OVP Guide to Using Processor Models

Model specific information for Microsemi_MiV_RV32IMA

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

Author	Imperas Software Limited
Version	20210408.0
Filename	OVP_Model_Specific_Information_microsemi_riscv_MiV_RV32IMA.pdf
Created	5 May 2021
Status	OVP Standard Release

Copyright Notice

Right to Copy Documentation

Destination Control Statement

Disclaimer

Model Release Status

This model is released as part of OVP releases and is included in OVPworld packages. Please visit OVPworld.org.

Contents

Chapter 1

Overview

This document provides the details of an OVP Fast Processor Model variant.

OVP Fast Processor Models are written in C and provide a C API for use in C based platforms. The models also provide a native interface for use in SystemC TLM2 platforms.

The models are written using the OVP VMI API that provides a Virtual Machine Interface that defines the behavior of the processor. The VMI API makes a clear line between model and simulator allowing very good optimization and world class high speed performance. Most models are provided as a binary shared object and also as source. This allows the download and use of the model binary or the use of the source to explore and modify the model.

The models are run through an extensive QA and regression testing process and most model families are validated using technology provided by the processor IP owners. There is a companion document (OVP Guide to Using Processor Models) which explains the general concepts of OVP Fast Processor Models and their use. It is downloadable from the OVPworld website documentation pages.

1.1 Description

RISC-V MiV_RV32IMA 32-bit processor model

1.2 Licensing

This Model is released under the Open Source Apache 2.0

1.3 Extensions

1.3.1 Extensions Enabled by Default

The model has the following architectural extensions enabled, and the following bits in the misa CSR Extensions field will be set upon reset:

misa bit 0: extension A (atomic instructions)

misa bit 8: RV32I/RV64I/RV128I base integer instruction set

misa bit 12: extension M (integer multiply/divide instructions)

To specify features that can be dynamically enabled or disabled by writes to the misa register in addition to those listed above, use parameter "add_Extensions_mask". This is a string parameter containing the feature letters to add; for example, value "DV" indicates that double-precision floating point and the Vector Extension can be enabled or disabled by writes to the misa register, if supported on this variant.

Legacy parameter "misa_Extensions_mask" can also be used. This Uns32-valued parameter specifies all writable bits in the misa Extensions field, replacing any permitted bits defined in the base variant.

Note that any features that are indicated as present in the misa mask but absent in the misa will be ignored. See the next section.

1.3.2 Available Extensions Not Enabled by Default

The following extensions are supported by the model, but not enabled by default in this variant:

misa bit 1: extension B (bit manipulation extension)

misa bit 2: extension C (compressed instructions)

misa bit 3: extension D (double-precision floating point)

misa bit 4: RV32E base integer instruction set (embedded)

misa bit 5: extension F (single-precision floating point)

misa bit 7: extension H (hypervisor)

misa bit 10: extension K (cryptographic)

misa bit 13: extension N (user-level interrupts)

misa bit 18: extension S (Supervisor mode)

misa bit 20: extension U (User mode)

misa bit 21: extension V (vector extension)

misa bit 23: extension X (non-standard extensions present)

To add features from this list to the base variant, use parameter "add_Extensions". This is a string parameter containing the feature letters to add; for example, value "DV" indicates that double-precision floating point and the Vector Extension should be enabled, if they are currently absent

and are available on this variant.

Legacy parameter "misa_Extensions" can also be used. This Uns32-valued parameter specifies the reset value for the misa CSR Extensions field, replacing any permitted bits defined in the base variant.

1.4 General Features

On this variant, the Machine trap-vector base-address register (mtvec) is writable. It can instead be configured as read-only using parameter "mtvec_is_ro".

Values written to "mtvec" are masked using the value 0xfffffffd. A different mask of writable bits may be specified using parameter "mtvec_mask" if required. In addition, when Vectored interrupt mode is enabled, parameter "tvec_align" may be used to specify additional hardware-enforced base address alignment. In this variant, "tvec_align" defaults to 0, implying no alignment constraint.

The initial value of "mtvec" is 0x0. A different value may be specified using parameter "mtvec" if required.

On reset, the model will restart at address 0x60000000. A different reset address may be specified using parameter "reset_address" or applied using optional input port "reset_addr" if required.

On an NMI, the model will restart at address 0x0. A different NMI address may be specified using parameter "nmi_address" or applied using optional input port "nmi_addr" if required.

WFI will halt the processor until an interrupt occurs. It can instead be configured as a NOP using parameter "wfi_is_nop". WFI timeout wait is implemented with a time limit of 0 (i.e. WFI causes an Illegal Instruction trap in Supervisor mode when mstatus.TW=1).

The "cycle" CSR is implemented in this variant. Set parameter "cycle_undefined" to True to instead specify that "cycle" is unimplemented and reads of it should trap to Machine mode.

The "time" CSR is implemented in this variant. Set parameter "time_undefined" to True to instead specify that "time" is unimplemented and reads of it should trap to Machine mode. Usually, the value of the "time" CSR should be provided by the platform - see notes below about the artifact "CSR" bus for information about how this is done.

The "instret" CSR is implemented in this variant. Set parameter "instret_undefined" to True to instead specify that "instret" is unimplemented and reads of it should trap to Machine mode.

Unaligned memory accesses are not supported by this variant. Set parameter "unaligned" to "T" to enable such accesses.

Unaligned memory accesses are not supported for AMO instructions by this variant. Set parameter "unalignedAMO" to "T" to enable such accesses.

A PMP unit is not implemented by this variant. Set parameter "PMP_registers" to indicate that the unit should be implemented with that number of PMP entries.

LR/SC instructions are implemented with a 1-byte reservation granule. A different granule size may be specified using parameter "lr_sc_grain".

1.5 CLIC

The model can be configured to implement a Core Local Interrupt Controller (CLIC) using parameter "CLICLEVELS"; when non-zero, the CLIC is present with the specified number of interrupt levels (2-256), as described in the RISC-V Core-Local Interrupt Controller specification, and further parameters are made available to configure other aspects of the CLIC. "CLICLEVELS" is zero in this variant, indicating that a CLIC is not implemented.

1.6 Load-Reserved/Store-Conditional Locking

By default, LR/SC locking is implemented automatically by the model and simulator, with a reservation granule defined by the "lr_sc_grain" parameter. It is also possible to implement locking externally to the model in a platform component, using the "LR_address", "SC_address" and "SC_valid" net ports, as described below.

The "LR_address" output net port is written by the model with the address used by a load-reserved instruction as it executes. This port should be connected as an input to the external lock management component, which should record the address, and also that an LR/SC transaction is active.

The "SC_address" output net port is written by the model with the address used by a store-conditional instruction as it executes. This should be connected as an input to the external lock management component, which should compare the address with the previously-recorded load-reserved address, and determine from this (and other implementation-specific constraints) whether the store should succeed. It should then immediately write the Boolean success/fail code to the "SC_valid" input net port of the model. Finally, it should update state to indicate that an LR/SC transaction is no longer active.

It is also possible to write zero to the "SC_valid" input net port at any time outside the context of a store-conditional instruction, which will mark any active LR/SC transaction as invalid.

Irrespective of whether LR/SC locking is implemented internally or externally, taking any exception or interrupt or executing exception-return instructions (e.g. MRET) will always mark any active LR/SC transaction as invalid.

1.7 Active Atomic Operation Indication

The "AMO_active" output net port is written by the model with a code indicating any current atomic memory operation while the instruction is active. The written codes are:

0: no atomic instruction active

1: AMOMIN active

2: AMOMAX active

3: AMOMINU active

4: AMOMAXU active

5: AMOADD active

6: AMOXOR active

7: AMOOR active

8: AMOAND active

9: AMOSWAP active

10: LR active

11: SC active

1.8 Interrupts

The "reset" port is an active-high reset input. The processor is halted when "reset" goes high and resumes execution from the reset address specified using the "reset_address" parameter or "reset_addr" port when the signal goes low. The "mcause" register is cleared to zero.

The "nmi" port is an active-high NMI input. The processor resumes execution from the address specified using the "nmi_address" parameter or "nmi_addr" port when the NMI signal goes high. The "mcause" register is cleared to zero.

All other interrupt ports are active high. For each implemented privileged execution level, there are by default input ports for software interrupt, timer interrupt and external interrupt; for example, for Machine mode, these are called "MSWInterrupt", "MTimerInterrupt" and "MExternalInterrupt", respectively. When the N extension is implemented, ports are also present for User mode. Parameter "unimp_int_mask" allows the default behavior to be changed to exclude certain interrupt ports. The parameter value is a mask in the same format as the "mip" CSR; any interrupt corresponding to a non-zero bit in this mask will be removed from the processor and read as zero in "mip", "mie" and "mideleg" CSRs (and Supervisor and User mode equivalents if implemented).

Parameter "external_int_id" can be used to enable extra interrupt ID input ports on each hart. If the parameter is True then when an external interrupt is applied the value on the ID port is sampled and used to fill the Exception Code field in the "mcause" CSR (or the equivalent CSR for other execution levels). For Machine mode, the extra interrupt ID port is called "MExternalInterruptID".

The "deferint" port is an active-high artifact input that, when written to 1, prevents any pending-and-enabled interrupt being taken (normally, such an interrupt would be taken on the next instruction after it becomes pending-and-enabled). The purpose of this signal is to enable alignment with hardware models in step-and-compare usage.

1.9 Debug Mode

The model can be configured to implement Debug mode using parameter "debug_mode". This implements features described in Chapter 4 of the RISC-V External Debug Support specification with version specified by parameter "debug_version" (see References). Some aspects of this mode are not defined in the specification because they are implementation-specific; the model provides infrastructure to allow implementation of a Debug Module using a custom harness. Features added

are described below.

Parameter "debug_mode" can be used to specify three different behaviors, as follows:

1. If set to value "vector", then operations that would cause entry to Debug mode result in the processor jumping to the address specified by the "debug_address" parameter. It will execute at this address, in Debug mode, until a "dret" instruction causes return to non-Debug mode. Any exception generated during this execution will cause a jump to the address specified by the "dexc_address" parameter.

2. If set to value "interrupt", then operations that would cause entry to Debug mode result in the processor simulation call (e.g. opProcessorSimulate) returning, with a stop reason of OP_SR_INTERRUPT. In this usage scenario, the Debug Module is implemented in the simulation harness.

3. If set to value "halt", then operations that would cause entry to Debug mode result in the processor halting. Depending on the simulation environment, this might cause a return from the simulation call with a stop reason of OP_SR_HALT, or debug mode might be implemented by another platform component which then restarts the debugged processor again.

1.9.1 Debug State Entry

The specification does not define how Debug mode is implemented. In this model, Debug mode is enabled by a Boolean pseudo-register, "DM". When "DM" is True, the processor is in Debug mode. When "DM" is False, mode is defined by "mstatus" in the usual way.

Entry to Debug mode can be performed in any of these ways:

1. By writing True to register "DM" (e.g. using opProcessorRegWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate), dcsr cause will be reported as trigger;

2. By writing a 1 then 0 to net "haltreq" (using opNetWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate);

3. By writing a 1 to net "resethaltreq" (using opNetWrite) while the "reset" signal undergoes a negedge transition, followed by simulation of at least one cycle (e.g. using opProcessorSimulate);

4. By executing an "ebreak" instruction when Debug mode entry for the current processor mode is enabled by dcsr.ebreakm, dcsr.ebreaks or dcsr.ebreaku.

In all cases, the processor will save required state in "dpc" and "dcsr" and then perform actions described above, depending in the value of the "debug_mode" parameter.

1.9.2 Debug State Exit

Exit from Debug mode can be performed in any of these ways:

1. By writing False to register "DM" (e.g. using opProcessorRegWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate);

2. By executing an "dret" instruction when Debug mode.

In both cases, the processor will perform the steps described in section 4.6 (Resume) of the Debug

specification.

1.9.3 Debug Registers

When Debug mode is enabled, registers "dcsr", "dpc", "dscratch0" and "dscratch1" are implemented as described in the specification. These may be manipulated externally by a Debug Module using opProcessorRegRead or opProcessorRegWrite; for example, the Debug Module could write "dcsr" to enable "ebreak" instruction behavior as described above, or read and write "dpc" to emulate stepping over an "ebreak" instruction prior to resumption from Debug mode.

1.9.4 Debug Mode Execution

The specification allows execution of code fragments in Debug mode. A Debug Module implementation can cause execution in Debug mode by the following steps:

1. Write the address of a Program Buffer to the program counter using opProcessorPCSet;

2. If "debug_mode" is set to "halt", write 0 to pseudo-register "DMStall" (to leave halted state);

3. If entry to Debug mode was handled by exiting the simulation callback, call opProcessorSimulate or opRootModuleSimulate to resume simulation.

Debug mode will be re-entered in these cases:

1. By execution of an "ebreak" instruction; or:

2. By execution of an instruction that causes an exception.

In both cases, the processor will either jump to the debug exception address, or return control immediately to the harness, with stopReason of OP_SR_INTERRUPT, or perform a halt, depending on the value of the "debug_mode" parameter.

1.9.5 Debug Single Step

When in Debug mode, the processor or harness can cause a single instruction to be executed on return from that mode by setting dcsr.step. After one non-Debug-mode instruction has been executed, control will be returned to the harness. The processor will remain in single-step mode until dcsr.step is cleared.

1.9.6 Debug Ports

Port "DM" is an output signal that indicates whether the processor is in Debug mode

Port "haltreq" is a rising-edge-triggered signal that triggers entry to Debug mode (see above).

Port "resethaltreq" is a level-sensitive signal that triggers entry to Debug mode after reset (see above).

1.10 Debug Mask

It is possible to enable model debug messages in various categories. This can be done statically using the "override_debugMask" parameter, or dynamically using the "debugflags" command. Enabled messages are specified using a bitmask value, as follows:

Value 0x002: enable debugging of PMP and virtual memory state;

Value 0x004: enable debugging of interrupt state.

All other bits in the debug bitmask are reserved and must not be set to non-zero values.

1.11 Integration Support

This model implements a number of non-architectural pseudo-registers and other features to facilitate integration.

1.11.1 CSR Register External Implementation

If parameter "enable_CSR_bus" is True, an artifact 16-bit bus "CSR" is enabled. Slave callbacks installed on this bus can be used to implement modified CSR behavior (use opBusSlaveNew or icmMapExternalMemory, depending on the client API). A CSR with index 0xABC is mapped on the bus at address 0xABC0; as a concrete example, implementing CSR "time" (number 0xC01) externally requires installation of callbacks at address 0xC010 on the CSR bus.

1.11.2 LR/SC Active Address

Artifact register "LRSCAddress" shows the active LR/SC lock address. The register holds all-ones if there is no LR/SC operation active or if LR/SC locking is implemented externally as described above.

1.12 Limitations

Instruction pipelines are not modeled in any way. All instructions are assumed to complete immediately. This means that instruction barrier instructions (e.g. fence.i) are treated as NOPs, with the exception of any Illegal Instruction behavior, which is modeled.

Caches and write buffers are not modeled in any way. All loads, fetches and stores complete immediately and in order, and are fully synchronous. Data barrier instructions (e.g. fence) are treated as NOPs, with the exception of any Illegal Instruction behavior, which is modeled.

Real-world timing effects are not modeled: all instructions are assumed to complete in a single cycle.

Hardware Performance Monitor registers are not implemented and hardwired to zero.

1.13 Verification

All instructions have been extensively tested by Imperas, using tests generated specifically for this model and also reference tests from https://github.com/riscv/riscv-tests.

Also reference tests have been used from various sources including:

https://github.com/riscv/riscv-tests

https://github.com/ucb-bar/riscv-torture

The Imperas OVPsim RISC-V models are used in the RISC-V Foundation Compliance Framework as a functional Golden Reference:

https://github.com/riscv/riscv-compliance

where the simulated model is used to provide the reference signatures for compliance testing. The Imperas OVPsim RISC-V models are used as reference in both open source and commercial instruction stream test generators for hardware design verification, for example:

http://valtrix.in/sting from Valtrix

https://github.com/google/riscv-dv from Google

The Imperas OVPsim RISC-V models are also used by commercial and open source RISC-V Core RTL developers as a reference to ensure correct functionality of their IP.

1.14 References

The Model details are based upon the following specifications:

RISC-V Instruction Set Manual, Volume I: User-Level ISA (User Architecture Version 2.2)

RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Privileged Architecture Version 1.10)

—- HB0800 MiV_RV32IMA_L1_AHB v2.0 HandbookCore

Chapter 2

Configuration

2.1 Location

This model's VLNV is microsemi.ovpworld.org/processor/riscv/1.0.
The model source is usually at:
$IMPERAS_HOME/ImperasLib/source/microsemi.ovpworld.org/processor/riscv/1.0
The model binary is usually at:
$IMPERAS_HOME/lib/$IMPERAS_ARCH/ImperasLib/microsemi.ovpworld.org/processor/riscv/1.0

2.2 GDB Path

The default GDB for this model is: $IMPERAS_HOME/lib/$IMPERAS_ARCH/gdb/riscv-none-embed-gdb.

2.3 Semi-Host Library

The default semi-host library file is riscv.ovpworld.org/semihosting/pk/1.0

2.4 Processor Endian-ness

This is a LITTLE endian model.

2.5 QuantumLeap Support

This processor is qualified to run in a QuantumLeap enabled simulator.

2.6 Processor ELF code

The ELF code supported by this model is: 0xf3.

Chapter 3

All Variants in this model

This model has these variants

Variant	Description
CoreRISCV	
MiV_RV32IMA	(described in this document)

Table 3.1: All Variants in this model

Chapter 4

Bus Master Ports

This model has these bus master ports.

Name	min	max	Connect?	Description
INSTRUCTION	32	34	mandatory	Instruction bus
DATA	32	34	optional	Data bus

Table 4.1: Bus Master Ports

Chapter 5

Bus Slave Ports

This model has no bus slave ports.

Chapter 6

Net Ports

This model has these net ports.

Name	Type	Connect?	Description
reset	input	optional	Reset
reset_addr	input	optional	externally-applied reset address
nmi	input	optional	NMI
nmi_addr	input	optional	externally-applied NMI address
MSWInterrupt	input	optional	Machine software interrupt
MTimerInterrupt	input	optional	Machine timer interrupt
MExternalInterrupt	input	optional	Machine external interrupt
irq_ack_o	output	optional	interrupt acknowledge (pulse)
irq_id_o	output	optional	acknowledged interrupt id (valid during irq_ack_o pulse)
sec_lvl_o	output	optional	current privilege level
LR_address	output	optional	Port written with effective address for LR instruction
SC_address	output	optional	Port written with effective address for SC instruction
SC_valid	input	optional	SC_address valid input signal
AMO_active	output	optional	Port written with code indicating active AMO
deferint	input	optional	Artifact signal causing interrupts to be held off when high

Table 6.1: Net Ports

Chapter 7

FIFO Ports

This model has no FIFO ports.

Chapter 8

Formal Parameters

Name	Type	Description
Fundamental		
variant	Enumeration	Selects variant (either a generic UISA or a specific model)
user_version	Enumeration	Specify required User Architecture version (2.2, 2.3 or 20190305)
priv_version	Enumeration	Specify required Privileged Architecture version (1.10, 1.11, 20190405 or master)
endian	Endian	Model endian
endianFixed	Boolean	Specify that data endianness is fixed (mstatus.{MBE,SBE,UBE} fields are read-only)
misa_MXL	Uns32	Override default value of misa.MXL
misa_Extensions	Uns32	Override default value of misa.Extensions
add_Extensions	String	Add extensions specified by letters to misa.Extensions (for example, specify "VD" to add V and D features)
misa_Extensions_mask	Uns32	Override mask of writable bits in misa.Extensions
add_Extensions_mask	String	Add extensions specified by letters to mask of writable bits in misa.Extensions (for example, specify "VD" to add V and D features)
Debug		
debug_mode	Enumeration	Specify how Debug mode is implemented (none, vector, interrupt or halt)
Simulation_Artifact		
verbose	Boolean	Specify verbose output messages
enable_CSR_bus	Boolean	Add artifact CSR bus port, allowing CSR registers to be externally implemented
CSR_remap	String	Comma-separated list of CSR number mappings, each of the form <csrName>=<number>
Memory		
unaligned	Boolean	Specify whether the processor supports unaligned memory accesses
unalignedAMO	Boolean	Specify whether the processor supports unaligned memory accesses for AMO instructions
lr_sc_grain	Uns32	Specify byte granularity of ll/sc lock region (constrained to a power of two)
PMP_grain	Uns32	Specify PMP region granularity, G (0 =>4 bytes, 1 =>8 bytes, etc)
PMP_registers	Uns32	Specify the number of implemented PMP address registers
PMP_max_page	Uns32	Specify the maximum size of PMP region to map if non-zero (may improve performance; constrained to a power of two)
PMP_decompose	Boolean	Whether unaligned PMP accesses are decomposed into separate aligned accesses
Instruction_CSR_Behavior		
wfi_is_nop	Boolean	Specify whether WFI should be treated as a NOP (if not, halt while waiting for interrupts)
counteren_mask	Uns32	Specify hardware-enforced mask of writable bits in mcounteren/scounteren registers
noinhibit_mask	Uns32	Specify hardware-enforced mask of always-zero bits in mcountinhibit register

cycle_undefined	Boolean	Specify that the cycle CSR is undefined (reads to it are emulated by a Machine mode trap)
time_undefined	Boolean	Specify that the time CSR is undefined (reads to it are emulated by a Machine mode trap)
instret_undefined	Boolean	Specify that the instret CSR is undefined (reads to it are emulated by a Machine mode trap)
Interrupts_Exceptions		
mtvec_is_ro	Boolean	Specify whether mtvec CSR is read-only
tvec_align	Uns32	Specify hardware-enforced alignment of mtvec/stvec/utvec when Vectored interrupt mode enabled
ecode_mask	Uns64	Specify hardware-enforced mask of writable bits in xcause.ExceptionCode
ecode_nmi	Uns64	Specify xcause.ExceptionCode for NMI
tval_zero	Boolean	Specify whether mtval/stval/utval are hard wired to zero
tval_zero_ebreak	Boolean	Specify whether mtval/stval/utval are set to zero by an ebreak
tval_ii_code	Boolean	Specify whether mtval/stval contain faulting instruction bits on illegal instruction exception
xret_preserves_lr	Boolean	Whether an xRET instruction preserves the value of LR
reset_address	Uns64	Override reset vector address
nmi_address	Uns64	Override NMI vector address
local_int_num	Uns32	Specify number of supplemental local interrupts
unimp_int_mask	Uns64	Specify mask of unimplemented interrupts (e.g. 1<<9 indicates Supervisor external interrupt unimplemented)
force_mideleg	Uns64	Specify mask of interrupts always delegated to lower-priority execution level from Machine execution level
no_ideleg	Uns64	Specify mask of interrupts that cannot be delegated to lower-priority execution levels
no_edeleg	Uns64	Specify mask of exceptions that cannot be delegated to lower-priority execution levels
external_int_id	Boolean	Whether to add nets allowing External Interrupt ID codes to be forced
CSR_Masks		
mtvec_mask	Uns64	Specify hardware-enforced mask of writable bits in mtvec register
Trigger		
trigger_num	Uns32	Specify the number of implemented hardware triggers
CSR_Defauts		
mvendorid	Uns64	Override mvendorid register
marchid	Uns64	Override marchid register
mimpid	Uns64	Override mimpid register
mhartid	Uns64	Override mhartid register (or first mhartid of an incrementing sequence if this is an SMP variant)
mtvec	Uns64	Override mtvec register
Fast_Interrupt		
CLICLEVELS	Uns32	Specify number of interrupt levels implemented by CLIC, or 0 if CLIC absent

Table 8.1: Parameters that can be set in: Hart

8.1 Parameters with enumerated types

8.1.1 Parameter user_version

Set to this value	Description
2.2	User Architecture Version 2.2
2.3	Deprecated and equivalent to 20190305
20190305	User Architecture Version 20190305-Base-Ratification

Table 8.2: Values for Parameter user_version

8.1.2 Parameter priv_version

Set to this value	Description
1.10	Privileged Architecture Version 1.10
1.11	Deprecated and equivalent to 20190405
20190405	Privileged Architecture Version 20190405-Priv-MSU-Ratification
master	Privileged Architecture Master Branch (1.12 draft)

Table 8.3: Values for Parameter priv_version

8.1.3 Parameter debug_mode

Set to this value	Description
none	Debug mode not implemented
vector	Debug mode implemented by execution at vector
interrupt	Debug mode implemented by interrupt
halt	Debug mode implemented by halt

Table 8.4: Values for Parameter debug_mode

Chapter 9

Execution Modes

Mode	Code	Description
Machine	3	Machine mode

Table 9.1: Modes implemented in: Hart

Chapter 10

Exceptions

Exception	Code	Description
InstructionAddressMisaligned	0	Fetch from unaligned address
InstructionAccessFault	1	No access permission for fetch
IllegalInstruction	2	Undecoded, unimplemented or disabled instruction
Breakpoint	3	EBREAK instruction executed
LoadAddressMisaligned	4	Load from unaligned address
LoadAccessFault	5	No access permission for load
StoreAMOAddressMisaligned	6	Store/atomic memory operation at unaligned address
StoreAMOAccessFault	7	No access permission for store/atomic memory operation
EnvironmentCallFromMMode	11	ECALL instruction executed in Machine mode
InstructionPageFault	12	Page fault at fetch address
LoadPageFault	13	Page fault at load address
StoreAMOPageFault	15	Page fault at store/atomic memory operation address
MSWInterrupt	67	Machine software interrupt
MTimerInterrupt	71	Machine timer interrupt
MExternalInterrupt	75	Machine external interrupt

Table 10.1: Exceptions implemented in: Hart

Chapter 11

Hierarchy of the model

A CPU core may be configured to instance many processors of a Symmetrical Multi Processor (SMP). A CPU core may also have sub elements within a processor, for example hardware threading blocks.
OVP processor models can be written to include SMP blocks and to have many levels of hierarchy. Some OVP CPU models may have a fixed hierarchy, and some may be configured by settings in a configuration register. Please see the register definitions of this model.
This model documentation shows the settings and hierarchy of the default settings for this model variant.

11.1 Level 1: Hart

This level in the model hierarchy has 2 commands.
This level in the model hierarchy has 3 register groups:

Group name	Registers
Core	33
Machine_Control_and_Status	188
Integration_support	2

Table 11.1: Register groups

This level in the model hierarchy has no children.

Chapter 12

Model Commands

A Processor model can implement one or more **Model Commands** available to be invoked from the simulator command line, from the OP API or from the Imperas Multiprocessor Debugger.

12.1 Level 1: Hart

12.1.1 isync

specify instruction address range for synchronous execution

Argument	Type	Description
-addresshi	Uns64	end address of synchronous execution range
-addresslo	Uns64	start address of synchronous execution range

Table 12.1: isync command arguments

12.1.2 itrace

enable or disable instruction tracing

Argument	Type	Description
-after	Uns64	apply after this many instructions
-enable	Boolean	enable instruction tracing
-instructioncount	Boolean	include the instruction number in each trace
-off	Boolean	disable instruction tracing
-on	Boolean	enable instruction tracing
-registerchange	Boolean	show registers changed by this instruction
-registers	Boolean	show registers after each trace

Table 12.2: itrace command arguments

Chapter 13

Registers

13.1 Level 1: Hart

13.1.1 Core

Registers at level:1, type:Hart group:Core

Name	Bits	Initial-Hex	RW	Description
zero	32	0	r-	
ra	32	0	rw	
sp	32	0	rw	stack pointer
gp	32	0	rw	
tp	32	0	rw	
t0	32	0	rw	
t1	32	0	rw	
t2	32	0	rw	
s0	32	0	rw	
s1	32	0	rw	
a0	32	0	rw	
a1	32	0	rw	
a2	32	0	rw	
a3	32	0	rw	
a4	32	0	rw	
a5	32	0	rw	
a6	32	0	rw	
a7	32	0	rw	
s2	32	0	rw	
s3	32	0	rw	
s4	32	0	rw	
s5	32	0	rw	
s6	32	0	rw	
s7	32	0	rw	
s8	32	0	rw	
s9	32	0	rw	
s10	32	0	rw	
s11	32	0	rw	
t3	32	0	rw	
t4	32	0	rw	
t5	32	0	rw	
t6	32	0	rw	
pc	32	60000000	rw	program counter

Table 13.1: Registers at level 1, type:Hart group:Core

13.1.2 Machine_Control_and_Status

Registers at level:1, type:Hart group:Machine_Control_and_Status

Name	Bits	Initial-Hex	RW	Description
mstatus	32	1800	rw	Machine Status
misa	32	40001101	rw	ISA and Extensions
mie	32	0	rw	Machine Interrupt Enable
mtvec	32	0	rw	Machine Trap-Vector Base-Address
mhpmevent3	32	0	rw	Machine Performance Monitor Event Select 3
mhpmevent4	32	0	rw	Machine Performance Monitor Event Select 4
mhpmevent5	32	0	rw	Machine Performance Monitor Event Select 5
mhpmevent6	32	0	rw	Machine Performance Monitor Event Select 6
mhpmevent7	32	0	rw	Machine Performance Monitor Event Select 7
mhpmevent8	32	0	rw	Machine Performance Monitor Event Select 8
mhpmevent9	32	0	rw	Machine Performance Monitor Event Select 9
mhpmevent10	32	0	rw	Machine Performance Monitor Event Select 10
mhpmevent11	32	0	rw	Machine Performance Monitor Event Select 11
mhpmevent12	32	0	rw	Machine Performance Monitor Event Select 12
mhpmevent13	32	0	rw	Machine Performance Monitor Event Select 13
mhpmevent14	32	0	rw	Machine Performance Monitor Event Select 14
mhpmevent15	32	0	rw	Machine Performance Monitor Event Select 15
mhpmevent16	32	0	rw	Machine Performance Monitor Event Select 16
mhpmevent17	32	0	rw	Machine Performance Monitor Event Select 17
mhpmevent18	32	0	rw	Machine Performance Monitor Event Select 18
mhpmevent19	32	0	rw	Machine Performance Monitor Event Select 19
mhpmevent20	32	0	rw	Machine Performance Monitor Event Select 20
mhpmevent21	32	0	rw	Machine Performance Monitor Event Select 21
mhpmevent22	32	0	rw	Machine Performance Monitor Event Select 22
mhpmevent23	32	0	rw	Machine Performance Monitor Event Select 23
mhpmevent24	32	0	rw	Machine Performance Monitor Event Select 24
mhpmevent25	32	0	rw	Machine Performance Monitor Event Select 25
mhpmevent26	32	0	rw	Machine Performance Monitor Event Select 26
mhpmevent27	32	0	rw	Machine Performance Monitor Event Select 27
mhpmevent28	32	0	rw	Machine Performance Monitor Event Select 28
mhpmevent29	32	0	rw	Machine Performance Monitor Event Select 29
mhpmevent30	32	0	rw	Machine Performance Monitor Event Select 30
mhpmevent31	32	0	rw	Machine Performance Monitor Event Select 31
mscratch	32	0	rw	Machine Scratch
mepc	32	0	rw	Machine Exception Program Counter
mcause	32	0	rw	Machine Cause
mtval	32	0	rw	Machine Trap Value
mip	32	0	rw	Machine Interrupt Pending
pmpcfg0	32	0	rw	Physical Memory Protection Configuration 0
pmpcfg1	32	0	rw	Physical Memory Protection Configuration 1
pmpcfg2	32	0	rw	Physical Memory Protection Configuration 2
pmpcfg3	32	0	rw	Physical Memory Protection Configuration 3
pmpaddr0	32	0	rw	Physical Memory Protection Address 0
pmpaddr1	32	0	rw	Physical Memory Protection Address 1
pmpaddr2	32	0	rw	Physical Memory Protection Address 2
pmpaddr3	32	0	rw	Physical Memory Protection Address 3
pmpaddr4	32	0	rw	Physical Memory Protection Address 4
pmpaddr5	32	0	rw	Physical Memory Protection Address 5

pmpaddr6	32	0	rw	Physical Memory Protection Address 6
pmpaddr7	32	0	rw	Physical Memory Protection Address 7
pmpaddr8	32	0	rw	Physical Memory Protection Address 8
pmpaddr9	32	0	rw	Physical Memory Protection Address 9
pmpaddr10	32	0	rw	Physical Memory Protection Address 10
pmpaddr11	32	0	rw	Physical Memory Protection Address 11
pmpaddr12	32	0	rw	Physical Memory Protection Address 12
pmpaddr13	32	0	rw	Physical Memory Protection Address 13
pmpaddr14	32	0	rw	Physical Memory Protection Address 14
pmpaddr15	32	0	rw	Physical Memory Protection Address 15
mcycle	32	0	rw	Machine Cycle Counter
minstret	32	0	rw	Machine Instructions Retired
mhpmcounter3	32	0	rw	Machine Performance Monitor Counter 3
mhpmcounter4	32	0	rw	Machine Performance Monitor Counter 4
mhpmcounter5	32	0	rw	Machine Performance Monitor Counter 5
mhpmcounter6	32	0	rw	Machine Performance Monitor Counter 6
mhpmcounter7	32	0	rw	Machine Performance Monitor Counter 7
mhpmcounter8	32	0	rw	Machine Performance Monitor Counter 8
mhpmcounter9	32	0	rw	Machine Performance Monitor Counter 9
mhpmcounter10	32	0	rw	Machine Performance Monitor Counter 10
mhpmcounter11	32	0	rw	Machine Performance Monitor Counter 11
mhpmcounter12	32	0	rw	Machine Performance Monitor Counter 12
mhpmcounter13	32	0	rw	Machine Performance Monitor Counter 13
mhpmcounter14	32	0	rw	Machine Performance Monitor Counter 14
mhpmcounter15	32	0	rw	Machine Performance Monitor Counter 15
mhpmcounter16	32	0	rw	Machine Performance Monitor Counter 16
mhpmcounter17	32	0	rw	Machine Performance Monitor Counter 17
mhpmcounter18	32	0	rw	Machine Performance Monitor Counter 18
mhpmcounter19	32	0	rw	Machine Performance Monitor Counter 19
mhpmcounter20	32	0	rw	Machine Performance Monitor Counter 20
mhpmcounter21	32	0	rw	Machine Performance Monitor Counter 21
mhpmcounter22	32	0	rw	Machine Performance Monitor Counter 22
mhpmcounter23	32	0	rw	Machine Performance Monitor Counter 23
mhpmcounter24	32	0	rw	Machine Performance Monitor Counter 24
mhpmcounter25	32	0	rw	Machine Performance Monitor Counter 25
mhpmcounter26	32	0	rw	Machine Performance Monitor Counter 26
mhpmcounter27	32	0	rw	Machine Performance Monitor Counter 27
mhpmcounter28	32	0	rw	Machine Performance Monitor Counter 28
mhpmcounter29	32	0	rw	Machine Performance Monitor Counter 29
mhpmcounter30	32	0	rw	Machine Performance Monitor Counter 30
mhpmcounter31	32	0	rw	Machine Performance Monitor Counter 31
mcycleh	32	0	rw	Machine Cycle Counter High
minstreth	32	0	rw	Machine Instructions Retired High
mhpmcounterh3	32	0	rw	Machine Performance Monitor Counter High 3
mhpmcounterh4	32	0	rw	Machine Performance Monitor Counter High 4
mhpmcounterh5	32	0	rw	Machine Performance Monitor Counter High 5
mhpmcounterh6	32	0	rw	Machine Performance Monitor Counter High 6
mhpmcounterh7	32	0	rw	Machine Performance Monitor Counter High 7
mhpmcounterh8	32	0	rw	Machine Performance Monitor Counter High 8
mhpmcounterh9	32	0	rw	Machine Performance Monitor Counter High 9
mhpmcounterh10	32	0	rw	Machine Performance Monitor Counter High 10
mhpmcounterh11	32	0	rw	Machine Performance Monitor Counter High 11
mhpmcounterh12	32	0	rw	Machine Performance Monitor Counter High 12
mhpmcounterh13	32	0	rw	Machine Performance Monitor Counter High 13
mhpmcounterh14	32	0	rw	Machine Performance Monitor Counter High 14
mhpmcounterh15	32	0	rw	Machine Performance Monitor Counter High 15

mhpmcounterh16	32	0	rw	Machine Performance Monitor Counter High 16
mhpmcounterh17	32	0	rw	Machine Performance Monitor Counter High 17
mhpmcounterh18	32	0	rw	Machine Performance Monitor Counter High 18
mhpmcounterh19	32	0	rw	Machine Performance Monitor Counter High 19
mhpmcounterh20	32	0	rw	Machine Performance Monitor Counter High 20
mhpmcounterh21	32	0	rw	Machine Performance Monitor Counter High 21
mhpmcounterh22	32	0	rw	Machine Performance Monitor Counter High 22
mhpmcounterh23	32	0	rw	Machine Performance Monitor Counter High 23
mhpmcounterh24	32	0	rw	Machine Performance Monitor Counter High 24
mhpmcounterh25	32	0	rw	Machine Performance Monitor Counter High 25
mhpmcounterh26	32	0	rw	Machine Performance Monitor Counter High 26
mhpmcounterh27	32	0	rw	Machine Performance Monitor Counter High 27
mhpmcounterh28	32	0	rw	Machine Performance Monitor Counter High 28
mhpmcounterh29	32	0	rw	Machine Performance Monitor Counter High 29
mhpmcounterh30	32	0	rw	Machine Performance Monitor Counter High 30
mhpmcounterh31	32	0	rw	Machine Performance Monitor Counter High 31
cycle	32	0	r-	Cycle Counter
time	32	0	r-	Timer
instret	32	0	r-	Instructions Retired
hpmcounter3	32	0	r-	Performance Monitor Counter 3
hpmcounter4	32	0	r-	Performance Monitor Counter 4
hpmcounter5	32	0	r-	Performance Monitor Counter 5
hpmcounter6	32	0	r-	Performance Monitor Counter 6
hpmcounter7	32	0	r-	Performance Monitor Counter 7
hpmcounter8	32	0	r-	Performance Monitor Counter 8
hpmcounter9	32	0	r-	Performance Monitor Counter 9
hpmcounter10	32	0	r-	Performance Monitor Counter 10
hpmcounter11	32	0	r-	Performance Monitor Counter 11
hpmcounter12	32	0	r-	Performance Monitor Counter 12
hpmcounter13	32	0	r-	Performance Monitor Counter 13
hpmcounter14	32	0	r-	Performance Monitor Counter 14
hpmcounter15	32	0	r-	Performance Monitor Counter 15
hpmcounter16	32	0	r-	Performance Monitor Counter 16
hpmcounter17	32	0	r-	Performance Monitor Counter 17
hpmcounter18	32	0	r-	Performance Monitor Counter 18
hpmcounter19	32	0	r-	Performance Monitor Counter 19
hpmcounter20	32	0	r-	Performance Monitor Counter 20
hpmcounter21	32	0	r-	Performance Monitor Counter 21
hpmcounter22	32	0	r-	Performance Monitor Counter 22
hpmcounter23	32	0	r-	Performance Monitor Counter 23
hpmcounter24	32	0	r-	Performance Monitor Counter 24
hpmcounter25	32	0	r-	Performance Monitor Counter 25
hpmcounter26	32	0	r-	Performance Monitor Counter 26
hpmcounter27	32	0	r-	Performance Monitor Counter 27
hpmcounter28	32	0	r-	Performance Monitor Counter 28
hpmcounter29	32	0	r-	Performance Monitor Counter 29
hpmcounter30	32	0	r-	Performance Monitor Counter 30
hpmcounter31	32	0	r-	Performance Monitor Counter 31
cycleh	32	0	r-	Cycle Counter High
timeh	32	0	r-	Timer High
instreth	32	0	r-	Instructions Retired High
hpmcounterh3	32	0	r-	Performance Monitor High 3
hpmcounterh4	32	0	r-	Performance Monitor High 4
hpmcounterh5	32	0	r-	Performance Monitor High 5
hpmcounterh6	32	0	r-	Performance Monitor High 6
hpmcounterh7	32	0	r-	Performance Monitor High 7

hpmcounterh8	32	0	r-	Performance Monitor High 8
hpmcounterh9	32	0	r-	Performance Monitor High 9
hpmcounterh10	32	0	r-	Performance Monitor High 10
hpmcounterh11	32	0	r-	Performance Monitor High 11
hpmcounterh12	32	0	r-	Performance Monitor High 12
hpmcounterh13	32	0	r-	Performance Monitor High 13
hpmcounterh14	32	0	r-	Performance Monitor High 14
hpmcounterh15	32	0	r-	Performance Monitor High 15
hpmcounterh16	32	0	r-	Performance Monitor High 16
hpmcounterh17	32	0	r-	Performance Monitor High 17
hpmcounterh18	32	0	r-	Performance Monitor High 18
hpmcounterh19	32	0	r-	Performance Monitor High 19
hpmcounterh20	32	0	r-	Performance Monitor High 20
hpmcounterh21	32	0	r-	Performance Monitor High 21
hpmcounterh22	32	0	r-	Performance Monitor High 22
hpmcounterh23	32	0	r-	Performance Monitor High 23
hpmcounterh24	32	0	r-	Performance Monitor High 24
hpmcounterh25	32	0	r-	Performance Monitor High 25
hpmcounterh26	32	0	r-	Performance Monitor High 26
hpmcounterh27	32	0	r-	Performance Monitor High 27
hpmcounterh28	32	0	r-	Performance Monitor High 28
hpmcounterh29	32	0	r-	Performance Monitor High 29
hpmcounterh30	32	0	r-	Performance Monitor High 30
hpmcounterh31	32	0	r-	Performance Monitor High 31
mvendorid	32	0	r-	Vendor ID
marchid	32	0	r-	Architecture ID
mimpid	32	0	r-	Implementation ID
mhartid	32	0	r-	Hardware Thread ID

Table 13.2: Registers at level 1, type:Hart group:Machine_Control_and_Status

13.1.3 Integration_support

Registers at level:1, type:Hart group:Integration_support

Name	Bits	Initial-Hex	RW	Description
LRSCAddress	32	ffffffff	rw	LR/SC active lock address
commercial	8	0	r-	Commercial feature in use

Table 13.3: Registers at level 1, type:Hart group:Integration_support