



## OVP Guide to Using Processor Models

### Model specific information for OpenHwGroup\_CV64A6

Imperas Software Limited  
Imperas Buildings, North Weston  
Thame, Oxfordshire, OX9 2HA, U.K.  
[docs@imperas.com](mailto:docs@imperas.com)



Author	Imperas Software Limited
Version	20230201.0
Filename	OVP_Model_Specific_Informationopenhwgroup_riscv_CV64A6.pdf
Created	1 February 2023
Status	OVP Standard Release

## Copyright Notice

Copyright (c) 2023 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 readers 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.

## Model Release Status

This model is released as part of OVP releases and is included in OVPworld packages. Please visit [OVPworld.org](http://OVPworld.org).

# Contents

<b>1</b>	<b>Overview</b>	<b>1</b>
1.1	Description	1
1.2	Licensing	1
1.3	Extensions	2
1.3.1	Extensions Enabled by Default	2
1.4	General Features	2
1.4.1	mtvec CSR	2
1.4.2	stvec CSR	3
1.4.3	Reset	3
1.4.4	NMI	3
1.4.5	WFI	3
1.4.6	cycle CSR	3
1.4.7	instret CSR	4
1.4.8	hpmcounter CSR	4
1.4.9	time CSR	4
1.4.10	mcycle CSR	4
1.4.11	minstret CSR	4
1.4.12	mhpmcounter CSR	4
1.4.13	Virtual Memory	4
1.4.14	Unaligned Accesses	5
1.4.15	PMP	5
1.4.16	Time and Timers	5
1.5	Compressed Extension	6
1.5.1	Compressed Extension Parameters	6
1.5.2	Legacy Version 1.10	6
1.5.3	Version 0.70.1	6
1.5.4	Version 0.70.5	6
1.5.5	Version 1.0.0-RC5.7	7
1.6	Privileged Architecture	7
1.6.1	Legacy Version 1.10	7
1.6.2	Version 20190608	7
1.6.3	Version 20211203	7
1.6.4	Version 1.12	7
1.6.5	Version master	8
1.7	Unprivileged Architecture	8
1.7.1	Legacy Version 2.2	8
1.7.2	Version 20191213	8

1.8	Other Extensions . . . . .	8
1.8.1	Zmmul . . . . .	8
1.8.2	Zicsr . . . . .	8
1.8.3	Zifencei . . . . .	9
1.8.4	Zicbom . . . . .	9
1.8.5	Zicbop . . . . .	9
1.8.6	Zicboz . . . . .	9
1.8.7	Svnapot . . . . .	9
1.8.8	Svpbmt . . . . .	9
1.8.9	Svinval . . . . .	10
1.8.10	Smstateen . . . . .	10
1.8.11	Sstc . . . . .	10
1.8.12	Zawrs . . . . .	10
1.9	CLIC . . . . .	10
1.10	Advanced Interrupt Architecture . . . . .	10
1.11	Load-Reserved/Store-Conditional Locking . . . . .	11
1.12	Active Atomic Operation Indication . . . . .	11
1.13	Interrupts . . . . .	12
1.14	Debug Mode . . . . .	12
1.14.1	Debug State Entry . . . . .	13
1.14.2	Debug State Exit . . . . .	13
1.14.3	Debug Registers . . . . .	14
1.14.4	Debug Mode Execution . . . . .	14
1.14.5	Debug Single Step . . . . .	14
1.14.6	Debug Event Priorities . . . . .	14
1.14.7	Debug Ports . . . . .	15
1.14.8	Debug Mode Versions . . . . .	15
1.14.9	Version 0.13.2-DRAFT . . . . .	15
1.14.10	Version 0.14.0-DRAFT . . . . .	15
1.14.11	Version 1.0.0-STABLE . . . . .	15
1.14.12	Version 1.0-STABLE . . . . .	15
1.15	Debug Mask . . . . .	15
1.16	Integration Support . . . . .	16
1.16.1	CSR Register External Implementation . . . . .	16
1.16.2	LR/SC Active Address . . . . .	16
1.16.3	Page Table Walk Introspection . . . . .	16
1.17	Instruction Disassembly . . . . .	17
1.18	Limitations . . . . .	17
1.19	Verification . . . . .	17
1.20	References . . . . .	18
<b>2</b>	<b>Openhwgroup-Specific Extensions</b>	<b>19</b>
2.1	PULP_XPULP Extension Status . . . . .	19
2.2	PMA Extension Status . . . . .	19
2.3	Custom Bus Fault Extension Status . . . . .	19
2.4	Secure Exceptions Status . . . . .	20
<b>3</b>	<b>Configuration</b>	<b>21</b>

3.1	Location . . . . .	21
3.2	GDB Path . . . . .	21
3.3	Semi-Host Library . . . . .	21
3.4	Processor Endian-ness . . . . .	21
3.5	QuantumLeap Support . . . . .	21
3.6	Processor ELF code . . . . .	21
<b>4</b>	<b>All Variants in this model</b>	<b>22</b>
<b>5</b>	<b>Bus Master Ports</b>	<b>23</b>
<b>6</b>	<b>Bus Slave Ports</b>	<b>24</b>
<b>7</b>	<b>Net Ports</b>	<b>25</b>
<b>8</b>	<b>FIFO Ports</b>	<b>26</b>
<b>9</b>	<b>Formal Parameters</b>	<b>27</b>
9.1	Extension Parameters . . . . .	30
9.2	Parameters with enumerated types . . . . .	30
9.2.1	Parameter user_version . . . . .	30
9.2.2	Parameter priv_version . . . . .	30
9.2.3	Parameter compress_version . . . . .	31
9.2.4	Parameter rnmi_version . . . . .	31
9.2.5	Parameter debug_mode . . . . .	31
9.2.6	Parameter lr_sc_constraint . . . . .	31
9.2.7	Parameter amo_constraint . . . . .	31
9.2.8	Parameter Zcea_version . . . . .	32
9.2.9	Parameter Zceb_version . . . . .	32
9.2.10	Parameter Zcee_version . . . . .	32
9.3	Parameter values . . . . .	32
<b>10</b>	<b>Execution Modes</b>	<b>36</b>
<b>11</b>	<b>Exceptions</b>	<b>37</b>
<b>12</b>	<b>Hierarchy of the model</b>	<b>38</b>
12.1	Level 1: Hart . . . . .	38
<b>13</b>	<b>Model Commands</b>	<b>39</b>
13.1	Level 1: Hart . . . . .	39
13.1.1	debugflags . . . . .	39
13.1.2	dumpTLB . . . . .	39
13.1.2.1	Argument description . . . . .	39
13.1.3	getCSRIndex . . . . .	39
13.1.4	isync . . . . .	39
13.1.5	itrace . . . . .	40
13.1.6	listCSRs . . . . .	40
13.1.6.1	Argument description . . . . .	40

<b>14 Registers</b>	<b>41</b>
14.1 Level 1: Hart . . . . .	41
14.1.1 Core . . . . .	41
14.1.2 User_Control_and_Status . . . . .	42
14.1.3 Supervisor_Control_and_Status . . . . .	42
14.1.4 Machine_Control_and_Status . . . . .	43
14.1.5 Integration_support . . . . .	45

# 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 CV64A6 64-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 corresponding bits in the misa CSR Extensions field will be set upon reset:

misa bit 0: extension A (atomic instructions)

misa bit 2: extension C (compressed instructions)

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

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

misa bit 18: extension S (Supervisor mode)

misa bit 20: extension U (User mode)

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. Parameter “sub\_Extensions\_mask” can be used to disable dynamic update of features in the same way.

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.

## 1.4 General Features

### 1.4.1 mtvec CSR

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 0xffffffffffffd. 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.

If parameter “mtvec\_sext” is True, values written to “mtvec” are sign-extended from the most-significant writable bit. In this variant, “mtvec\_sext” is False, indicating that “mtvec” is not sign-extended.

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



### 1.4.2 stvec CSR

Values written to “stvec” are masked using the value 0xffffffffffffd. A different mask of writable bits may be specified using parameter “stvec\_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.

If parameter “stvec\_sext” is True, values written to “stvec” are sign-extended from the most-significant writable bit. In this variant, “stvec\_sext” is False, indicating that “stvec” is not sign-extended.

### 1.4.3 Reset

On reset, the model will restart at address 0x0. A different reset address may be specified using parameter “reset\_address” or applied using optional input port “reset\_addr” if required.

### 1.4.4 NMI

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. The cause reported on an NMI is 0x0 by default; a different cause may be specified using parameter “ecode\_nmi” or applied using optional input port “nmi\_cause” if required.

If parameter “rnmi\_version” is not “none”, resumable NMIs are supported, managed by additional CSRs “mnscratch”, “mnepc”, “mncause” and “mnstatus”, following the indicated version of the Resumable NMI extension proposal. In this variant, “rnmi\_version” is “none”.

The NMI input is level-sensitive. To instead specify that the NMI input is latched on the rising edge of the NMI signal, set parameter “nmi.is.latched” to True.

### 1.4.5 WFI

WFI will halt the processor until an interrupt occurs. It can instead be configured as a NOP by setting parameter “wfi.is.nop” to True.

The nominal time limit for WFI instructions can be set by parameter “TW\_time\_limit”. In this variant, the time limit is 0 cycles.

Parameter “wfi\_resume\_not\_trap” is 0 on this variant, meaning that pending wakeup events when WFI is executed will not prevent a trap occurring. If “wfi\_resume\_not\_trap” is set to 1 then pending wakeup events when WFI is executed will cause the WFI to be treated as a NOP.

### 1.4.6 cycle CSR

The “cycle” CSR is implemented in this variant. Set parameter “cycle\_undefined” to True to instead specify that “cycle” is unimplemented and accesses should cause Illegal Instruction traps.

#### 1.4.7 instret CSR

The “instret” CSR is implemented in this variant. Set parameter “instret\_undefined” to True to instead specify that “instret” is unimplemented and accesses should cause Illegal Instruction traps.

#### 1.4.8 hpmcounter CSR

The “hpmcounter” CSRs are implemented in this variant. Set parameter “hpmcounter\_undefined” to True to instead specify that “hpmcounter” CSRs are unimplemented and accesses should cause Illegal Instruction traps.

#### 1.4.9 time CSR

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 cause Illegal Instruction traps. Usually, the value of the “time” CSR should be provided by the platform - see section “Time and Timers” for more information.

#### 1.4.10 mcycle CSR

The “mcycle” CSR is implemented in this variant. Set parameter “mcycle\_undefined” to True to instead specify that “mcycle” is unimplemented and accesses should cause Illegal Instruction traps.

#### 1.4.11 minstret CSR

The “minstret” CSR is implemented in this variant. Set parameter “minstret\_undefined” to True to instead specify that “minstret” is unimplemented and accesses should cause Illegal Instruction traps.

#### 1.4.12 mhpmcounter CSR

The “mhpmcounter” CSRs are implemented in this variant. Set parameter “mhpmcounter\_undefined” to True to instead specify that “mhpmcounter” CSRs are unimplemented and accesses should cause Illegal Instruction traps.

#### 1.4.13 Virtual Memory

This variant supports address translation modes 0 (bare), 8 (Sv39), 9 (Sv48) and 10 (Sv57). Use parameter “Sv\_modes” to specify a bit mask of different implemented modes if required; for example, setting “Sv\_modes” to  $(1 < 0) + (1 < 8)$  indicates that mode 0 (bare) and mode 8 (Sv39) are implemented. These indices correspond to writable values in the satp.MODE CSR field.

A 0-bit ASID is implemented. Use parameter “ASID\_bits” to specify a different implemented ASID size if required.

TLB behavior is controlled by parameter “ASIDCacheSize”. If this parameter is 0, then an unlimited number of TLB entries will be maintained concurrently. If this parameter is non-zero, then only TLB entries for up to “ASIDCacheSize” different ASIDs will be maintained concurrently initially; as new ASIDs are used, TLB entries for less-recently used ASIDs are deleted, which improves model performance in some cases. If the model detects that the TLB entry cache is too small (entry ejections are very frequent), it will increase the cache size automatically. In this variant, “ASIDCacheSize” is 8.

Boolean parameter “ignore\_non\_leaf\_DAU” specifies how non-zero D, A and U fields in non-leaf PTE entries are handled. If “ignore\_non\_leaf\_DAU” is False, then using such entries will trigger Page Fault traps. If “ignore\_non\_leaf\_DAU” is True, then such entries will be handled as if D, A and U fields are all zero. In this variant, “ignore\_non\_leaf\_DAU” is 0.

#### 1.4.14 Unaligned Accesses

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 “Zam” to “T” to enable such accesses.

Address misaligned exceptions are higher priority than page fault or access fault exceptions on this variant. Set parameter “unaligned\_low\_pri” to “T” to specify that they are lower priority instead.

#### 1.4.15 PMP

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.

Accesses to unimplemented PMP registers are write-ignored and read as zero on this variant. Set parameter “PMP\_undefined” to True to indicate that such accesses should cause Illegal Instruction exceptions instead.

#### 1.4.16 Time and Timers

A RISC-V hart requires a time source to be available in any of the following cases:

1. The “time” CSR is implemented (“time\_undefined” is False);
2. The “Sstc” extension is present (“Sstc” is True);
3. The internal CLINT model is enabled (“CLINT\_address” is non-zero).

For cases 1 and 2, a 64-bit input port “mtime” is present. If this port is connected, it must be driven periodically by an external source with the current time value, which is visible in the “time” CSR and used for timer calculations by the “Sstc” extension. If the port is not connected, the value of time is internally derived with a period specified by the “mtime\_Hz” parameter (1e+06Hz by default).

For case 3, time is always internally derived and the “mtime” port is not present.

If the “time” CSR is implemented but the “Sstc” extension and the internal CLINT model are both absent, then it is also possible to implement the “time” CSR using a read callback on the CSR bus instead of using the “mtime” port: this may improve simulation performance if “time” increments at high frequency. See section “CSR Register External Implementation” for more information.

## 1.5 Compressed Extension

Standard compressed instructions are present in this variant. Legacy compressed extension features may also be configured using parameters described below. Use parameter “compress\_version” to enable more recent compressed extension features if required. See the following sections for detailed information about differences between each supported version.

### 1.5.1 Compressed Extension Parameters

Parameter “Zcea\_version” is used to specify the version of Zcea instructions present. By default, “Zcea\_version” is set to “none” in this variant. Updates to this parameter require a commercial product license.

Parameter “Zceb\_version” is used to specify the version of Zceb instructions present. By default, “Zceb\_version” is set to “none” in this variant. Updates to this parameter require a commercial product license.

Parameter “Zcee\_version” is used to specify the version of Zcee instructions present. By default, “Zcee\_version” is set to “none” in this variant. Updates to this parameter require a commercial product license.

### 1.5.2 Legacy Version 1.10

Legacy encodings with version specified using Zcea, Zceb and Zcee parameters.

### 1.5.3 Version 0.70.1

All instruction encodings changed from legacy version, with instructions divided into Zca, Zcf, Zcb, Zcmb, Zcmp, Zcmpe and Zcmt subsets.

### 1.5.4 Version 0.70.5

Version 0.70.5, with these changes compared to version 0.70.1:

- access to jt and jalt instructions is enabled by Smstateen.
- jvt.base is WARL and fewer bits than the maximum can be implemented

### 1.5.5 Version 1.0.0-RC5.7

Version 1.0.0-RC5.7, with these changes compared to version 0.70.5:

- encodings of jt and jalt instructions changed.
- Zcmb and Zcmpe subsets removed.

## 1.6 Privileged Architecture

This variant implements the Privileged Architecture with version specified in the References section of this document. Note that parameter “priv\_version” can be used to select the required architecture version; see the following sections for detailed information about differences between each supported version.

### 1.6.1 Legacy Version 1.10

1.10 version of May 7 2017.

### 1.6.2 Version 20190608

Stable 1.11 version of June 8 2019, with these changes compared to version 1.10:

- mcountinhibit CSR defined;
- pages are never executable in Supervisor mode if page table entry U bit is 1;
- mstatus.TW is writable if any lower-level privilege mode is implemented (previously, it was just if Supervisor mode was implemented);

### 1.6.3 Version 20211203

1.12 draft version of December 3 2021, with these changes compared to version 20190608:

- mstatush, mseccfg, mseccfgh, menvcfg, menvcfgh, senvcfg, henvcfg, henvcfgh and mconfigptr CSRs defined;
- xret instructions clear mstatus.MPRV when leaving Machine mode if new mode is less privileged than M-mode;
- maximum number of PMP registers increased to 64;
- data endian is now configurable.

### 1.6.4 Version 1.12

Official 1.12 version, identical to 20211203.

### 1.6.5 Version master

Unstable master version, currently identical to 1.12.

## 1.7 Unprivileged Architecture

This variant implements the Unprivileged Architecture with version specified in the References section of this document. Note that parameter “user\_version” can be used to select the required architecture version; see the following sections for detailed information about differences between each supported version.

### 1.7.1 Legacy Version 2.2

2.2 version of May 7 2017.

### 1.7.2 Version 20191213

Stable 20191213-Base-Ratified version of December 13 2019, with these changes compared to version 2.2:

- floating point fmin/fmax instruction behavior modified to comply with IEEE 754-201x.
- numerous other optional behaviors can be separately enabled using Z-prefixed parameters.

## 1.8 Other Extensions

Other extensions that can be configured are described in this section.

### 1.8.1 Zmmul

Parameter “Zmmul” is 0 on this variant, meaning that all multiply and divide instructions are implemented. if “Zmmul” is set to 1 then multiply instructions are implemented but divide and remainder instructions are not implemented.

### 1.8.2 Zicsr

Parameter “Zicsr” is 1 on this variant, meaning that standard CSRs and CSR access instructions are implemented. if “Zicsr” is set to 0 then standard CSRs and CSR access instructions are not implemented and an alternative scheme must be provided as a processor extension.

### 1.8.3 Zifencei

Parameter “Zifencei” is 1 on this variant, meaning that the fence.i instruction is implemented (but treated as a NOP by the model). if “Zifencei” is set to 0 then the fence.i instruction is not implemented.

### 1.8.4 Zicbom

Parameter “Zicbom” is 0 on this variant, meaning that code block management instructions are undefined. if “Zicbom” is set to 1 then code block management instructions cbo.clean, cbo.flush and cbo.inval are defined.

If Zicbom is present, the cache block size is given by parameter “cmomp\_bytes”. The instructions may cause traps if used illegally but otherwise are NOPs in this model.

### 1.8.5 Zicbop

Parameter “Zicbop” is 0 on this variant, meaning that prefetch instructions are undefined. if “Zicbop” is set to 1 then prefetch instructions prefetch.i, prefetch.r and prefetch.w are defined (but behave as NOPs in this model).

### 1.8.6 Zicboz

Parameter “Zicboz” is 0 on this variant, meaning that the cbo.zero instruction is undefined. if “Zicboz” is set to 1 then the cbo.zero instruction is defined.

If Zicboz is present, the cache block size is given by parameter “cmoz\_bytes”.

### 1.8.7 Svnapot

Parameter “Svnapot\_page\_mask” is 0x0 on this variant, meaning that NAPOT Translation Contiguity is not implemented. if “Svnapot\_page\_mask” is non-zero then NAPOT Translation Contiguity is enabled for page sizes indicated by that mask value when page table entry bit 63 is set.

If Svnapot is present, “Svnapot\_page\_mask” is a mask of page sizes for which contiguous pages can be created. For example, a value of 0x10000 implies that 64KiB contiguous pages are supported.

### 1.8.8 Svpbmt

Parameter “Svpbmt” is 0 on this variant, meaning that page-based memory types are not implemented. if “Svpbmt” is set to 1 then page-based memory types are indicated by page table entry bits 62:61.

Note that except for their effect on Page Faults, the encoded memory types do not alter the behavior of this model, which always implements strongly-ordered non-cacheable semantics.

### 1.8.9 Svinval

Parameter “Svinval” is 0 on this variant, meaning that fine-grained address-translation cache invalidation instructions are not implemented. if “Svinval” is set to 1 then fine-grained address-translation cache invalidation instructions `sinval.vma`, `sfence.w.inval` and `sfence.inval.ir` are implemented.

### 1.8.10 Smstateen

Parameter “Smstateen” is 0 on this variant, meaning that state enable CSRs are undefined. if “Smstateen” is set to 1 then state enable CSRs are defined.

Within the state enable CSRs, only bit 1 (for `Zfinx`), bit 57 (for `xcontext` CSR access), bit 62 (for `xenvcfg` CSR access) and bit 63 (for lower-level state enable CSR access) are currently implemented.

### 1.8.11 Sstc

Parameter “Sstc” is 0 on this variant, meaning that `stimecmp` is not implemented. if “Sstc” is set to 1 then `stimecmp` is implemented.

### 1.8.12 Zawrs

Parameter “Zawrs” is 0 on this variant, meaning that wait-for-reservation-set instructions are not implemented. if “Zawrs” is set to 1 then wait-for-reservation-set instructions are implemented, in which case parameter “`TW_time_limit`” is used to specify the nominal cycle delay for `wrs.nto`, and parameter “`STO_time_limit`” is used to specify the nominal cycle delay for `wrs.sto`.

## 1.9 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.

## 1.10 Advanced Interrupt Architecture

The model can be configured to implement the Advanced Interrupt Architecture (AIA) interface using Boolean parameter “`Smaia`”; when True, the AIA interface is present as described in the RISC-V Advanced Interrupt Architecture specification, and further parameters are made available to configure other aspects of the interface. “`Smaia`” is False in this variant, indicating that the AIA interface is not implemented.



## 1.11 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; this variant implements a 1-byte reservation granule. 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.

Parameter “amo\_aborts\_lr\_sc” is used to specify whether AMO operations abort any active LR/SC pair. In this variant, “amo\_aborts\_lr\_sc” is 0.

Parameter “lr\_sc\_match\_size” is used to specify whether data sizes of LR and SC instructions must match for the SC instruction to succeed. In this variant, “lr\_sc\_match\_size” is False.

## 1.12 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

## 1.13 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 taken the value on the ID port is sampled and used to fill the Exception Code field in the relevant “xcause” CSR. For Machine External interrupts, the extra interrupt ID port is called “MExternalInterruptID”; for Supervisor External interrupts, the extra interrupt ID port is called “SEExternalInterruptID”.

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.

## 1.14 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.

#### 1.14.1 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.

#### 1.14.2 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.

### 1.14.3 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.

### 1.14.4 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.

### 1.14.5 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.

### 1.14.6 Debug Event Priorities

The model supports three different models for determining which debug exception occurs when step, execute address, `resethaltreq` and `haltreq` events are all pending. These options are listed below, with highest-priority event first:

1. when parameter “debug\_priority”=“sxh”: `step ->execute address ->resethaltreq ->haltreq`;
2. when parameter “debug\_priority”=“shx”: `step ->resethaltreq ->haltreq ->execute address`;
3. when parameter “debug\_priority”=“hsx”: `resethaltreq ->haltreq ->step ->execute address`.

### 1.14.7 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).

### 1.14.8 Debug Mode Versions

Debug mode specification has been under active development. To enable simulation of hardware that may be based on an older version of the specification, the model implements behavior for a number of versions of the specification. The differing features of these are listed below, in chronological order.

#### 1.14.9 Version 0.13.2-DRAFT

0.13.2-DRAFT version of March 22 2019.

#### 1.14.10 Version 0.14.0-DRAFT

0.14.0-DRAFT version of November 6 2020.

#### 1.14.11 Version 1.0.0-STABLE

1.0.0-STABLE version of February 9 2022.

#### 1.14.12 Version 1.0-STABLE

1.0-STABLE version of December 28 2022, with these changes compared to version 1.0.0-STABLE:

- nmi is moved from etrigger to itrigger and is now subject to the mode bits in that trigger.

## 1.15 Debug Mask

It is possible to enable model debug messages in various categories. This can be done statically using the “debugflags” 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.

## 1.16 Integration Support

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

### 1.16.1 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 a read callback at address 0xC010 on the CSR bus.

If both read and write callbacks are installed, or if a read callback is installed and the CSR is in the read-only address space, then the read callback will be used to provide the value for both true accesses and for trace and API register read (using opRegRead, etc). However, if only a read callback is installed and the CSR is in the CSR read/write address space then the callback will be used for true register reads *\*only\**; in this case, the *\*model\** CSR implementation will be used for trace and API register read. This idiom allows values to be injected for volatile CSRs without changing fundamental model behavior.

### 1.16.2 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. When parameter “lr\_sc\_match\_size” is True and this is a 64-bit access, the least-significant bit of “LRSCAddress” is one (to indicate a 64-bit access). If parameter “lr\_sc\_match\_size” is False or this is a 32-bit access, the least-significant bit of “LRSCAddress” is zero.

### 1.16.3 Page Table Walk Introspection

Artifact register “PTWStage” shows the active page table translation stage (0 if no stage active, 1 if HS-stage active, 2 if VS-stage active and 3 if G-stage active). This register is visibly non-zero only in a memory access callback triggered by a page table walk event.

Artifact register “PTWInputAddr” shows the input address of active page table translation. This register is visibly non-zero only in a memory access callback triggered by a page table walk event.

Artifact register “PTWLevel” shows the active level of page table translation (corresponding to index variable “i” in the algorithm described by Virtual Address Translation Process in the RISC-V Privileged Architecture specification). This register is visibly non-zero only in a memory access callback triggered by a page table walk event.

## 1.17 Instruction Disassembly

This model implements a number of parameters to control instruction disassembly, as shown in trace output.

If parameter “use\_hw\_reg\_names” is True, instruction disassembly shows hardware names x0-x31. If “use\_hw\_reg\_names” is False, ABI names are shown instead.

If parameter “no\_pseudo\_inst” is True, instruction disassembly always shows true instructions. If “no\_pseudo\_inst” is False, pseudo-instructions are shown instead where applicable.

If parameter “show\_c\_prefix” is True, instruction disassembly of 16-bit instructions will include a compressed prefix (e.g. “c.” or “cm.”). If “show\_c\_prefix” is False, the compressed prefix will be omitted.

## 1.18 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.

The TLB is architecturally-accurate but not device accurate. This means that all TLB maintenance and address translation operations are fully implemented but the cache is larger than in the real device.

THIS IS A STARTING POINT AS THE SPECS DEVELOP More detail to be added once confirmed

## 1.19 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.

## 1.20 References

The Model details are based upon the following specifications:

RISC-V Instruction Set Manual, Volume I: User-Level ISA (User Architecture Version 20191213)

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



## Chapter 2

# Openhwgroup-Specific Extensions

Open HW Group processors add various custom extensions to the basic RISC-V architecture. This model supports the following CORE-V Instruction Set Extensions:

- PULP\_XPULP Features
- Static 16 entry PMA
- Custom bus fault extensions
- Secure exceptions

The PULP\_CLUSTER and PULP\_ZFINX CORE-V Instruction Set Extensions are not supported, although the standard Risc-V Zfinx extension is supported in the base model.

In addition to the base model RISC-V parameters, this model implements parameters allowing openhwgroup-specific model features to be controlled. These parameters are documented below.

### 2.1 PULP\_XPULP Extension Status

The XPULP extension is only supported for RV32, so is not supported on this variant.

### 2.2 PMA Extension Status

The PMA extension is not supported on this variant.

### 2.3 Custom Bus Fault Extension Status

The Custom Bus Fault extension is not supported on this variant.

## 2.4 Secure Exceptions Status

Secure Exceptions are not supported on this variant.

# Chapter 3

## Configuration

### 3.1 Location

This model's VLNv is [openhwgroup.ovpworld.org/processor/riscv/1.0](https://openhwgroup.org/processor/riscv/1.0).

The model source is usually at:

`$IMPERAS_HOME/ImperasLib/source/openhwgroup.ovpworld.org/processor/riscv/1.0`

The model binary is usually at:

`$IMPERAS_HOME/lib/$IMPERAS_ARCH/ImperasLib/openhwgroup.ovpworld.org/processor/riscv/1.0`

### 3.2 GDB Path

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

### 3.3 Semi-Host Library

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

### 3.4 Processor Endian-ness

This is a LITTLE endian model.

### 3.5 QuantumLeap Support

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

### 3.6 Processor ELF code

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

## Chapter 4

# All Variants in this model

This model has these variants

<b>Variant</b>	Description
CV32E20	
CV32E41P	
CV32A6	
CV64A6	(described in this document)

Table 4.1: All Variants in this model

## Chapter 5

# Bus Master Ports

This model has these bus master ports.

<b>Name</b>	min	max	Connect?	Description
INSTRUCTION	32	64	mandatory	Instruction bus
DATA	32	64	optional	Data bus

Table 5.1: Bus Master Ports

## Chapter 6

# Bus Slave Ports

This model has no bus slave ports.

## Chapter 7

# Net Ports

This model has these net ports.

<b>Name</b>	Type	Connect?	Description
reset	input	optional	Reset
reset_addr	input	optional	Externally-applied reset address
nmi	input	optional	NMI
nmi_cause	input	optional	Externally-applied NMI cause
nmi_addr	input	optional	Externally-applied NMI address
mtime	input	optional	External mtime source
SSWInterrupt	input	optional	Supervisor software interrupt
MSWInterrupt	input	optional	Machine software interrupt
STimerInterrupt	input	optional	Supervisor timer interrupt
MTimerInterrupt	input	optional	Machine timer interrupt
SExternalInterrupt	input	optional	Supervisor external 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
IllegalInstruction	input	optional	Illegal Instruction Exception

Table 7.1: Net Ports

## Chapter 8

# FIFO Ports

This model has no FIFO ports.



## Chapter 9

# Formal Parameters

Name	Type	Description
<b>Fundamental</b>		
variant	Enumeration	Selects variant (either a generic UISA or a specific model)
user_version	Enumeration	Specify required User Architecture version (2.2, 2.3, 20190305 or 20191213)
priv_version	Enumeration	Specify required Privileged Architecture version (1.10, 1.11, 20190405, 20190608, 20211203, 1.12 or master)
endian	Endian	Model endian
enable_expanded	Boolean	Specify that 48-bit and 64-bit expanded instructions are supported
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)
sub_Extensions	String	Remove extensions specified by letters from misa.Extensions (for example, specify “VD” to remove 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)
sub_Extensions_mask	String	Remove extensions specified by letters from mask of writable bits in misa.Extensions (for example, specify “VD” to remove V and D features)
add_implicit_Extensions	String	Add extensions specified by letters to implicitly-present extensions not visible in misa.Extensions
sub_implicit_Extensions	String	Remove extensions specified by letters from implicitly-present extensions not visible in misa.Extensions
<b>Compressed_Extension</b>		
compress_version	Enumeration	Specify required Compressed Architecture version (legacy, 0.70.1, 0.70.5 or 1.0.0-RC5.7)
Zcea_version	Enumeration	Specify version of Zcea implemented (legacy only) (none or 0.50.1)
Zceb_version	Enumeration	Specify version of Zceb implemented (legacy only) (none or 0.50.1)
Zcee_version	Enumeration	Specify version of Zcee implemented (legacy only) (none or 1.0.0-rc)
<b>Interrupts_Exceptions</b>		
rnmi_version	Enumeration	Specify required RNMI Architecture version (none, 0.2.1 or 0.4)
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
nmi_is_latched	Boolean	Specify whether NMI input is latched on rising edge (if False, it is level-sensitive)

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
trap_preserves_lr	Boolean	Whether a trap preserves active LR/SC state
xret_preserves_lr	Boolean	Whether an xret instruction preserves active LR/SC state
reset_address	Uns64	Override reset vector address
nmi_address	Uns64	Override NMI vector address
CLINT_address	Uns64	Specify base address of internal CLINT model (or 0 for no CLINT)
mtime_Hz	Double	Specify clock frequency of time CSR
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
force_sideleg	Uns64	Specify mask of interrupts always delegated to User execution level from Supervisor 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
<b>Debug Extension</b>		
debug_mode	Enumeration	Specify how Debug mode is implemented (none, vector, interrupt or halt)
<b>Memory</b>		
lr_sc_constraint	Enumeration	Specify memory constraint for LR/SC instructions (none, user1 or user2)
amo_constraint	Enumeration	Specify memory constraint for AMO instructions (none, user1 or user2)
updatePTEA	Boolean	Specify whether hardware update of PTE A bit is supported
updatePTED	Boolean	Specify whether hardware update of PTE D bit is supported
unaligned_low_pri	Boolean	Specify whether address misaligned exceptions are lower priority than page or access fault exceptions
unaligned	Boolean	Specify whether the processor supports unaligned memory accesses
Zam	Boolean	Specify whether the processor supports unaligned memory accesses for AMO instructions
amo_aborts_lr_sc	Boolean	Specify whether AMO operations abort any active LR/SC pair
ASID_bits	Uns32	Specify the number of implemented ASID bits
lr_sc_grain	Uns32	Specify byte granularity of LR/SC lock region (constrained to a power of two)
lr_sc_match_size	Boolean	Whether LR/SC access sizes must match
ignore_non_leaf_DAU	Boolean	Whether non-zero D, A and U bits in non-leaf PTEs are ignored (if False, a trap is taken)
Sv_modes	Uns32	Specify bit mask of implemented address translation modes (e.g. (1<<0)+(1<<8) indicates “bare” and “Sv39” modes may be selected in satp.MODE)
<b>Simulation Artifact</b>		
use_hw_reg_names	Boolean	Specify whether to use hardware register names x0-x31 and f0-f31 instead of ABI register names
no_pseudo_inst	Boolean	Specify whether pseudo-instructions should not be reported in trace and disassembly
show_c_prefix	Boolean	Specify whether compressed instruction prefix should be reported in trace and disassembly
verbose	Boolean	Specify verbose output messages
traceVolatile	Boolean	Specify whether volatile registers (e.g. minstret) should be shown in change trace
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 <csr-Name>=<number>
ASID_cache_size	Uns32	Specify the number of different ASIDs for which TLB entries are cached; a value of 0 implies no limit
<b>Instruction_CSR_Behavior</b>		
wfi_is_nop	Boolean	Specify whether WFI should be treated as a NOP (if not, halt while waiting for interrupts)
wfi_resume_not_trap	Boolean	Specify whether pending wakeup events should cause WFI to be treated as a NOP instead of taking a trap
TW_time_limit	Uns32	Specify nominal cycle timeout for instructions controlled by mstatus.TW
counteren_mask	Uns32	Specify hardware-enforced mask of writable bits in mcounteren/scounteren registers
scounteren_zero_mask	Uns32	Specify hardware-enforced mask of always-zero bits in scounteren register
noinhibit_mask	Uns32	Specify hardware-enforced mask of always-zero bits in mcountinhibit register
cycle_undefined	Boolean	Specify that the cycle CSR is undefined
mcycle_undefined	Boolean	Specify that the mcycle CSR is undefined
time_undefined	Boolean	Specify that the time CSR is undefined
instret_undefined	Boolean	Specify that the instret CSR is undefined
minstret_undefined	Boolean	Specify that the minstret CSR is undefined
hpmcounter_undefined	Boolean	Specify that the hpmcounter CSRs are undefined
mhpmcounter_undefined	Boolean	Specify that the mhpmcounter CSRs are undefined
<b>CSR Masks</b>		
mtvec_mask	Uns64	Specify hardware-enforced mask of writable bits in mtvec register
stvec_mask	Uns64	Specify hardware-enforced mask of writable bits in stvec register
mip_mask	Uns64	Specify hardware-enforced mask of writable bits in mip register
sip_mask	Uns64	Specify hardware-enforced mask of writable bits in sip register
mtvec_sext	Boolean	Specify whether mtvec is sign-extended from most-significant bit
stvec_sext	Boolean	Specify whether stvec is sign-extended from most-significant bit
MXL_writable	Boolean	Specify that misa.MXL is writable (feature under development)
SXL_writable	Boolean	Specify that mstatus.SXL is writable (feature under development)
UXL_writable	Boolean	Specify that mstatus.UXL is writable (feature under development)
<b>Trigger</b>		
trigger_num	Uns32	Specify the number of implemented hardware triggers
<b>PMP Configuration</b>		
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
PMP_undefined	Boolean	Whether accesses to unimplemented PMP registers are undefined (if True) or write ignored and zero (if False)
PMP_maskparams	Boolean	Enable parameters to change the read-only masks for PMP CSRs
PMP_initialparams	Boolean	Enable parameters to change the reset values for PMP CSRs
<b>Other Extensions</b>		
Svnapot_page_mask	Uns64	Specify mask of implemented Svnapot intermediate page sizes (e.g. 1<<16 means 64KiB contiguous regions are supported)
Smstateen	Boolean	Specify that Smstateen is implemented
Sstc	Boolean	Specify that Sstc is implemented
Svpbmt	Boolean	Specify that Svpbmt is implemented
Svinval	Boolean	Specify that Svinval is implemented
Zihintntl	Boolean	Specify that Zihintntl is implemented (instruction decode only, implemented as NOP)
Zicond	Boolean	Specify that Zicond is implemented
Zicsr	Boolean	Specify that Zicsr is implemented

Zifencei	Boolean	Specify that Zifencei is implemented
Zicbom	Boolean	Specify that Zicbom is implemented
Zicbop	Boolean	Specify that Zicbop is implemented
Zicboz	Boolean	Specify that Zicboz is implemented
Zawrs	Boolean	Specify that Zawrs is implemented
Zmmul	Boolean	Specify that Zmmul is implemented
<b>CSR Defaults</b>		
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
<b>Floating Point</b>		
mstatus_FS_zero	Boolean	Specify that mstatus.FS is hard-wired to zero
<b>Fast Interrupt</b>		
CLICLEVELS	Uns32	Specify number of interrupt levels implemented by CLIC, or 0 if CLIC absent
<b>AIA Interrupts</b>		
Smaia	Boolean	Specify that Smaia CSRs are present

Table 9.1: Parameters that can be set in: Hart

## 9.1 Extension Parameters

Name	Type	Description
debug	Boolean	debug flags
mcountinhibit_reset	Uns32	reset value of mcountinhibit
tdata1_reset	Uns32	reset value of tdata1
dcsr_reset	Uns32	reset value of dcsr
cpuctrl_reset	Uns32	reset value of cpuctrl
secureseed0_reset	Uns32	reset value of secureseed0
secureseed1_reset	Uns32	reset value of secureseed1
secureseed2_reset	Uns32	reset value of secureseed2

Table 9.2: Parameters for extension

## 9.2 Parameters with enumerated types

### 9.2.1 Parameter user\_version

Set to this value	Description
2.2	User Architecture Version 2.2
2.3	Deprecated and equivalent to 20191213
20190305	Deprecated and equivalent to 20191213
20191213	User Architecture Version 20191213

Table 9.3: Values for Parameter user\_version

### 9.2.2 Parameter priv\_version

Set to this value	Description
-------------------	-------------

1.10	Privileged Architecture Version 1.10
1.11	Privileged Architecture Version 1.11, equivalent to 20190608
20190405	Deprecated and equivalent to 20190608
20190608	Privileged Architecture Version Ratified-IMFDQC-and-Priv-v1.11
20211203	Privileged Architecture Version 20211203
1.12	Privileged Architecture Version 1.12, equivalent to 20211203
master	Privileged Architecture Master Branch as of commit 6bdeb58 (this is subject to change)

Table 9.4: Values for Parameter `priv_version`

### 9.2.3 Parameter `compress_version`

Set to this value	Description
legacy	Compressed Architecture absent or legacy version
0.70.1	Compressed Architecture Version 0.70.1
0.70.5	Compressed Architecture Version 0.70.5
1.0.0-RC5.7	Compressed Architecture Version 1.0.0-RC5.7

Table 9.5: Values for Parameter `compress_version`

### 9.2.4 Parameter `rnmi_version`

Set to this value	Description
none	RNMI not implemented
0.2.1	RNMI version 0.2.1
0.4	RNMI version 0.4

Table 9.6: Values for Parameter `rnmi_version`

### 9.2.5 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 9.7: Values for Parameter `debug_mode`

### 9.2.6 Parameter `lr_sc_constraint`

Set to this value	Description
none	Memory access not constrained
user1	Memory access constrained by MEM_CONSTRAINT_USER1
user2	Memory access constrained by MEM_CONSTRAINT_USER2

Table 9.8: Values for Parameter `lr_sc_constraint`

### 9.2.7 Parameter `amo_constraint`

Set to this value	Description
none	Memory access not constrained

user1	Memory access constrained by MEM.CONSTRAINT.USER1
user2	Memory access constrained by MEM.CONSTRAINT.USER2

Table 9.9: Values for Parameter amo\_constraint

### 9.2.8 Parameter Zcea\_version

Set to this value	Description
none	Zcea not implemented
0.50.1	Zcea version 0.50.1

Table 9.10: Values for Parameter Zcea\_version

### 9.2.9 Parameter Zceb\_version

Set to this value	Description
none	Zceb not implemented
0.50.1	Zceb version 0.50.1

Table 9.11: Values for Parameter Zceb\_version

### 9.2.10 Parameter Zcee\_version

Set to this value	Description
none	Zcee not implemented
1.0.0-rc	Zcee version 1.0.0-rc

Table 9.12: Values for Parameter Zcee\_version

## 9.3 Parameter values

These are the current parameter values.

Name	Value
<b>Fundamental</b>	
variant	CV64A6
user_version	20191213
priv_version	1.10
endian	none
enable_expanded	F
endianFixed	F
misa_MXL	2
misa_Extensions	0x141105
add_Extensions	
sub_Extensions	
misa_Extensions_mask	0
add_Extensions_mask	
sub_Extensions_mask	
add_implicit_Extensions	

sub_implicit_Extensions	
<b>Compressed_Extension</b>	
compress_version	legacy
Zcea_version	none
Zceb_version	none
Zcee_version	none
<b>Interrupts_Exceptions</b>	
rnmi_version	none
mtvec_is_ro	F
tvec_align	0
ecode_mask	0x7fffffffffff
ecode_nmi	0
nmi_is_latched	F
tval_zero	F
tval_zero_ebreak	F
tval_ii_code	F
trap_preserves_lr	F
xret_preserves_lr	F
reset_address	0
nmi_address	0
CLINT_address	0
mtime_Hz	1.000000e+06
local_int_num	0
unimp_int_mask	0
force_mideleg	0
force_sideleg	0
no_ideleg	0
no_e deleg	0
external_int_id	F
<b>Debug_Extension</b>	
debug_mode	none
<b>Memory</b>	
lr_sc_constraint	user1
amo_constraint	user1
updatePTEA	F
updatePTED	F
unaligned_low_pri	F
unaligned	F
Zam	F
amo_aborts_lr_sc	F
ASID_bits	0
lr_sc_grain	1
lr_sc_match_size	F
ignore_non_leaf_DAU	F
Sv_modes	0x701

<b>Simulation Artifact</b>	
use_hw_reg_names	F
no_pseudo_inst	F
show_c_prefix	F
verbose	F
traceVolatile	F
enable_CSR_bus	F
CSR_remap	
ASID_cache_size	8
<b>Instruction_CSR_Behavior</b>	
wfi_is_nop	F
wfi_resume_not_trap	F
TW_time_limit	0
counteren_mask	0
scounteren_zero_mask	0
noinhibit_mask	0
cycle_undefined	F
mcycle_undefined	F
time_undefined	F
instret_undefined	F
minstret_undefined	F
hpmcounter_undefined	F
mhpmcounter_undefined	F
<b>CSR Masks</b>	
mtvec_mask	0
stvec_mask	0
mip_mask	0x337
sip_mask	0x103
mtvec_sext	F
stvec_sext	F
MXL_writable	F
SXL_writable	F
UXL_writable	F
<b>Trigger</b>	
trigger_num	0
<b>PMP Configuration</b>	
PMP_grain	0
PMP_registers	0
PMP_max_page	0
PMP_decompose	F
PMP_undefined	F
PMP_maskparams	F
PMP_initialparams	F
<b>Other Extensions</b>	
Svnapot_page_mask	0



Smstateen	F
Sstc	F
Svpbmt	F
Svinval	F
Zihintntl	F
Zicond	F
Zicshr	T
Zifencei	T
Zicbom	F
Zicbop	F
Zicboz	F
Zawrs	F
Zmmul	F
<b>CSR Defaults</b>	
mvendorid	0
marchid	0
mimpid	0
mhartid	0
mtvec	0
<b>Floating Point</b>	
mstatus_FS_zero	F
<b>Fast Interrupt</b>	
CLICLEVELS	0
<b>AIA Interrupts</b>	
Smaia	F
<b>extension</b>	
debug*	F
mcountinhibit_reset*	13
tdata1_reset*	0
dcsr_reset*	0
cpuctrl_reset*	0
secureseed0_reset*	0
secureseed1_reset*	0
secureseed2_reset*	0

Table 9.13: Parameter values

\* Parameters marked with an asterisk are part of the processor extension library.

## Chapter 10

# Execution Modes

Mode	Code	Description
User	0	User mode
Supervisor	1	Supervisor mode
Machine	3	Machine mode

Table 10.1: Modes implemented in: Hart

# Chapter 11

## 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
EnvironmentCallFromUMode	8	ECALL instruction executed in User mode
EnvironmentCallFromSMode	9	ECALL instruction executed in Supervisor mode
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
SSWInterrupt	65	Supervisor software interrupt
MSWInterrupt	67	Machine software interrupt
STimerInterrupt	69	Supervisor timer interrupt
MTimerInterrupt	71	Machine timer interrupt
SExternalInterrupt	73	Supervisor external interrupt
MExternalInterrupt	75	Machine external interrupt
GenericNMI	4294967295	Generic NMI

Table 11.1: Exceptions implemented in: Hart

## Chapter 12

# 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.

### 12.1 Level 1: Hart

This level in the model hierarchy has 6 commands.

This level in the model hierarchy has 5 register groups:

Group name	Registers
Core	33
User_Control_and_Status	32
Supervisor_Control_and_Status	10
Machine_Control_and_Status	95
Integration_support	6

Table 12.1: Register groups

This level in the model hierarchy has no children.

# Chapter 13

## 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.

### 13.1 Level 1: Hart

#### 13.1.1 debugflags

show or modify the processor debug flags

Argument	Type	Description
-get	Boolean	print current processor flags value
-mask	Boolean	print valid debug flag bits
-set	Int32	new processor flags (only flags 0x00000006 can be modified)

Table 13.1: debugflags command arguments

#### 13.1.2 dumpTLB

##### 13.1.2.1 Argument description

show TLB contents

#### 13.1.3 getCSRIndex

Return index for a named CSR (or -1 if no matching CSR)

Argument	Type	Description
-name	String	CSR name

Table 13.2: getCSRIndex command arguments

#### 13.1.4 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 13.3: isync command arguments

### 13.1.5 itrace

enable or disable instruction tracing

Argument	Type	Description
-access	String	show memory accesses by this instruction. Argument can be any combination of X (execute), A (load or store access) and S (system)
-after	Uns64	apply after this many instructions
-enable	Boolean	enable instruction tracing
-instructioncount	Boolean	include the instruction number in each trace
-memory	String	(Alias for access). show memory accesses by this instruction. Argument can be any combination of X (execute), A (load or store access) and S (system)
-mode	Boolean	show processor mode changes
-off	Boolean	disable instruction tracing
-on	Boolean	enable instruction tracing
-processorname	Boolean	Include processor name in all trace lines
-registerchange	Boolean	show registers changed by this instruction
-registers	Boolean	show registers after each trace

Table 13.4: itrace command arguments

### 13.1.6 listCSRs

#### 13.1.6.1 Argument description

List all CSRs in index order

# Chapter 14

## Registers

### 14.1 Level 1: Hart

#### 14.1.1 Core

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

Name	Bits	Initial-Hex	RW	Description
zero	64	0	r-	
ra	64	0	rw	
sp	64	0	rw	stack pointer
gp	64	0	rw	
tp	64	0	rw	
t0	64	0	rw	
t1	64	0	rw	
t2	64	0	rw	
s0	64	0	rw	
s1	64	0	rw	
a0	64	0	rw	
a1	64	0	rw	
a2	64	0	rw	
a3	64	0	rw	
a4	64	0	rw	
a5	64	0	rw	
a6	64	0	rw	
a7	64	0	rw	
s2	64	0	rw	
s3	64	0	rw	
s4	64	0	rw	
s5	64	0	rw	
s6	64	0	rw	
s7	64	0	rw	
s8	64	0	rw	
s9	64	0	rw	
s10	64	0	rw	
s11	64	0	rw	
t3	64	0	rw	
t4	64	0	rw	
t5	64	0	rw	
t6	64	0	rw	
pc	64	0	rw	program counter

Table 14.1: Registers at level 1, type:Hart group:Core

### 14.1.2 User\_Control\_and\_Status

Registers at level:1, type:Hart group:User\_Control\_and\_Status

Name	Bits	Initial-Hex	RW	Description
cycle	64	0	r-	Cycle Counter
time	64	0	r-	Timer
instret	64	0	r-	Instructions Retired
hpmcounter3	64	0	r-	Performance Monitor Counter 3
hpmcounter4	64	0	r-	Performance Monitor Counter 4
hpmcounter5	64	0	r-	Performance Monitor Counter 5
hpmcounter6	64	0	r-	Performance Monitor Counter 6
hpmcounter7	64	0	r-	Performance Monitor Counter 7
hpmcounter8	64	0	r-	Performance Monitor Counter 8
hpmcounter9	64	0	r-	Performance Monitor Counter 9
hpmcounter10	64	0	r-	Performance Monitor Counter 10
hpmcounter11	64	0	r-	Performance Monitor Counter 11
hpmcounter12	64	0	r-	Performance Monitor Counter 12
hpmcounter13	64	0	r-	Performance Monitor Counter 13
hpmcounter14	64	0	r-	Performance Monitor Counter 14
hpmcounter15	64	0	r-	Performance Monitor Counter 15
hpmcounter16	64	0	r-	Performance Monitor Counter 16
hpmcounter17	64	0	r-	Performance Monitor Counter 17
hpmcounter18	64	0	r-	Performance Monitor Counter 18
hpmcounter19	64	0	r-	Performance Monitor Counter 19
hpmcounter20	64	0	r-	Performance Monitor Counter 20
hpmcounter21	64	0	r-	Performance Monitor Counter 21
hpmcounter22	64	0	r-	Performance Monitor Counter 22
hpmcounter23	64	0	r-	Performance Monitor Counter 23
hpmcounter24	64	0	r-	Performance Monitor Counter 24
hpmcounter25	64	0	r-	Performance Monitor Counter 25
hpmcounter26	64	0	r-	Performance Monitor Counter 26
hpmcounter27	64	0	r-	Performance Monitor Counter 27
hpmcounter28	64	0	r-	Performance Monitor Counter 28
hpmcounter29	64	0	r-	Performance Monitor Counter 29
hpmcounter30	64	0	r-	Performance Monitor Counter 30
hpmcounter31	64	0	r-	Performance Monitor Counter 31

Table 14.2: Registers at level 1, type:Hart group:User\_Control\_and\_Status

### 14.1.3 Supervisor\_Control\_and\_Status

Registers at level:1, type:Hart group:Supervisor\_Control\_and\_Status

Name	Bits	Initial-Hex	RW	Description
sstatus	64	2 00000000	rw	Supervisor Status
sie	64	0	rw	Supervisor Interrupt Enable
stvec	64	0	rw	Supervisor Trap-Vector Base-Address
scounteren	64	0	rw	Supervisor Counter Enable
sscratch	64	0	rw	Supervisor Scratch
sepc	64	0	rw	Supervisor Exception Program Counter
scause	64	0	rw	Supervisor Cause



stval	64	0	rw	Supervisor Trap Value
sip	64	0	rw	Supervisor Interrupt Pending
satp	64	0	rw	Supervisor Address Translation and Protection

Table 14.3: Registers at level 1, type:Hart group:Supervisor\_Control\_and\_Status

#### 14.1.4 Machine\_Control\_and\_Status

Registers at level:1, type:Hart group:Machine\_Control\_and\_Status

Name	Bits	Initial-Hex	RW	Description
mstatus	64	a 00000000	rw	Machine Status
misa	64	80000000 00141105	rw	ISA and Extensions
medeleg	64	0	rw	Machine Exception Delegation
mideleg	64	0	rw	Machine Interrupt Delegation
mie	64	0	rw	Machine Interrupt Enable
mtvec	64	0	rw	Machine Trap-Vector Base-Address
mcounteren	64	0	rw	Machine Counter Enable
mhpmevent3	64	0	rw	Machine Performance Monitor Event Select 3
mhpmevent4	64	0	rw	Machine Performance Monitor Event Select 4
mhpmevent5	64	0	rw	Machine Performance Monitor Event Select 5
mhpmevent6	64	0	rw	Machine Performance Monitor Event Select 6
mhpmevent7	64	0	rw	Machine Performance Monitor Event Select 7
mhpmevent8	64	0	rw	Machine Performance Monitor Event Select 8
mhpmevent9	64	0	rw	Machine Performance Monitor Event Select 9
mhpmevent10	64	0	rw	Machine Performance Monitor Event Select 10
mhpmevent11	64	0	rw	Machine Performance Monitor Event Select 11
mhpmevent12	64	0	rw	Machine Performance Monitor Event Select 12
mhpmevent13	64	0	rw	Machine Performance Monitor Event Select 13
mhpmevent14	64	0	rw	Machine Performance Monitor Event Select 14
mhpmevent15	64	0	rw	Machine Performance Monitor Event Select 15
mhpmevent16	64	0	rw	Machine Performance Monitor Event Select 16
mhpmevent17	64	0	rw	Machine Performance Monitor Event Select 17
mhpmevent18	64	0	rw	Machine Performance Monitor Event Select 18
mhpmevent19	64	0	rw	Machine Performance Monitor Event Select 19
mhpmevent20	64	0	rw	Machine Performance Monitor Event Select 20
mhpmevent21	64	0	rw	Machine Performance Monitor Event Select 21
mhpmevent22	64	0	rw	Machine Performance Monitor Event Select 22
mhpmevent23	64	0	rw	Machine Performance Monitor Event Select 23
mhpmevent24	64	0	rw	Machine Performance Monitor Event Select 24
mhpmevent25	64	0	rw	Machine Performance Monitor Event Select 25
mhpmevent26	64	0	rw	Machine Performance Monitor Event Select 26
mhpmevent27	64	0	rw	Machine Performance Monitor Event Select 27
mhpmevent28	64	0	rw	Machine Performance Monitor Event Select 28
mhpmevent29	64	0	rw	Machine Performance Monitor Event Select 29
mhpmevent30	64	0	rw	Machine Performance Monitor Event Select 30
mhpmevent31	64	0	rw	Machine Performance Monitor Event Select 31
mscratch	64	0	rw	Machine Scratch
mepc	64	0	rw	Machine Exception Program Counter
mcause	64	0	rw	Machine Cause
mtval	64	0	rw	Machine Trap Value
mip	64	0	rw	Machine Interrupt Pending
pmpcfg0	64	0	rw	Physical Memory Protection Configuration 0
pmpcfg2	64	0	rw	Physical Memory Protection Configuration 2
pmpaddr0	64	0	rw	Physical Memory Protection Address 0

pmpaddr1	64	0	rw	Physical Memory Protection Address 1
pmpaddr2	64	0	rw	Physical Memory Protection Address 2
pmpaddr3	64	0	rw	Physical Memory Protection Address 3
pmpaddr4	64	0	rw	Physical Memory Protection Address 4
pmpaddr5	64	0	rw	Physical Memory Protection Address 5
pmpaddr6	64	0	rw	Physical Memory Protection Address 6
pmpaddr7	64	0	rw	Physical Memory Protection Address 7
pmpaddr8	64	0	rw	Physical Memory Protection Address 8
pmpaddr9	64	0	rw	Physical Memory Protection Address 9
pmpaddr10	64	0	rw	Physical Memory Protection Address 10
pmpaddr11	64	0	rw	Physical Memory Protection Address 11
pmpaddr12	64	0	rw	Physical Memory Protection Address 12
pmpaddr13	64	0	rw	Physical Memory Protection Address 13
pmpaddr14	64	0	rw	Physical Memory Protection Address 14
pmpaddr15	64	0	rw	Physical Memory Protection Address 15
tcontrol*	64	0	rw	Trigger Control
mcycle	64	0	rw	Machine Cycle Counter
minstret	64	0	rw	Machine Instructions Retired
mhpmcounter3	64	0	rw	Machine Performance Monitor Counter 3
mhpmcounter4	64	0	rw	Machine Performance Monitor Counter 4
mhpmcounter5	64	0	rw	Machine Performance Monitor Counter 5
mhpmcounter6	64	0	rw	Machine Performance Monitor Counter 6
mhpmcounter7	64	0	rw	Machine Performance Monitor Counter 7
mhpmcounter8	64	0	rw	Machine Performance Monitor Counter 8
mhpmcounter9	64	0	rw	Machine Performance Monitor Counter 9
mhpmcounter10	64	0	rw	Machine Performance Monitor Counter 10
mhpmcounter11	64	0	rw	Machine Performance Monitor Counter 11
mhpmcounter12	64	0	rw	Machine Performance Monitor Counter 12
mhpmcounter13	64	0	rw	Machine Performance Monitor Counter 13
mhpmcounter14	64	0	rw	Machine Performance Monitor Counter 14
mhpmcounter15	64	0	rw	Machine Performance Monitor Counter 15
mhpmcounter16	64	0	rw	Machine Performance Monitor Counter 16
mhpmcounter17	64	0	rw	Machine Performance Monitor Counter 17
mhpmcounter18	64	0	rw	Machine Performance Monitor Counter 18
mhpmcounter19	64	0	rw	Machine Performance Monitor Counter 19
mhpmcounter20	64	0	rw	Machine Performance Monitor Counter 20
mhpmcounter21	64	0	rw	Machine Performance Monitor Counter 21
mhpmcounter22	64	0	rw	Machine Performance Monitor Counter 22
mhpmcounter23	64	0	rw	Machine Performance Monitor Counter 23
mhpmcounter24	64	0	rw	Machine Performance Monitor Counter 24
mhpmcounter25	64	0	rw	Machine Performance Monitor Counter 25
mhpmcounter26	64	0	rw	Machine Performance Monitor Counter 26
mhpmcounter27	64	0	rw	Machine Performance Monitor Counter 27
mhpmcounter28	64	0	rw	Machine Performance Monitor Counter 28
mhpmcounter29	64	0	rw	Machine Performance Monitor Counter 29
mhpmcounter30	64	0	rw	Machine Performance Monitor Counter 30
mhpmcounter31	64	0	rw	Machine Performance Monitor Counter 31
mvendorid	64	0	r-	Vendor ID
marchid	64	0	r-	Architecture ID
mimpid	64	0	r-	Implementation ID
mhartid	64	0	r-	Hardware Thread ID

Table 14.4: Registers at level 1, type:Hart group:Machine\_Control\_and\_Status

\* Registers marked with an asterisk are part of the processor extension library.

### 14.1.5 Integration\_support

Registers at level:1, type:Hart group:Integration\_support

Name	Bits	Initial-Hex	RW	Description
LRSCAddress	64	ffffff fffffff	rw	LR/SC active lock address
commercial	8	0	r-	Commercial feature in use
PTWStage	8	0	r-	PTW active stage (0:none 1:HS 2:VS 3:G)
PTWInputAddr	64	0	r-	PTW input address
PTWLevel	8	0	r-	PTW active level
ASYNCPPE	8	0	r-	Asynchronous Event Pending & Enabled

Table 14.5: Registers at level 1, type:Hart group:Integration\_support