OVP OP Platform and Testbench Creation and Usage Guide

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

Author:	Imperas Software Limited
Version:	0.2
Status:	Draft
Filename:	OVP_OP_Platform_and_Testbench_Creation_and_Usage_Guide.doc
Project:	OP API Usage
Last Saved:	Tuesday, 25 August 2015

Keywords:	OP API

Copyright Notice

Right to Copy Documentation

Destination Control Statement

Disclaimer

Table of Contents

1 Introduction

This document describes the modeling of modules and platforms for use with the OVPsim and CpuManager simulators, using the OP *(OVP Platform*) API.

1.1 What are CpuManager and OVPsim?

CpuManager and OVPsim are dynamic linked libraries (`.so` suffix on Linux, `.dll` suffix on Windows) implementing Imperas simulation technology. The shared objects contain implementations of the OP interface functions described later in this document. The OP functions enable instantiation, interconnection and simulation of complex multiprocessor platforms using multicore processors, advanced peripheral devices and complex memory topologies.

Processor models for use with CpuManager and OVPsim are created using another API, the *OVP Virtual Machine Interface* (VMI) API, also available for download from the www.ovpworld.org website. This API enables processor models to be created that run at very high simulation speeds (typically hundreds of millions of simulated instructions per second). This is described in the *OVP Processor Modeling Guide*, also available for download from the www.ovpworld.org website.

CpuManager is the commercial product available from Imperas. OVPsim is the freely-available (for Non-Commercial usage) version of this product. Which one to use is determined at runtime by the IMPERAS_RUNTIME environment variable. If it is not set or is set to OVPsim then the OVPsim library (which requires an OVP license) is dynamically linked at runtime. If it is set to CpuManager then the CpuManager library (which requires an Imperas license) will be used.

The legacy ICM API is also supported by the same products, providing a subset of the functionality offered by OP. In fact, the ICM API is implemented using OP so will be supported for the foreseeable future.

A subset of OP functionality can be used in SystemC TLM2.0. The TLM2.0 C++ interface code is available as source for processor and peripheral models, allowing the use of these models in SystemC TLM2.0 platforms.

1.2 Use of OP with Imperas tools

- A program using the OP or ICM APIs must be linked with the Imperas RuntimeLoader to perform runtime dynamic loading of either the CpuManager or OVPsim dynamic linked libraries, to produce a stand-alone executable.

1.3 Compiling Examples Described in this Document

This documentation is supported by C code samples in an `Examples` directory, available either to download from the www.ovpworld.org website or as part of an Imperas installation.

GCC Compiler Versions

Linux32	4.5.2	i686-nptl-linux-gnu (Crosstool-ng)
Linux64	4.4.3	x86_64-unknown-linux-gnu (Crosstool-ng)
Windows32	4.4.7	mingw-w32-bin_i686-mingw
Windows64	4.4.7	mingw-w64-bin_i686-mingw

The examples use processor models and tool chains, available to download from the www.ovpworld.org website or as part of an Imperas installation.

SystemC TLM2.0 models can be used on Linux with gcc or on Windows with MinGW/MSys (since SystemC release v2.3.0) or MSVC 8.0. It is assumed that users of this environment will be familiar with SystemC, TLM2.0 and will have obtained this software from www.systemc.org or similar.

1.4 C, TCL and iGen

OVP platform models are written from C code, compiled on the host computer, linked with the libRuntimeLoader run-time library to produce a shared object or executable.

iGen is a program from Imperas that can create either outline or substantially complete C code for a module using a description of the model written in the TCL programming language. Use of iGen and the TCL language is described elsewhere, but used in the examples in this document.

1.5 Shared Objects and executables

The shared objects referred to in this document are either Linux shared objects, with suffix *.so* or Windows dynamic link libraries with suffix *.dll*.

The executables referred to in this document are either Linux or Windows programs and have the suffix *.exe*

The Makefiles referred to in this document are written for GNU make. Standard Makefiles supplied by Imperas support compilation and linking using GNU tools on both Windows and Linux.

1.6 Example scripts

Example scripts will be referred to as (for example) *example.sh*, this being the extension used on Linux. On Windows the script would be called *example.bat*

2 Overview

2.1 Platforms and Modules

A *module* is a model that creates and connects instances of processors, peripherals, RAMs, ROMs, caches (known as MMCs), and other modules. It is linked with the libRuntimeLoader library to produce a *shared object*.

A *platform* is special type of module that creates the top level of the system. It is linked with the libRuntimeLoader library to produce an *executable*.

2.2 Entry points

The entry point to an intermediate module is the symbol *modelAttrs* which refers to a predefined table of functions and constants in the shared object.

The entry point to an executable top level module is the function *main.*

2.3 Building using the Imperas makefile

A template module produced by iGen has both entry points; *main* and *modelAttrs*.
The standard Makefiles supplied by Imperas produce both a shared object and an executable for every module. A module must be used in the correct context according to its internal design.

2.4 An intermediate module

An intermediate module must contain the `modelAttrs` table which has

- a code to identify the shared object as a module (and not a processor, for instance)
- the version of the API
- a default name of the module
- pointers to functions on the module
- classification and status of the module to be interrogated by other tools.

The `modelAttrs` table is defined in `ImpPublic/include/host/op/optTypes.h`

A *main* function in an intermediate module will be ignored.

2.4.1 Module functions

An intermediate module defines some or all of the following functions:

2.4.1.1 Interface Iterators

Functions to return in sequence each interface object(bus, net, packetnet, FIFO).

Parameter Iterator

A function to return the name, type and description of each parameter accepted by the module.

2.4.1.2 Constructor

(This function must be provided) A function to construct the contents of the module including creation of component instances, buses, nets etc. and connection to components.

2.4.1.3 Initialization
Called when simulation is about to begin.
2.4.1.4 Reporting
Called when simulation has finished, but before any destructors are called.
2.4.1.5 Destructor
If required, code to close files, free memory etc.

2.5 A typical top level module

The entry point of a top level module is (like any C program) called `main`. It is called by the operating system after setting up the heap and stack.

2.5.1 Command Line parser

`main` will usually include a command line parser to read the user's options for this simulation. The OP standard parser gives a consistent method of parsing standard data types, but also allows the user to specify any of the rich set of options accepted by the simulator. Please refer to the OVP Control File User Guide.

2.5.2 The root Module

`main` must create a root module. This module could then either:

- contain all the components in the system (this is how a legacy ICM) platform is constructed).
- create an instance of the top level of the design and supply any test–bench functionality required to run the design. In fact it could create instances of several designs and run then independently to perform *step and compare* testing.
- Reference the `modelAttrs` of this module to make this the top level of the design. Thus, if the compiled executable is used, main is called and this becomes the top level, or if the shared object is loaded, this becomes part of a larger system simulation.

2.6 Module Parameterization

To increase the flexibility and reusability of a module, it can accept parameters which may be set by the module that creates the instance of this module (its parent) and whose values can be obtained by code in the module, to influence its behavior. Parameter values can be passed to its components (including sub-modules) or can influence the execution of its code. Parameter types include Boolean, Integer, Floating point and String.

2.7 The module interface

A module connects to its model instances using the following interface abstractions. Interface objects can be connected to components within the module or, via module ports, to model instances in other modules.

2.7.1 Bus

A bus is a high level model of a microprocessor bus system. It represents a distinct physical address space. It allows *bus masters* such as processors or DMA engines to read or write to *bus slaves* such are memories or memory-mapped registers. A bus cannot

model contention, has no facility to model the time taken for each access, and consequently has no means to model burst modes, bus locking or priority schemes.

2.7.2 Net

A net is used to model a wire carrying a digital value, usually zero or one. In fact a net carries a 32-bit value so can carry more information but is usually used to model resets, interrupts, and mode controls. A net can have multiple drivers and receivers but does not model contention – the current value is that set by the most recent driver. All receivers are notified of a new value by a callback function in the model.

2.7.3 Packetnet

A packetnet is used to model packet-based protocols such as RS232, USB, Ethernet or GSM. A packetnet can have multiple drivers and receivers but does not model contention. A packet sent by a model is received instantaneously (time does not advance during its propagation) by all connected models in the order they were connected. Receiving models can modify the packet which can be examined by the sending model at the end of the transaction.

2.7.4 FIFO

A FIFO is a unidirectional point to point connection between two processor models. Words of a specified width (in bits) are pushed into one end of a FIFO and popped out of the other. A FIFO has a specified depth which cannot be exceeded. Primitives allow blocking or non-blocking push and pop operations.

2.8 Efficiency

Modules can be assembled with arbitrary depth of hierarchy, allowing the simulation of complex systems. However, the depth of hierarchy has no effect on simulation efficiency; module ports are removed before simulation.

3 Imperas OP Header Files

The OP API, used by CpuManager and OVPsim, is defined by header files in the Imperas tools release tree at $IMPERAS_HOME/ImpPublic/include/host/op:

file	**contents**
op.h	Includes all the other files
opcConstruct.h	functions for platform construction
opfFormals.h	Define the names of simulator system parameters
opmMessage.h	Formatted output to the simulator log system
oppCommandParser.h	OP standard command line parser
opqQuery.h	Function to interrogate an existing design
oprRuntime.h	Functions used when the simulator is running
optTypes.h	OP data types
opvVersion.h	The API version

In general, the prefixes of functions and type names match those of their containing file; OP data types begin `opt`, constructor functions begin `opc`.

4 Order of platform construction

This section summarizes the operation of a hierarchical platform.

- Host computer calls the program entry point : `main`
  - start `opcInit`
  - construct the command line parser `oppCmdParserNew, oppCmdParserAdd`
  - parse the command line `oppCmdParseArgs`
  - create instance of root module `opcRootModuleNew`
    - call the constructor `moduleConstruct`
      - create instance of the design `opcModuleNew`
        - call parameter iterator
        - call interface iterators
        - call module constructor `moduleConstruct`
          - create model instances `opcProcessorNew`
          - create module instances `opcModuleNew`
  - run the simulator `oprModuleSimulate` (maybe more than once)
    - call pre-simulate functions in all modules (first time only)
    - run the simulator
  - finish `opcTerminate`
    - call post-simulation functions for all modules
    - call destructors

Higher-level modules are constructed before lower modules.
Leaf components (processors, memories etc) can be created at any level.
A module can instance itself (so long as there is code to prevent infinite recursion).
The simulator can determine the interface to a module without constructing it.

5 Example platform

The example at $IMPERAS_HOME/Examples/Platforms/OP/simple has a test-bench *platform* and a module called m*odule* that contains the components; an OR1K processor, bus and memory.

The product should first be installed then the *simple* directory (and everything below) should be copied to a new directory.

In the new directory, compile the example application, the test bench and design then run the complete platform, by typing

```
>example.sh
```

Examine the platform and modules directories. iGen was used to convert the TCL description of the test-bench and module to C.

5.1 Module files created by iGen

The `module` directory contains the intermediate module. iGen splits the generated code (as far as possible) into files that can be modified by the user and those that do not need modification. This allows the user to modify the source TCL and rerun iGen without overwriting their modifications.

5.1.1 User modifiable file

module.c

```
/*
 * Copyright (c) 2005-2015 Imperas Software Ltd., www.imperas.com
 */

// This file declares functions to be implemented by the user.
// Be careful to avoid overwriting any edits should igen be re-run.


#include <string.h>
#include <stdlib.h>

#include "op/op.h"

#define MODULE_NAME "testfifo"

#include "platform.options.igen.h"
#include "platform.constructor.igen.h"

typedef struct optModuleObjectS {
    // insert module persistent data here
} optModuleObject;

////////////////////////////////////////////////////////////////////////////////
//                          U S E R   F U N C T I O N S
////////////////////////////////////////////////////////////////////////////////

static OPT_PRE_SIMULATE_FN(modulePreSimulate) {
// insert modulePreSimulate code here
}
```

```
static OPT_SIMULATE_FN(moduleSimulate) {
// insert moduleSimulate code here
}

static OPT_POST_SIMULATE_FN(modulePostSimulate) {
// insert modulePostSimulate code here
}

static OPT_DESTRUCT_FN(moduleDestruct) {
// insert moduleDestruct code here
}

#include "platform.attr.igen.h"

int main(int argc, const char *argv[]) {

    opcInit(OP_VERSION);
    optModuleP top = opcRootModuleNew(&modelAttrs, MODULE_NAME, 0);
    oprModuleSimulate(top);
    opcTerminate();
    return 0;
}
```

`optModuleObject` is a structure allocated for each instance of the module. Its pointer is passed to every callback. Use this to store instance-specific information that is not shared between instances (in the unlikely event that data should be shared between instances of a module type, declare a global static structure).

Pre-simulate, simulate and post-simulate functions can be modified if required (less frequently used functions need to be written and then referenced in the `modelAttrs` table).

The declaration of the `modelAttrs` structure is included in a separate file.

In this example, the module has a function `main` that will never be used. In fact in the shared object created by linking this code, `main` is unreachable.

5.1.2 Files not modified by the user

platform.attr.igen.h

```
optModuleAttr modelAttrs = {
    .versionString      = OP_VERSION,
    .type               = OPT_MODULE,
    .name               = MODULE_NAME,
    .objectSize         = sizeof(optModuleObject),
    .constructCB        = moduleConstruct,
    .preSimulateCB      = modulePreSimulate,
    .simulateCB         = moduleSimulate,
    .postSimulateCB     = modulePostSimulate,
    .destructCB         = moduleDestruct,
    .busPortSpecsCB     = moduleBusPortIterator,
};
```

This structure, the definition of `modelAttrs` makes the entry point to the shared object. `OP_VERSION` and `OP_TYPE` must come from the include file – they allow the simulator to verify the model against the API version, and to confirm that the correct kind of model is being loaded.

The Iterator constructor functions are defined in `module.igen.constructor`. The other callbacks are user modified functions mentioned earlier.

platform.constructor.igen.h

```
static OPT_CONSTRUCT_FN(moduleConstruct) {

    optBusP mainBus_b = opcBusNew(mi, "mainBus", 32, "mainBusPort", 0);

    const char *cpu1_path = opcVLNVString(
        0, // use the default VLNV path
        0,
        0,
        "or1k",
        0,
        OPT_PROCESSOR,
        1   // report errors
    );

    optParamP cpu1_param = 0;

    cpu1_param = opcParamDoubleSet(cpu1_param, "mips", 100.000000);
    optProcessorP cpu1_c = opcProcessorNew(mi, cpu1_path, "cpu1", cpu1_param);

    opcProcessorBusConnect(cpu1_c, mainBus_b, "INSTRUCTION");
    opcProcessorBusConnect(cpu1_c, mainBus_b, "DATA");

    const char *or1kNewlib_0_expath = opcVLNVString(
        0, // use the default VLNV path
        0,
        0,
        "or1kNewlib",
        0,
        OPT_EXTENSION,
        1   // report errors
    );

    opcProcessorExtensionNew(cpu1_c, or1kNewlib_0_expath, "or1kNewlib_0", 0);

    optMemoryP ram1_m = opcMemoryNew(mi, "ram1", OPT_PRIV_RWX, 0xffffffff, 0);

    opcMemoryBusConnect(ram1_m, mainBus_b, "sp1", 0x0, 0xffffffff);
}

static optBusPortInfo busPortSpecs[] = {
    { .name  = "mainBusPort" },
    { 0 }
};

static OPT_BUS_PORT_FN(moduleBusPortIterator) {
    prev = prev ? prev + 1 : busPortSpecs;
    return prev->name ? prev : 0;
}
```

This file should not need to be edited. The Iterator functions are called by the simulator immediately after loading the model. They return port and parameter descriptors to the simulator so it can determine the module's interface before the constructor is called. The simulator is then able to compare the interface with the connections made from the instancing module above. If everything matches, the constructor function is called. This uses `opc` construction functions to create instances of models (in this case a processor and memory), instances of interconnection objects (in this case a bus), then connects the components to the interconnection objects.

When a model instance comes from the library, the function `opcVLNVString` converts the Vendor, Library, Name and Version strings to the Linux (or Windows) path to the model's shared object (this function reports an error if the model is missing).

If this example (and in any module generated by iGen) the iterator functions iterate over static lists of objects. A hand-written model can iterate over a dynamically produced list (depending perhaps, on model parameters).

5.1.3 Files produced by the standard Makefile

All .c files are compiled by the Makefile (.h files are compiled only if included by .c). The intermediate files are linked first to product an executable and again to produce a shared objected. In this module (an intermediate), the shared object `module.so` is used; the executable `module.<architecture>.exe` is unused.

5.2 Platform files created by iGen

The `platform` directory contains the top level design.

5.2.1 User modifiable file

platform.c

```
#include <string.h>
#include <stdlib.h>

#include "op/op.h"

// Change the name here if required.

#define MODULE_NAME "testbench"

#include "platform.options.igen.h"
#include "platform.constructor.igen.h"
#include "platform.clp.igen.h"

typedef struct optModuleObjectS {
    // insert module persistent data here
} optModuleObject;

static OPT_PRE_SIMULATE_FN(modulePreSimulate) {
// insert modulePreSimulate code here
}

static OPT_SIMULATE_FN(moduleSimulate) {
// insert moduleSimulate code here
}

static OPT_POST_SIMULATE_FN(modulePostSimulate) {
// insert modulePostSimulate code here
}

static OPT_DESTRUCT_FN(moduleDestruct) {
// insert moduleDestruct code here
}

#include "platform.attr.igen.h"

int main(int argc, const char *argv[]) {

    opcInit(OP_VERSION);
```

```
    optCmdParserP parser = oppCmdParserNew(MODULE_NAME, OPT_AC_ALL);
    cmdParser(parser);

    // insert modifications to the CLP here

    oppCmdParseArgs(parser, argc, argv);

    // insert overrides here

    optModuleP top = opcRootModuleNew(&modelAttrs, MODULE_NAME, 0);
    oprModuleSimulate(top);
    opcTerminate();
    return 0;
}
```

When building a top level module, iGen produces the same files as for the intermediate module (it doesn't know how the module will be used). The pre-simulation, simulation and post-simulation and destructor callbacks are less likely to be modified by the user.

In the top level module, `main` will be used. After initializing the simulator (`opcInit`) main constructs the simulator's standard command line parser. The ability to parse user-defined arguments is added at this stage and C variables are created to hold the outcome of parsing the command line and/or control file.

The command line is parsed, causing user defined variables to be updated and simulator controls to be stored for reference. These allow control of tracing, VAP tools and allow user programs to be loaded into simulated memories.

A root module is created (using `opcRootModuleNew`), passing the `modelAttrs` table. This triggers the elaboration of the entire design and its hierarchy (elaboration is described elsewhere in this document).

`main` calls `oprModuleSimulate` to start simulation.
The simulator runs until it detects a condition to cause it to stop. In this case the stop is caused by the intercept library intercepting the execution of the `exit` function (called by the C runtime system when `main` returns).

[At this point the return value of `oprModuleSimulate` could be examined to determine the stop reason, in case the test bench wishes to continue simulation, though in the example this is not the case].

`opcTerminate` calls post-simulation callbacks, calls destructors, closes down the simulator, frees licenses and reports statistics.

5.2.2 Files not modified by the user

platform.attr.igen.h
(This is the same as for the intermediate module)

platform.constructor.igen.h

```
static OPT_CONSTRUCT_FN(moduleConstruct) {

    const char *mod0_path = "module/model";
```

```
    optParamP mod0_param = 0;

    optConnections mod0_conn = {
    };
    opcModuleNew(
        mi,         // parent module
        mod0_path,  // modelfile
        "mod0",     // name
        &mod0_conn, // conns
        mod0_param  // parameters
    );
}
```

The constructor callback creates an instance of the intermediate module. In this case no parameters are passed and no connections are made to the module's interface. The module is stored in the `module` directory, not the library, so its path is supplied directly, rather than using `opcVLNVString`.

platform.clp.igen.h

```
static void cmdParser(optCmdParserP parser) {
    oppCmdParserAdd(
        parser,                 // parser instance handle
        "userarg",              // name of the user defined argument
        0,
        0,
        "usergroup",            // group of the user defined argument
        OPT_FT_UNS32VAL,        // type
        &options.userarg,       // address of the variable updated by the argument
        "Illustrating the use of user define command line arguments",
         0x0,
         0,
         0
    );
}
```

The user defined argument is added to the parser before the parser is used.

platform.options.igen.h

```
struct optionsS {
    Uns32  userarg;
} options = {};
```

A variable is declared for each user-defined argument. The structure is global (there can only be one instance of the root module) so can be used anywhere in the code.

5.2.3 Files produced by the standard Makefile

In this module (the top level), the shared object `module.so` is unused; the executable `module.<architecture>.exe` is used to simulate this design.

6 Platform query

The opq section of the API allows a module to interrogate other modules in the design. A module can always be interrogated by the simulator, but can prevent interrogation by other modules.

7 Compatibility with ICM

The OP API replaces the ICM API used in earlier Imperas products. A new version of the ICM provides ongoing support for existing designs, implemented internally using OP functions. Therefore all necessary ICM functions will be supported for as long as they are required.
There is no need to convert an ICM platform to an OP module unless:

- The platform must be modified to use sub-modules
- The platform must be modified to be included in an OP design.

7.1 Interoperability

A design must use either OP or ICM. The two styles should not be mixed. OP modules can load other OP modules, but not ICM. ICM cannot load OP modules.

The processor, peripheral and intercept APIs have not changed. All leaf level models and plug-ins can be used with OP or ICM.

The GDB RSP interface, the multiprocessor debugger and other Imperas tools support platforms that use ICM or OP.

OP introduces a new way to construct a platform. Once constructed, the execution and scheduling of processor, peripheral and other modules is unchanged from previous versions of the simulator.

7.2 API tracing

Set either of the environment variables `IMPERAS_ICM_TRACE=1` or `IMPERAS_OP_TRACE=1` to turn on tracing of entry to and exit from ICM and OP functions. The two variables are interchangeable. Output is to the standard output of the console or shell that invokes the simulator.

7.3 The legacy simulator

The ICM-only simulator is still available in case the new simulator is suspected of behaving differently. Set the environment variable `IMPERAS_LOAD_ICM=1` to use the old simulator. The old simulator will be supplied until any discrepancies have been resolved.

7.4 ICM to OP conversion

This table shows OP functions replacing ICM functions.

ICM	OP
icmAbortRead	oprProcessorReadAbort
icmAbortWrite	oprProcessorWriteAbort
icmAddBoolAttr	opcParamBoolSet
icmAddBusFetchCallback	oprBusFetchCallback
icmAddBusReadCallback	oprBusReadCallback

ICM	OP
icmAddBusWriteCallback	oprBusWriteCallback
icmAddControlFile	deleted
icmAddDoubleAttr	opcParamDoubleSet
icmAddFetchCallback	oprProcessorFetchCallback
icmAddInterceptObject	opcProcessorExtensionNew
icmAddNetCallback	oprNetCallbackAdd
icmAddPacketnetCallback	opcPacketnetCallbackAdd
icmAddPortMapCB	oprBusPortConnMapNotify
icmAddPseInterceptObject	opcPeripheralExtensionNew
icmAddPtrAttr	opcParamPtrSet
icmAddReadCallback	oprProcessorReadCallback
icmAddStringAttr	opcParamStringSet
icmAddSymbol	oprProcessorApplicationSymbolAdd
icmAddUns32Attr	opcParamUns32Set
icmAddUns64Attr	opcParamUns64Set
icmAddWriteCallback	oprProcessorWriteCallback
icmAdvanceTime	oprModuleTimeAdvance
icmAdvanceTimeDouble	oprModuleTimeAdvance
icmAllVlnvFiles	opcVLNVIter
icmAtExit	opmAtExit
icmBanner	opmBanner
icmBridgeBuses	oprDynamicBridge
icmCLPDefaultObjectFile	oppDefaultApplication
icmCLParseArgUsed	oppCmdArgUsed
icmCLParseArgs	oppCmdParseArgs
icmCLParseFile	oppCmdParseFile
icmCLParseStd	oppCmdParseStd
icmCLParser	oppCmdParserNew
icmCLParserAdd	oppCmdParserAdd
icmCLParserOld	oppCmdParserOld
icmCLParserUsageMessage	oppCmdUsageMessage
icmCallCommand	oprCommandStringCall
icmCancelTrigger	oprModuleTriggerDelete
icmClearAddressBreakpoint	oprProcessorBreakpointAddrClear
icmClearICountBreakpoint	oprProcessorBreakpointICountClear
icmConnectMMCBus	opcMMCBusConnect
icmConnectMemoryToBus	opcMemoryBusConnect
icmConnectPSEBus	opcPeripheralBusConnectSlave
icmConnectPSEBusDynamic	opcPeripheralBusConnectSlaveDynamic
icmConnectPSENet	opcPeripheralNetConnect
icmConnectPSEPacketnet	opcPeripheralPacketnetConnect
icmConnectProcessorBusByName	opcProcessorBusConnect

ICM	OP
icmConnectProcessorBusses	opcProcessorBusConnect
icmConnectProcessorConn	opcProcessorFIFOConnect
icmConnectProcessorNet	opcProcessorNetConnect
icmDebugReadProcessorMemory	oprProcessorRead
icmDebugThisProcessor	opcProcessorDebug
icmDebugWriteProcessorMemory	oprProcessorWrite
icmDeleteWatchPoint	oprWatchpointDelete
icmDisableTraceBuffer	oprProcessorTraceOffAfter
icmDisassemble	oprProcessorDisassemble
icmDocChildNode	oprDocChildNext
icmDocIsText	oprDocIsTitle
icmDocNextNode	oprDocChildNext
icmDocSectionAdd	opcDocSectionAdd
icmDocText	oprDocText
icmDocTextAdd	opcDocTextAdd
icmDumpRegisters	oprProcessorRegisterDump
icmDumpTraceBuffer	oprProcessorTraceBufferDump
icmEnableTraceBuffer	oprProcessorTraceOnAfter
icmErrors	opmErrors
icmExit	oprProcessorExit
icmExitSimulation	opmExit
icmFindInterceptObject	opqObjectExtensionByName
icmFindMMCByName	opqMMCByName
icmFindPSEInterceptObject	opqObjectExtensionByName
icmFindProcessorByName	opqProcessorByName
icmFindProcessorDoubleAttribute	opqObjectParamNext
icmFindProcessorNetPort	opqObjectNetPortConnByName
icmFindProcessorStringAttribute	opqObjectParamByName
icmFindPseByName	opqPeripheralByName
icmFindPseNetPort	opqObjectNetPortConnNext
icmFinish	oprProcessorFinish
icmFlushMemory	oprMemoryFlush
icmFlushProcessorMemory	oprProcessorFlush
icmFreeAttrList	deleted
icmFreeBus	deleted
icmFreeBusBridge	deleted
icmFreeFifo	deleted
icmFreeMMC	deleted
icmFreeMemory	deleted
icmFreeNet	deleted
icmFreePSE	deleted
icmFreeProcessor	deleted

ICM	OP
icmFreeze	oprProcessorFreeze
icmGetAllPlatformCommands	oprExtensionCommandNext
icmGetAllProcessorCommands	oprProcessorCommandNext
icmGetBusHandle	opqParamPtrValue
icmGetBusPortAddrBits	oprBusPortAddrBitsMin
icmGetBusPortAddrBitsMax	oprBusPortAddrBitsMax
icmGetBusPortAddrBitsMin	oprBusPortAddrBitsMin
icmGetBusPortBytes	oprBusPortAddrHi
icmGetBusPortDesc	oprBusPortDescription
icmGetBusPortDomainType	oprBusPortDomainType
icmGetBusPortDomainTypeString	oprBusPortDomainTypeString
icmGetBusPortMustBeConnected	oprBusPortMustBeConnected
icmGetBusPortName	opqObjectName
icmGetBusPortType	oprBusPortTypeEnum
icmGetBusPortTypeString	oprBusPortTypeString
icmGetCurrentTime	oprModuleCurrentTime
icmGetException	oprProcessorException
icmGetExceptionInfoCode	oprExceptionCode
icmGetExceptionInfoDescription	oprExceptionDescription
icmGetExceptionInfoName	oprExceptionName
icmGetFaultAddress	oprProcessorFaultAddress
icmGetFifoHandle	opqParamPtrValue
icmGetFifoPortDesc	oprFIFOPortDescription
icmGetFifoPortName	opqObjectName
icmGetFifoPortType	oprFIFOPortTypeEnum
icmGetFifoPortTypeString	oprFIFOPortTypeString
icmGetFifoPortWidth	oprFIFOPortWidth
icmGetImagefileElfcode	oprApplicationElfCode
icmGetImagefileEndian	oprApplicationEndian
icmGetImagefileEntry	oprApplicationEntry
icmGetInterceptObjectName	opqObjectName
icmGetInterceptObjectReleaseStatus	opqObjectReleaseStatus
icmGetInterceptObjectVisibility	opqObjectVisibility
icmGetInterceptVlnv	oprObjectVLNV
icmGetMMCName	opqObjectName
icmGetMMCPortName	opqObjectName
icmGetMMCReleaseStatus	opqObjectReleaseStatus
icmGetMMCVisibility	opqObjectVisibility
icmGetMMCVlnv	oprObjectVLNV
icmGetMemoryHandle	opqParamPtrValue
icmGetMode	oprProcessorMode
icmGetModeInfoCode	oprModeCode

ICM	OP
icmGetModeInfoDescription	oprModeDescription
icmGetModeInfoName	oprModeName
icmGetNetHandle	opqParamPtrValue
icmGetNetPortDesc	oprNetPortDescription
icmGetNetPortMustBeConnected	oprNetPortMustBeConnected
icmGetNetPortName	opqObjectName
icmGetNetPortType	oprNetPortType
icmGetNetPortTypeString	oprNetPortTypeString
icmGetNextBusPortInfo	opqObjectBusPortNext
icmGetNextException	oprProcessorExceptionNext
icmGetNextFifoPortInfo	opqObjectFIFOPortNext
icmGetNextInstructionAddress	oprProcessorInstructionNext
icmGetNextInterceptParamInfo	opqObjectFormalNext
icmGetNextMMCBusPortInfo	opqObjectBusPortNext
icmGetNextMMCParamInfo	opqObjectFormalNext
icmGetNextMode	oprProcessorModeNext
icmGetNextNetPortInfo	opqObjectNetPortNext
icmGetNextPSEBusPortInfo	opqObjectBusPortNext
icmGetNextPSENetPortInfo	opqObjectNetPortNext
icmGetNextPSEPacketnetPort	opqObjectPacketnetPortNext
icmGetNextPSEParamInfo	opqObjectFormalNext
icmGetNextParamEnum	oprFormalEnumNext
icmGetNextProcessorBusPortInfo	opqObjectBusPortNext
icmGetNextProcessorFifoPortInfo	opqObjectFIFOPortNext
icmGetNextProcessorNetPortInfo	opqObjectNetPortNext
icmGetNextProcessorParamInfo	opqObjectFormalNext
icmGetNextReg	oprProcessorRegNext
icmGetNextRegGroup	oprProcessorRegGroupNext
icmGetNextRegInGroup	oprRegGroupRegNext
icmGetNextTriggeredWatchPoint	oprModuleWatchpointNext
icmGetPC	oprProcessorPC
icmGetPCDS	oprProcessorPCDS
icmGetPSEDoc	oprPeripheralDocNodeNext
icmGetPSEHandle	opqParamPtrValue
icmGetPSEName	opqObjectName
icmGetPSEReleaseStatus	opqObjectReleaseStatus
icmGetPSESaveRestoreSupported	opqObjectSaveRestoreSupported
icmGetPSEVisibility	opqObjectVisibility
icmGetPSEVlnv	oprObjectVLNV
icmGetPacketnetMaxBytes	opqPacketnetMaxBytes
icmGetPacketnetName	opqObjectName
icmGetPacketnetPortDesc	opqPacketnetPortDescription

ICM	OP
icmGetPacketnetPortMustBeConnected	opqPacketnetPortMustBeConnected
icmGetPacketnetPortName	opqPacketnetPortName
icmGetPacketnetPortNet	opqPacketnetPortConnPacketnet
icmGetParamDesc	oprFormalDescription
icmGetParamEnumDesc	oprEnumDescription
icmGetParamEnumName	opqObjectName
icmGetParamEnumValue	oprEnumValue
icmGetParamName	oprFormalName
icmGetParamType	oprFormalType
icmGetParamTypeString	oprFormalTypeString
icmGetPlatformName	opqObjectName
icmGetPlatformPurpose	opqModulePurpose
icmGetPlatformReleaseStatus	opqObjectReleaseStatus
icmGetProcessorClocks	oprProcessorClocks
icmGetProcessorDataBus	opqObjectBusPortConnNext
icmGetProcessorDefaultSemihost	oprProcessorDefaultSemihost
icmGetProcessorDesc	oprProcessorDescription
icmGetProcessorDoc	oprProcessorDocNodeNext
icmGetProcessorElfcode	oprProcessorElfCodes
icmGetProcessorEndian	oprProcessorEndian
icmGetProcessorFamily	oprProcessorFamily
icmGetProcessorGdbFlags	oprProcessorGdbFlags
icmGetProcessorGdbPath	oprProcessorGdbPath
icmGetProcessorGroupH	oprProcessorGroupH
icmGetProcessorGroupL	oprProcessorQLQualified
icmGetProcessorHandle	opqParamPtrValue
icmGetProcessorHelper	oprProcessorHelper
icmGetProcessorICount	oprProcessorICount
icmGetProcessorInstructionBus	opqObjectBusPortConnNext
icmGetProcessorLoadPhysical	oprProcessorLoadPhysical
icmGetProcessorName	opqObjectName
icmGetProcessorQLQualified	oprProcessorQLQualified
icmGetProcessorReleaseStatus	opqObjectReleaseStatus
icmGetProcessorVariant	oprProcessorVariant
icmGetProcessorVisibility	opqObjectVisibility
icmGetProcessorVlnv	oprObjectVLNV
icmGetRegByIndex	oprProcessorRegByIndex
icmGetRegByName	oprProcessorRegByName
icmGetRegByUsage	oprProcessorRegByUsage
icmGetRegGroupByName	oprProcessorRegGroupByName
icmGetRegGroupName	oprRegGroupName
icmGetRegInfoAccess	oprRegReadOnly

ICM	OP
icmGetRegInfoAccessString	oprRegAccessString
icmGetRegInfoBits	oprRegBits
icmGetRegInfoDesc	oprRegDescription
icmGetRegInfoGdbIndex	oprRegGdbIndex
icmGetRegInfoGroup	oprRegGroup
icmGetRegInfoName	oprRegName
icmGetRegInfoReadOnly	oprRegReadOnly
icmGetRegInfoUsage	oprRegUsageEnum
icmGetRegInfoUsageString	oprRegUsageString
icmGetSMPChild	oprProcessorChild
icmGetSMPData	oprProcessorData
icmGetSMPIndex	oprProcessorIndex
icmGetSMPNextSibling	oprProcessorSiblingNext
icmGetSMPParent	oprProcessorParent
icmGetSMPPrevSibling	oprProcessorPrev
icmGetStatus	oprModuleFinishStatus
icmGetStopReason	oprProcessorStopReason
icmGetVlnvString	opcVLNVString
icmGetWatchPointCurrentValue	oprWatchpointCurrentValue
icmGetWatchPointHighAddress	oprWatchpointAddressHi
icmGetWatchPointLowAddress	oprWatchpointAddressLo
icmGetWatchPointPreviousValue	oprWatchpointPreviousValue
icmGetWatchPointRegister	oprWatchpointRegister
icmGetWatchPointTriggeredBy	oprWatchpointTriggeredBy
icmGetWatchPointType	oprWatchpointTypeEnum
icmGetWatchPointUserData	oprWatchpointUserData
icmHalt	oprProcessorHalt
icmIgnoreMessage	opmMessageDisable
icmInFetchContext	oprInFetchContext
icmInitInternal	opcModuleNewFromAttrs
icmInitPlatform	opcModuleNew
icmInstallObjectReader	opcApplicationReaderInstall
icmInterrupt	oprInterrupt
icmInterruptRSP	oprInterruptRSP
icmIsFrozen	oprProcessorFrozen
icmIterAllChildren	oprProcessorIterChildren
icmIterAllDescendants	oprProcessorIterDescendants
icmIterAllModelParameters	opqFormalsShow
icmIterAllProcessors	oprProcessorIterAll
icmIterAllUserAttributes	opqFormalsShow
icmLastMessage	opmLastMessage
icmLegalUsageEnable	(not required)

ICM	OP
icmLoadBus	opcBusApplicationLoad
icmLoadModelHook	opcInit
icmLoadProcessorMemory	opcProcessorApplicationLoad
icmLoadProcessorMemoryOffset	opcProcessorApplicationLoad
icmLoadSymbols	opcMemoryApplicationLoad
icmMMRegBits	opqMMRegisterBits
icmMMRegDescription	opqMMRegisterDescription
icmMMRegName	opqMMRegisterName
icmMMRegOffset	opqMMRegisterOffset
icmMapExternalMemory	opcBusRegionAsCallbacks
icmMapExternalNativeMemory	opcBusRegionAsCallbacks
icmMapLocalMemory	opcMemoryNew
icmMapNativeMemory	oprDynamicNativeMemory
icmMemoryRestoreState	oprMemoryStateRestore
icmMemoryRestoreStateFile	oprMemoryStateRestoreFile
icmMemorySaveState	oprMemoryStateSave
icmMemorySaveStateFile	oprMemoryStateSaveFile
icmMessage	opmMessage
icmMessageQuiet	opmMessageQuiet
icmMessageSetNoWarn	opmMessageSetNoWarn
icmMessageSetQuiet	opmMessageSetQuiet
icmMessageVerbose	opmMessageVerbose
icmNetPortDirection	oprNetPortType
icmNetPortName	opqObjectName
icmNewAttrList	opcParamVoid
icmNewBus	opcBusNew
icmNewBusBridge	opcBridgeNew
icmNewBusWithHandle	opcParamPtrSet
icmNewFifo	opcFIFONew
icmNewFifoWithHandle	opcParamPtrSet
icmNewMMC	opcMMCNew
icmNewMemory	opcMemoryNew
icmNewMemoryWithHandle	opcParamPtrSet
icmNewNet	opcNetNew
icmNewNetWithHandle	opcParamPtrSet
icmNewPSE	opcPeripheralNew
icmNewPSEWithHandle	opcParamPtrSet
icmNewPacketnet	opcPacketnetNew
icmNewProcessor	opcProcessorNewWithSemihost
icmNewProcessorIASAttrs	opcProcessorNewFromAttrs
icmNewProcessorWithHandle	opcParamPtrSet
icmNextBusPortMMRegInfo	opqBusPortMMRegisterNext

ICM	OP
icmNextInterceptObject	opqProcessorExtensionNext
icmNextMmc	opqMMCNext
icmNextProcessor	opqProcessorNext
icmNextProcessorNetPort	opqObjectNetPortConnNext
icmNextPse	opqPeripheralNext
icmNextPseNetPort	opqObjectNetPortConnNext
icmNoBanner	opmNoBanner
icmOverride	opcParamOverrideString
icmPrintAllBusConnections	opqModuleShow
icmPrintAllPacketnetConnections	(not required)
icmPrintBusConnections	opqModuleBusShow
icmPrintNetConnections	opqNetShow
icmPrintf	opmPrintf
icmProcessorIsVisible	deleted
icmProcessorRestoreState	oprProcessorStateRestore
icmProcessorRestoreStateFile	oprProcessorStateRestoreFile
icmProcessorSaveState	oprProcessorStateSave
icmProcessorSaveStateFile	oprProcessorStateSaveFile
icmReadBus	oprProcessorBusRead
icmReadMemory	oprMemoryRead
icmReadObject	opcProcessorApplicationRead
icmReadObjectFileHeader	opcApplicationHeaderRead
icmReadObjectFileHeaderInfo	opcApplicationHeaderRead
icmReadObjectFileInfo	opcProcessorApplicationRead
icmReadProcessorMemory	oprProcessorRead
icmReadReg	oprProcessorRegRead
icmReadRegInfoValue	oprRegRead
icmResetErrors	opmResetErrors
icmResetWatchPoint	oprWatchpointReset
icmSMPIsLeaf	oprProcessorIsLeaf
icmSetAddressBreakpoint	oprProcessorBreakpointAddrSet
icmSetBusAccessWatchPoint	oprBusWatchpointAccessSet
icmSetBusReadWatchPoint	oprBusWatchpointReadSet
icmSetBusWriteWatchPoint	oprBusWatchpointWriteSet
icmSetContextString	deleted
icmSetDebugMode	deleted
icmSetDebugNotifiers	opcDebuggerNotifiersAdd
icmSetDebugStopTime	oprModuleSetDebugStopTime
icmSetExceptionWatchPoint	oprProcessorExceptionWatchpointSet
icmSetICountBreakpoint	oprProcessorBreakpointICountSet
icmSetMemoryAccessWatchPoint	oprBusWatchpointAccessSet
icmSetMemoryReadWatchPoint	oprBusWatchpointReadSet

ICM	OP
icmSetMemoryWriteWatchPoint	oprBusWatchpointWriteSet
icmSetModeWatchPoint	oprProcessorModeWatchpointSet
icmSetPC	oprProcessorPCSet
icmSetPSEGdbPath	deleted
icmSetPSEdiagnosticLevel	oprPeripheralDiagnosticLevelSet
icmSetPersonality	opcLicPersonalitySet
icmSetPlatformName	opcModuleNameChange
icmSetPlatformStatus	Not required: supplied by modelAttrs
icmSetProcessorAccessWatchPoint	oprProcessorWatchpointAccessSet
icmSetProcessorGdbBasic	deleted
icmSetProcessorGdbPath	deleted
icmSetProcessorReadWatchPoint	oprProcessorWatchpointReadSet
icmSetProcessorWriteWatchPoint	oprProcessorWatchpointWriteSet
icmSetProduct	opcProductSet
icmSetRegisterWatchPoint	oprProcessorRegWatchpointSet
icmSetSimulationRandomSeed	oprModuleSetSimulationRandomSeed
icmSetSimulationStopTime	oprModuleSetSimulationStopTime
icmSetSimulationStopTimeDouble	oprModuleSetSimulationStopTime
icmSetSimulationTimePrecision	oprModuleSetSimulationTimePrecision
icmSetSimulationTimePrecisionDouble	oprModuleSetSimulationTimePrecision
icmSetSimulationTimeSlice	oprModuleSetSimulationTimeSlice
icmSetSimulationTimeSliceDouble	oprModuleSetSimulationTimeSlice
icmSetTextOutputFn	opmMessageDirect
icmSetWallClockFactor	oprModuleSetWallClockFactor
icmSimulate	oprProcessorSimulate
icmSimulatePlatform	oprModuleSimulate
icmSimulationEnding	oprModulePostSimulate
icmSimulationStarting	oprModulePreSimulate
icmTerminate	opcTerminate
icmTraceOffAfter	oprProcessorTraceOffAfter
icmTraceOnAfter	oprProcessorTraceOnAfter
icmTriggerAfter	oprModuleTriggerAdd
icmTryVlnvString	opcVLNVString
icmUnbridgeBuses	oprDynamicUnbridge
icmUnfreeze	oprProcessorUnfreeze
icmVAbort	opmVAbort
icmVMessage	opmVMessage
icmVPrintf	opmVPrintf
icmWriteBus	oprProcessorBusWrite
icmWriteMemory	oprMemoryWrite
icmWriteNet	oprNetWrite
icmWriteNetPort	oprNetWrite

ICM	OP
icmWritePacketnet	oprPacketnetWrite
icmWriteProcessorMemory	oprProcessorWrite
icmWriteReg	oprProcessorRegWrite
icmWriteRegInfoValue	oprRegWrite
icmYield	oprProcessorYield