



## OVP Guide to Using Processor Models

### Model specific information for RISC-V\_RV64X

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	20211118.0
Filename	OVP_Model_Specific_Information_imperas_riscv_RV64X.pdf
Created	31 December 2021
Status	RESTRICTED - only under NDA

## Copyright Notice

Copyright (c) 2021 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 available under a RESTRICTED license. This is normally because the model is proprietary to a specific IP vendor and requires their formal written approval to release this model. A Software License Agreement with confidentiality clauses is required. You must have written approval from Imperas to access and use this model.

# 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
1.3.2	Enabling Other Extensions	2
1.3.3	Disabling Extensions	3
1.4	General Features	3
1.4.1	mtvec CSR	3
1.4.2	Reset	4
1.4.3	NMI	4
1.4.4	WFI	4
1.4.5	cycle CSR	4
1.4.6	time CSR	4
1.4.7	instret CSR	5
1.4.8	hpmcounter CSRs	5
1.4.9	Unaligned Accesses	5
1.4.10	PMP	5
1.4.11	LR/SC Granule	5
1.5	Compressed Extension	5
1.6	Floating Point Features	6
1.7	Privileged Architecture	6
1.7.1	Legacy Version 1.10	6
1.7.2	Version 20190608	6
1.7.3	Version master	7
1.8	Unprivileged Architecture	7
1.8.1	Legacy Version 2.2	7
1.8.2	Version 20191213	7
1.9	Other Extensions	7
1.9.1	Zmmul	8
1.9.2	Zicsr	8
1.9.3	Zifencei	8
1.9.4	Zicbom	8
1.9.5	Zicbop	8
1.9.6	Zicboz	8
1.10	CLIC	8
1.11	Load-Reserved/Store-Conditional Locking	9

1.12	Active Atomic Operation Indication . . . . .	9
1.13	Interrupts . . . . .	10
1.14	Debug Mode . . . . .	10
1.14.1	Debug State Entry . . . . .	11
1.14.2	Debug State Exit . . . . .	11
1.14.3	Debug Registers . . . . .	12
1.14.4	Debug Mode Execution . . . . .	12
1.14.5	Debug Single Step . . . . .	12
1.14.6	Debug Ports . . . . .	12
1.15	Debug Mask . . . . .	13
1.16	Integration Support . . . . .	13
1.16.1	CSR Register External Implementation . . . . .	13
1.16.2	LR/SC Active Address . . . . .	13
1.16.3	Artifact Register “fflags.i” . . . . .	13
1.17	Limitations . . . . .	13
1.18	Verification . . . . .	14
1.19	References . . . . .	14
<b>2</b>	<b>Instruction Extensions</b>	<b>15</b>
2.1	Custom Instructions . . . . .	15
2.1.1	xend . . . . .	15
2.1.2	xbegin . . . . .	15
2.1.3	xabort . . . . .	16
2.1.4	wfe . . . . .	16
<b>3</b>	<b>Instruction Extensions</b>	<b>17</b>
3.1	Custom Instructions . . . . .	17
3.1.1	pushb . . . . .	17
3.1.2	popb . . . . .	17
<b>4</b>	<b>Configuration</b>	<b>18</b>
4.1	Location . . . . .	18
4.2	GDB Path . . . . .	18
4.3	Semi-Host Library . . . . .	18
4.4	Processor Endian-ness . . . . .	18
4.5	QuantumLeap Support . . . . .	18
4.6	Processor ELF code . . . . .	18
<b>5</b>	<b>All Variants in this model</b>	<b>19</b>
<b>6</b>	<b>Bus Master Ports</b>	<b>20</b>
<b>7</b>	<b>Bus Slave Ports</b>	<b>21</b>
<b>8</b>	<b>Net Ports</b>	<b>22</b>
<b>9</b>	<b>FIFO Ports</b>	<b>23</b>
<b>10</b>	<b>Formal Parameters</b>	<b>24</b>

10.1	Extension Parameters . . . . .	26
10.2	Parameters with enumerated types . . . . .	27
10.2.1	Parameter user_version . . . . .	27
10.2.2	Parameter priv_version . . . . .	27
10.2.3	Parameter rnmi_version . . . . .	27
10.2.4	Parameter mstatus_fs_mode . . . . .	27
10.2.5	Parameter debug_mode . . . . .	27
10.2.6	Parameter Zfinx_version . . . . .	27
10.2.7	Parameter Zcea_version . . . . .	28
10.2.8	Parameter Zceb_version . . . . .	28
10.2.9	Parameter Zcee_version . . . . .	28
10.3	Parameter values . . . . .	28
<b>11</b>	<b>Execution Modes</b>	<b>31</b>
<b>12</b>	<b>Exceptions</b>	<b>32</b>
<b>13</b>	<b>Hierarchy of the model</b>	<b>33</b>
13.1	Level 1: Hart . . . . .	33
<b>14</b>	<b>Model Commands</b>	<b>34</b>
14.1	Level 1: Hart . . . . .	34
14.1.1	printCustomRegs . . . . .	34
14.1.1.1	Argument description . . . . .	34
14.1.2	getCSRIndex . . . . .	34
14.1.3	isync . . . . .	34
14.1.4	itrace . . . . .	34
14.1.5	listCSRs . . . . .	35
14.1.5.1	Argument description . . . . .	35
14.1.6	diagnostic . . . . .	35
<b>15</b>	<b>Registers</b>	<b>36</b>
15.1	Level 1: Hart . . . . .	36
15.1.1	Core . . . . .	36
15.1.2	Floating_point . . . . .	37
15.1.3	User_Control_and_Status . . . . .	37
15.1.4	Machine_Control_and_Status . . . . .	38
15.1.5	Integration_support . . . . .	40

# 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 RV64X 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 3: extension D (double-precision floating point)

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

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

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

`misa` bit 20: extension U (User mode)

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

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.

### 1.3.2 Enabling Other Extensions

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 4: RV32E base integer instruction set (embedded)

`misa` bit 7: extension H (hypervisor)

`misa` bit 10: extension K (cryptographic)

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

`misa` bit 15: extension P (DSP instructions)

`misa` bit 18: extension S (Supervisor mode)

`misa` bit 21: extension V (vector extension)

To add features from this list to the visible set in the misa register, use parameter “add\_Extensions”. This is a string containing identification letters of features to enable; 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.

To add features from this list to the implicitly-enabled set (not visible in the misa register), use parameter “add\_implicit\_Extensions”. This is a string parameter in the same format as the “add\_Extensions” parameter described above.

### 1.3.3 Disabling Extensions

The following extensions are enabled by default in the model and can be disabled:

misa bit 0: extension A (atomic instructions)

misa bit 2: extension C (compressed instructions)

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

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

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

misa bit 20: extension U (User mode)

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

To disable features that are enabled by default, use parameter “sub\_Extensions”. This is a string containing identification letters of features to disable; for example, value “DF” indicates that double-precision and single-precision floating point extensions should be disabled, if they are enabled by default on this variant.

To remove features from this list from the implicitly-enabled set (not visible in the misa register), use parameter “sub\_implicit\_Extensions”. This is a string parameter in the same format as the “sub\_Extensions” parameter described above.

## 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 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.

### 1.4.3 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”.

### 1.4.4 WFI

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).

### 1.4.5 cycle CSR

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 cause Illegal Instruction traps.

### 1.4.6 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 notes below about the artifact “CSR” bus for information about how this is done.

### 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 reads of it should cause Illegal Instruction traps.

### 1.4.8 hpmcounter CSRs

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

### 1.4.9 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 “unalignedAMO” to “T” to enable such accesses.

### 1.4.10 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.

### 1.4.11 LR/SC Granule

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 Compressed Extension

Standard compressed instructions are present in this variant.

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.

## 1.6 Floating Point Features

Half precision floating point is not implemented. Use parameter “Zfh” to enable this if required.

By default, the processor starts with floating-point instructions disabled (`mstatus.FS=0`). Use parameter “`mstatus_FS`” to force `mstatus.FS` to a non-zero value for floating-point to be enabled from the start.

The specification is imprecise regarding the conditions under which `mstatus.FS` is set to Dirty state (3). Parameter “`mstatus_fs_mode`” can be used to specify the required behavior in this model, as described below.

If “`mstatus_fs_mode`” is set to “`always_dirty`” then the model implements a simplified floating point status view in which `mstatus.FS` holds values 0 (Off) and 3 (Dirty) only; any write of values 1 (Initial) or 2 (Clean) from privileged code behave as if value 3 was written.

If “`mstatus_fs_mode`” is set to “`write_1`” then `mstatus.FS` will be set to 3 (Dirty) by any explicit write to the `fflags`, `frm` or `fcsr` control registers, or by any executed instruction that writes an FPR, or by any executed floating point compare or conversion to integer/unsigned that signals a floating point exception. Floating point compare or conversion to integer/unsigned instructions that do not signal an exception will not set `mstatus.FS`.

If “`mstatus_fs_mode`” is set to “`write_any`” then `mstatus.FS` will be set to 3 (Dirty) by any explicit write to the `fflags`, `frm` or `fcsr` control registers, or by any executed instruction that writes an FPR, or by any executed floating point compare or conversion even if those instructions do not signal a floating point exception.

In this variant, “`mstatus_fs_mode`” is set to “`write_1`”.

## 1.7 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.

### 1.7.1 Legacy Version 1.10

1.10 version of May 7 2017.

### 1.7.2 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);

### 1.7.3 Version master

Unstable master version corresponding to evolving 1.12 specification, 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.

## 1.8 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.

### 1.8.1 Legacy Version 2.2

2.2 version of May 7 2017.

### 1.8.2 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.

## 1.9 Other Extensions

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

### 1.9.1 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.

### 1.9.2 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.

### 1.9.3 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.

### 1.9.4 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.

### 1.9.5 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).

### 1.9.6 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”.

## 1.10 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.

## 1.11 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. 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.

## 1.12 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 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.

## 1.14 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 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).

## 1.15 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.

## 1.16 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 callbacks at address 0xC010 on the CSR bus.

### 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.

### 1.16.3 Artifact Register “fflags\_i”

If parameter “enable\_fflags\_i” is True, an 8-bit artifact register “fflags\_i” is added to the model. This register shows the floating point flags set by the current instruction (unlike the standard “fflags” CSR, in which the flag bits are sticky).

## 1.17 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.

## 1.18 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.

## 1.19 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 Ratified-IMFDQC-and-Priv-v1.11)

## Chapter 2

# Instruction Extensions

RISCV processors may add various custom extensions to the basic RISC-V architecture. This processor has been extended, using an extension library, to add Transactional Memory instructions to the Custom0 opcode space.

To enable this extension the read-only CSR register “tm\_cfg” must be initialized to the value 1.

### 2.1 Custom Instructions

This model includes the following custom instructions.

#### 2.1.1 xend

31	25	24	20	19	15	14	12	11	7	6	0
0000000		00000		00000		010		r		Custom0	
										0001011	

end transaction

The xend instruction will terminate an active transaction, committing the cached memory values to memory.

If no transaction is currently active the xabort instruction is a nop.

#### 2.1.2 xbegin

31	25	24	20	19	15	14	12	11	7	6	0
0000000		00000		00000		011		00000		Custom0	
										0001011	

begin transaction

The value returned in the rd register by the xbegin instruction will be 0 when the transaction has successfully started.

If the transaction subsequently aborts, a jump to the instruction following the xbegin instruction

with the register state of the processor restored to the values when the xbegin instruction was executed but with the following bits set in the rd register indicating the cause(s) for the abort:

Bit 0: a memory conflict occurred with another transaction.

Bit 1: another xbegin instruction was executed while this transaction was active.

Bit 2: an xabort instruction executed during transaction (See bits 8:15).

Bit 3: an exception or return from exception occurred during the transaction.

Bit 4: TM buffer overflow - too many cache lines needed.

Bits 8:15: if bit 2 is set then bits 8:15 = bits 7:0 of value passed to xabort, otherwise 0.

### 2.1.3 xabort

31	25	24	20	19	15	14	12	11	7	6	0
0000000		00000		00000		100		00000		Custom0 0001011	

abort transaction

The xabort transaction instruction will cause a current transaction to abort. The code returned to the aborted xbegin instruction will have bit 2 set and bits 7:15 of the code will contain bits 0:7 of the rsrc register specified on the xabort instruction.

If no transaction is currently active the xabort instruction is a nop.

### 2.1.4 wfe

31	25	24	20	19	15	14	12	11	7	6	0
0000000		00000		00000		10		00000		Custom0 0001011	

wait for event hint

The wfe instruction is a hint that the processor is waiting for an external event before it can proceed. In the simulator, this tells the model to suspend execution until the end of the current simulated quantum of instructions for this processor. (This is the equivalent of executing NOP instructions for the remainder of the quantum, but is more efficient for the simulator.)

Hardware implementations might put the processor in a low power mode until an interrupt or other signal occurs.

## Chapter 3

# Instruction Extensions

RISCV processors may add various custom extensions to the basic RISC-V architecture. This processor has been extended, using an extension library, to add FIFO instructions to the Custom0 opcode space, and FIFO ports.

### 3.1 Custom Instructions

This model includes two FIFO instructions Note: these are blocking instructions.

#### 3.1.1 pushb

31	25	24	20	19	15	14	12	11	7	6	0
0000000		00000		00000		000		Rdata		Custom0 0001011	

push(Rdata) - block if full

#### 3.1.2 popb

31	25	24	20	19	15	14	12	11	7	6	0
0000000		00000		00000		001		Rdata		Custom0 0001011	

pop(Rdata) - block if empty

# Chapter 4

## Configuration

### 4.1 Location

This model's VLVN is `vendor.com/processor/riscv/1.0`.

The model source is usually at:

`$IMPERAS_HOME/ImperasLib/source/vendor.com/processor/riscv/1.0`

The model binary is usually at:

`$IMPERAS_HOME/lib/$IMPERAS_ARCH/ImperasLib/vendor.com/processor/riscv/1.0`

### 4.2 GDB Path

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

### 4.3 Semi-Host Library

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

### 4.4 Processor Endian-ness

This is a LITTLE endian model.

### 4.5 QuantumLeap Support

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

### 4.6 Processor ELF code

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

## Chapter 5

# All Variants in this model

This model has these variants

Variant	Description
RV32X	
RV64X	(described in this document)

Table 5.1: All Variants in this model



## Chapter 6

# 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 6.1: Bus Master Ports

## Chapter 7

# Bus Slave Ports

This model has no bus slave ports.

## Chapter 8

# 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
MSWInterrupt	input	optional	Machine software interrupt
MTimerInterrupt	input	optional	Machine timer interrupt
MExternalInterrupt	input	optional	Machine external interrupt
LocalInterrupt5	input	optional	Local interrupt 5
LocalInterrupt6	input	optional	Local interrupt 6
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 8.1: Net Ports

## Chapter 9

# FIFO Ports

This model has these FIFO ports.

<b>Name</b>	Type	Bit width	Connect?	Description
fifoPortIn	input	32	optional	
fifoPortOut	output	32	optional	

Table 9.1: FIFO Ports

# Chapter 10

## 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 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
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
Zmmul	Boolean	Specify that Zmmul is implemented
<b>Interrupts.Exceptions</b>		
rnmi_version	Enumeration	Specify required RNMI Architecture version (none or 0.2.1)
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
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)
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_e deleg	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>Floating Point</b>		
mstatus_fs_mode	Enumeration	Specify conditions causing update of mstatus.FS to dirty (write_1, write_any or always_dirty)
d_requires_f	Boolean	If D and F extensions are separately enabled in the misa CSR, whether D is enabled only if F is enabled
enable_fflags_i	Boolean	Whether fflags.i artifact register present (shows per-instruction floating point flags)
mstatus_FS	Uns32	Override default value of mstatus.FS (initial state of floating point unit)
Zfh	Boolean	Specify that Zfh is implemented (IEEE half-precision floating point is supported)
Zfhmin	Boolean	Specify that Zfhmin is implemented (restricted IEEE half-precision floating point is supported)
Zfinx_version	Enumeration	Specify version of Zfinx implemented (use integer register file for floating point instructions) (none, 0.4 or 0.41)
<b>Debug</b>		
debug_mode	Enumeration	Specify how Debug mode is implemented (none, vector, interrupt or halt)
<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
ABI_d	Boolean	Specify whether D registers are used for parameters (ABI SemiHosting)
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>
<b>Memory</b>		
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

<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)
counteren_mask	Uns32	Specify hardware-enforced mask of writable bits in mcounteren/scouteren 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
time_undefined	Boolean	Specify that the time CSR is undefined
instret_undefined	Boolean	Specify that the instret CSR is undefined
hpmcounter_undefined	Boolean	Specify that the hpmcounter CSRs are undefined
<b>CSR_Masks</b>		
mtvec_mask	Uns64	Specify hardware-enforced mask of writable bits in mtvec register
mip_mask	Uns64	Specify hardware-enforced mask of writable bits in mip register
mtvec_sext	Boolean	Specify whether mtvec is sign-extended from most-significant bit
MXL_writable	Boolean	Specify that misa.MXL 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>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>Compressed</b>		
Zcea_version	Enumeration	Specify version of Zcea implemented (code-size reduction extension) (none or 0.50.1)
Zceb_version	Enumeration	Specify version of Zceb implemented (code-size reduction extension) (none or 0.50.1)
Zcee_version	Enumeration	Specify version of Zcee implemented (code-size reduction extension) (none or 1.0.0-rc)
<b>Fast Interrupt</b>		
CLICLEVELS	Uns32	Specify number of interrupt levels implemented by CLIC, or 0 if CLIC absent

Table 10.1: Parameters that can be set in: Hart

## 10.1 Extension Parameters

Name	Type	Description
FIFO_bits	Uns32	Specify FIFO element bits

Table 10.2: Parameters for fifoExtensions

Name	Type	Description
diagnosticlevel	Uns32	Override the initial diagnostic level
variant	Uns32	Override the configured variant

Table 10.3: Parameters for tmExtensions

## 10.2 Parameters with enumerated types

### 10.2.1 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 10.4: Values for Parameter user\_version

### 10.2.2 Parameter priv\_version

Set to this value	Description
1.10	Privileged Architecture Version 1.10
1.11	Deprecated and equivalent to 20190608
20190405	Deprecated and equivalent to 20190608
20190608	Privileged Architecture Version Ratified-IMFDQC-and-Priv-v1.11
master	Privileged Architecture Master Branch (1.12 draft)

Table 10.5: Values for Parameter priv\_version

### 10.2.3 Parameter rnmi\_version

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

Table 10.6: Values for Parameter rnmi\_version

### 10.2.4 Parameter mstatus\_fs\_mode

Set to this value	Description
write_1	Any non-zero flag result sets mstatus.fs dirty
write_any	Any write of flags sets mstatus.fs dirty
always_dirty	mstatus.fs is either off or dirty

Table 10.7: Values for Parameter mstatus\_fs\_mode

### 10.2.5 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 10.8: Values for Parameter debug\_mode

### 10.2.6 Parameter Zfinx\_version



Set to this value	Description
none	Zfinx not implemented
0.4	Zfinx version 0.4
0.41	Zfinx version 0.41

Table 10.9: Values for Parameter Zfinx\_version

### 10.2.7 Parameter Zcea\_version

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

Table 10.10: Values for Parameter Zcea\_version

### 10.2.8 Parameter Zceb\_version

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

Table 10.11: Values for Parameter Zceb\_version

### 10.2.9 Parameter Zcee\_version

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

Table 10.12: Values for Parameter Zcee\_version

## 10.3 Parameter values

These are the current parameter values.

Name	Value
<b>Fundamental</b>	
variant	RV64X
user_version	20191213
priv_version	20190608
endian	none
enable_expanded	F
endianFixed	F
misa_MXL	2
misa_Extensions	0x90112d
add_Extensions	
sub_Extensions	
misa_Extensions_mask	0
add_Extensions_mask	

sub_Extensions_mask	
add_implicit_Extensions	
sub_implicit_Extensions	
Zicsr	T
Zifencei	T
Zicbom	F
Zicbop	F
Zicboz	F
Zmmul	F
<b>Interrupts_Exceptions</b>	
rnmi_version	none
mtvec_is_ro	F
tvec_align	0
ecode_mask	0x7fffffffffff
ecode_nmi	0
tval_zero	F
tval_zero_ebreak	F
tval_ii_code	T
trap_preserves_lr	F
xret_preserves_lr	F
reset_address	0
nmi_address	0
CLINT_address	0
local_int_num	7
unimp_int_mask	0x1f0000
force_mideleg	0
no_ideleg	0
no_e deleg	0
external_int_id	F
<b>Floating_Point</b>	
mstatus_fs_mode	write_1
d_requires_f	F
enable_fflags_i	F
mstatus_FS	0
Zfh	F
Zfhmin	F
Zfinx_version	none
<b>Debug</b>	
debug_mode	none
<b>Simulation_Artifact</b>	
use_hw_reg_names	F
ABI_d	T
verbose	F
traceVolatile	F
enable_CSR_bus	F

CSR_remap	
<b>Memory</b>	
unaligned	F
unalignedAMO	F
lr_sc_grain	1
PMP_grain	0
PMP_registers	0
PMP_max_page	0
PMP_decompose	F
<b>Instruction_CSR_Behavior</b>	
wfi_is_nop	F
counteren_mask	0
noinhibit_mask	0
cycle_undefined	F
time_undefined	F
instret_undefined	F
hpmcounter_undefined	F
<b>CSR_Masks</b>	
mtvec_mask	0
mip_mask	0x337
mtvec_sext	F
MXL_writable	F
UXL_writable	F
<b>Trigger</b>	
trigger_num	0
<b>CSR_Defaults</b>	
mvendorid	0
marchid	0
mimpid	0
mhartid	0
mtvec	0
<b>Compressed</b>	
Zcea_version	none
Zceb_version	none
Zcee_version	none
<b>Fast Interrupt</b>	
CLICLEVELS	0
<b>fifoExtensions</b>	
FIFO_bits*	32
<b>tmExtensions</b>	
diagnosticlevel*	0
variant*	

Table 10.13: Parameter values

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

## Chapter 11

# Execution Modes

Mode	Code	Description
User	0	User mode
Machine	3	Machine mode

Table 11.1: Modes implemented in: Hart

## Chapter 12

# 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
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
EXCEPT24	24	Custom Exception 24
MSWInterrupt	67	Machine software interrupt
MTimerInterrupt	71	Machine timer interrupt
MExternalInterrupt	75	Machine external interrupt
LocalInterrupt5	85	Local interrupt 5
LocalInterrupt6	86	Local interrupt 6

Table 12.1: Exceptions implemented in: Hart

## Chapter 13

# 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.

### 13.1 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
Floating_point	32
User_Control_and_Status	35
Machine_Control_and_Status	100
Integration_support	2

Table 13.1: Register groups

This level in the model hierarchy has no children.

# Chapter 14

## 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.

### 14.1 Level 1: Hart

#### 14.1.1 printCustomRegs

##### 14.1.1.1 Argument description

Show values of all custom registers

#### 14.1.2 getCSRIndex

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

Argument	Type	Description
-name	String	CSR name

Table 14.1: getCSRIndex command arguments

#### 14.1.3 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 14.2: isync command arguments

#### 14.1.4 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
-memory	String	show memory accesses by this instruction. Argument can be any combination of X (execute), L (load or store access) and S (system)
-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 14.3: itrace command arguments

### 14.1.5 listCSRs

#### 14.1.5.1 Argument description

List all CSRs in index order

### 14.1.6 diagnostic

Set how much additional information is reported for the library

Argument	Type	Description
-level	Int32	Higher numbers print more diagnostic information 0 = off (initial) 1 = startup and shutdown 2 = changes of major modes, infrequent commands 3 = full noisy

Table 14.4: diagnostic command arguments



# Chapter 15

## Registers

### 15.1 Level 1: Hart

#### 15.1.1 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 15.1: Registers at level 1, type:Hart group:Core

### 15.1.2 Floating\_point

Registers at level:1, type:Hart group:Floating\_point

Name	Bits	Initial-Hex	RW	Description
ft0	64	0	rw	
ft1	64	0	rw	
ft2	64	0	rw	
ft3	64	0	rw	
ft4	64	0	rw	
ft5	64	0	rw	
ft6	64	0	rw	
ft7	64	0	rw	
fs0	64	0	rw	
fs1	64	0	rw	
fa0	64	0	rw	
fa1	64	0	rw	
fa2	64	0	rw	
fa3	64	0	rw	
fa4	64	0	rw	
fa5	64	0	rw	
fa6	64	0	rw	
fa7	64	0	rw	
fs2	64	0	rw	
fs3	64	0	rw	
fs4	64	0	rw	
fs5	64	0	rw	
fs6	64	0	rw	
fs7	64	0	rw	
fs8	64	0	rw	
fs9	64	0	rw	
fs10	64	0	rw	
fs11	64	0	rw	
ft8	64	0	rw	
ft9	64	0	rw	
ft10	64	0	rw	
ft11	64	0	rw	

Table 15.2: Registers at level 1, type:Hart group:Floating\_point

### 15.1.3 User\_Control\_and\_Status

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

Name	Bits	Initial-Hex	RW	Description
fflags	64	0	rw	Floating-Point Flags
frm	64	0	rw	Floating-Point Rounding Mode
fcsr	64	0	rw	Floating-Point Control and Status
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 15.3: Registers at level 1, type:Hart group:User\_Control\_and\_Status

#### 15.1.4 Machine\_Control\_and\_Status

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

Name	Bits	Initial-Hex	RW	Description
mstatus*	64	2 00000000	rw	Machine Status
misa	64	80000000 0090112d	rw	ISA and Extensions
mie	64	0	rw	Machine Interrupt Enable
mtvec	64	0	rw	Machine Trap-Vector Base-Address
mcounteren	64	0	rw	Machine Counter Enable
mcounthinhibit	64	0	rw	Machine Counter Inhibit
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
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
custom_rw1_32*	64	0	rw	32-bit R/W CSR (plain)
custom_rw2_64*	64	34	rw	XLEN R/W CSR (plain)
custom_rw3_32*	64	0	rw	32-bit R/W CSR (cb)
custom_rw4_64*	64	0	rw	XLEN R/W CSR (cb)
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
custom_ro1*	64	12345678	r-	R/O CSR (plain)
tm_cfg*	64	1	r-	TM Configuration
fifo_cfg*	64	1	r-	FIFO Configuration

Table 15.4: Registers at level 1, type:Hart group:Machine\_Control\_and\_Status

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

### 15.1.5 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

Table 15.5: Registers at level 1, type:Hart group:Integration\_support