference to code labels found close to addresses reference by the instructions.

Note that tracing is produced as each model is executed, so the effects of the scheduling algorithm will be seen in the trace.

9.2 Binary interception and Semihosting

Application code can be executed on a Bare Metal platform. In this situation it is useful for the application to read input data from files and write output data to files or a terminal on the host system. The application can, for example, be linked with a standard LibC and use the input and output functions available in LibC. The simulator then intercepts calls to low-level I/O functions in LibC, passing the requests to the host operating system. This is referred to as *semihosting*. All Imperas OVP Fast Processor Models have semihost libraries to implement a useful subset of the C language library. Since the C++ compiler uses the C library for many of its functions, the basic C++ I/O library can also be used.

9.3 Verification, Analysis and Profiling

CpuManager can use additional tools that help analyze the application code as it executes. See the reference to V.A.P. in section 2.

9.4 Summary of features

Feature	**OVPsim**	**CpuManager**
Address tracing	y	y
Disassembly	y	y
Code label tracing	n	y
Semihosting	y (one library only)	y (multiple libraries)
VAP tools	n	y

10 Model commands

Commands are implemented by a particular model and have no standard names or formats. Please refer to the model specific information documents.

Commands are used by debug and analysis tools to access the internal state of the model without using the hardware interfaces, in the same way that a hardware debugger might use JTAG. A command can send text to the simulator standard output stream or return a value to the API function.

Commands can be issued from a suitably equipped debugger, from a control file (in CpuManager), from a simulator command-line (in the Imperas Simulator), or by using the C API (all simulators).

11 Available Imperas OVP Fast Processor Models

The current OVP processor library as included in the OVPworld distributions contains:

- arc
- arm (classic, Cortex-A, and Cortex-R profiles)
- armm (Cortex-M Profile)
- mips32
- mips64
- v850
- or1k
- powerpc32
- m16c
- r8c
- microblaze
- Nios II

Processor models are available from other sources:

- ms1750a
- sparc
- vinchip

12 Model Specific Documentation

Please read the companion documents that describe the features, parameters etc of the specific models and model variants.

#