



## OVP Guide to Using Processor Models

### Model specific information for RISC-V\_RVB64I

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_riscv_RVB64I.pdf
Created	31 December 2021
Status	OVP Standard Release

## 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 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
1.3.2	Enabling Other Extensions	2
1.4	General Features	3
1.4.1	mtvec CSR	3
1.4.2	Reset	3
1.4.3	NMI	3
1.4.4	WFI	4
1.4.5	cycle CSR	4
1.4.6	time CSR	4
1.4.7	instret CSR	4
1.4.8	hpmcounter CSRs	4
1.4.9	Unaligned Accesses	4
1.4.10	PMP	4
1.5	Privileged Architecture	5
1.5.1	Legacy Version 1.10	5
1.5.2	Version 20190608	5
1.5.3	Version master	5
1.6	Unprivileged Architecture	5
1.6.1	Legacy Version 2.2	5
1.6.2	Version 20191213	6
1.7	Other Extensions	6
1.7.1	Zicsr	6
1.7.2	Zifencei	6
1.7.3	Zicbom	6
1.7.4	Zicbop	6
1.7.5	Zicboz	6
1.8	CLIC	7
1.9	Interrupts	7
1.10	Debug Mode	7
1.10.1	Debug State Entry	8
1.10.2	Debug State Exit	8
1.10.3	Debug Registers	9
1.10.4	Debug Mode Execution	9

1.10.5	Debug Single Step . . . . .	9
1.10.6	Debug Ports . . . . .	9
1.11	Trigger Module . . . . .	10
1.11.1	Trigger Module Restrictions . . . . .	10
1.11.2	Trigger Module Parameters . . . . .	10
1.12	Debug Mask . . . . .	11
1.13	Integration Support . . . . .	11
1.13.1	CSR Register External Implementation . . . . .	11
1.14	Limitations . . . . .	11
1.15	Verification . . . . .	11
1.16	References . . . . .	12
<b>2</b>	<b>Configuration</b>	<b>13</b>
2.1	Location . . . . .	13
2.2	GDB Path . . . . .	13
2.3	Semi-Host Library . . . . .	13
2.4	Processor Endian-ness . . . . .	13
2.5	QuantumLeap Support . . . . .	13
2.6	Processor ELF code . . . . .	13
<b>3</b>	<b>All Variants in this model</b>	<b>14</b>
<b>4</b>	<b>Bus Master Ports</b>	<b>16</b>
<b>5</b>	<b>Bus Slave Ports</b>	<b>17</b>
<b>6</b>	<b>Net Ports</b>	<b>18</b>
<b>7</b>	<b>FIFO Ports</b>	<b>19</b>
<b>8</b>	<b>Formal Parameters</b>	<b>20</b>
8.1	Parameters with enumerated types . . . . .	22
8.1.1	Parameter user_version . . . . .	22
8.1.2	Parameter priv_version . . . . .	22
8.1.3	Parameter debug_version . . . . .	22
8.1.4	Parameter rnmi_version . . . . .	23
8.1.5	Parameter debug_mode . . . . .	23
8.2	Parameter values . . . . .	23
<b>9</b>	<b>Execution Modes</b>	<b>26</b>
<b>10</b>	<b>Exceptions</b>	<b>27</b>
<b>11</b>	<b>Hierarchy of the model</b>	<b>28</b>
11.1	Level 1: Hart . . . . .	28
<b>12</b>	<b>Model Commands</b>	<b>29</b>
12.1	Level 1: Hart . . . . .	29
12.1.1	getCSRIndex . . . . .	29
12.1.2	isync . . . . .	29

12.1.3	itrace . . . . .	29
12.1.4	listCSRs . . . . .	30
12.1.4.1	Argument description . . . . .	30
<b>13</b>	<b>Registers</b>	<b>31</b>
13.1	Level 1: Hart . . . . .	31
13.1.1	Core . . . . .	31
13.1.2	Machine_Control_and_Status . . . . .	32
13.1.3	Integration_support . . . . .	34

# 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 RVB64I 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 8: RV32I/RV64I/RV128I base integer instruction set

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 0: extension A (atomic instructions)

misa bit 1: extension B (bit manipulation extension)

misa bit 2: extension C (compressed instructions)

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

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

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

misa bit 7: extension H (hypervisor)

misa bit 10: extension K (cryptographic)

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

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

misa bit 15: extension P (DSP instructions)

misa bit 18: extension S (Supervisor mode)

misa bit 20: extension U (User mode)

misa bit 21: extension V (vector extension)

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

To add features from this list to the 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.

## 1.4 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.

#### 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.

## 1.5 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.

### 1.5.1 Legacy Version 1.10

1.10 version of May 7 2017.

### 1.5.2 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);

### 1.5.3 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.

## 1.6 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.

### 1.6.1 Legacy Version 2.2

2.2 version of May 7 2017.

### 1.6.2 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.

## 1.7 Other Extensions

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

### 1.7.1 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.

### 1.7.2 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.

### 1.7.3 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.

### 1.7.4 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).

### 1.7.5 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 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.

## 1.9 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.

## 1.10 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.

### 1.10.1 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.

### 1.10.2 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.

### 1.10.3 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.

### 1.10.4 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.

### 1.10.5 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.

### 1.10.6 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).

## 1.11 Trigger Module

This model is configured with a trigger module, implementing a subset of the behavior described in Chapter 5 of the RISC-V External Debug Support specification with version specified by parameter “debug\_version” (see References).

### 1.11.1 Trigger Module Restrictions

The model currently supports tdata1 of type 0, type 2 (mcontrol), type 3 (icount), type 4 (itrigger), type 5 (etrigger) and type 6 (mcontrol6). icount triggers are implemented for a single instruction only, with count hard-wired to 1 and automatic zeroing of mode bits when the trigger fires.

### 1.11.2 Trigger Module Parameters

Parameter “trigger\_num” is used to specify the number of implemented triggers. In this variant, “trigger\_num” is 4.

Parameter “tinfo” is used to specify the value of the read-only “tinfo” register, which indicates the trigger types supported. In this variant, “tinfo” is 0x7d.

Parameter “tinfo\_undefined” is used to specify whether the “tinfo” register is undefined, in which case reads of it trap to Machine mode. In this variant, “tinfo\_undefined” is 0.

Parameter “tcontrol\_undefined” is used to specify whether the “tcontrol” register is undefined, in which case accesses to it trap to Machine mode. In this variant, “tcontrol\_undefined” is 0.

Parameter “mcontext\_undefined” is used to specify whether the “mcontext” register is undefined, in which case accesses to it trap to Machine mode. In this variant, “mcontext\_undefined” is 0.

Parameter “scontext\_undefined” is used to specify whether the “scontext” register is undefined, in which case accesses to it trap to Machine mode. In this variant, “scontext\_undefined” is 0.

Parameter “mscontext\_undefined” is used to specify whether the “mscontext” register is undefined, in which case accesses to it trap to Machine mode. In this variant, “mscontext\_undefined” is 0.

Parameter “amo\_trigger” is used to specify whether load/store triggers are activated for AMO instructions. In this variant, “amo\_trigger” is 0.

Parameter “no\_hit” is used to specify whether the “hit” bit in tdata1 is unimplemented. In this variant, “no\_hit” is 0.

Parameter “mcontext\_bits” is used to specify the number of writable bits in the “mcontext” register. In this variant, “mcontext\_bits” is 13.

Parameter “mvalue\_bits” is used to specify the number of writable bits in the “mvalue” field in “extra32”/“extra64” registers; if zero, the “mselect” field is tied to zero. In this variant, “mvalue\_bits” is 13.

Parameter “mcontrol\_maskmax” is used to specify the value of field “maskmax” in the “mcontrol” register. In this variant, “mcontrol\_maskmax” is 63.

## 1.12 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.

## 1.13 Integration Support

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

### 1.13.1 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.

## 1.14 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.

## 1.15 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.

## 1.16 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

## Configuration

### 2.1 Location

This model's VLNv is `riscv.ovpworld.org/processor/riscv/1.0`.

The model source is usually at:

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

The model binary is usually at:

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

### 2.2 GDB Path

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

### 2.3 Semi-Host Library

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

### 2.4 Processor Endian-ness

This is a LITTLE endian model.

### 2.5 QuantumLeap Support

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

### 2.6 Processor ELF code

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

## Chapter 3

# All Variants in this model

This model has these variants

Variant	Description
RV32I	
RV32IM	
RV32IMC	
RV32IMCZ <sub>ce</sub>	
RV32IMAC	
RV32G	
RV32GC	
RV32GCZ <sub>finx</sub>	
RV32GCB	
RV32GCH	
RV32GCK	
RV32GCN	
RV32GCP	
RV32GCV	
RV32E	
RV32EC	
RV64I	
RV64IM	
RV64IMC	
RV64IMCZ <sub>ce</sub>	
RV64IMAC	
RV64G	
RV64GC	
RV64GCZ <sub>finx</sub>	
RV64GCB	
RV64GCH	
RV64GCK	
RV64GCN	
RV64GCP	
RV64GCV	
RVB32I	

RVB32E	
RVB64I	(described in this document)

Table 3.1: All Variants in this model

## Chapter 4

# 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 4.1: Bus Master Ports

## Chapter 5

# Bus Slave Ports

This model has no bus slave ports.

## Chapter 6

# Net Ports

This model has these net ports.

<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
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
deferint	input	optional	Artifact signal causing interrupts to be held off when high

Table 6.1: Net Ports

## Chapter 7

# FIFO Ports

This model has no FIFO ports.



# Chapter 8

## Formal Parameters

Name	Type	Description
<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)
numHarts	Uns32	Specify the number of hart contexts in a multiprocessor
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
<b>Debug</b>		
debug_version	Enumeration	Specify required Debug Architecture version (0.13.2-DRAFT, 0.14.0-DRAFT or 1.0.0-STABLE)
debug_mode	Enumeration	Specify how Debug mode is implemented (none, vector, interrupt or halt)
<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
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>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
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
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/scounteren registers
noinhibit_mask	Uns32	Specify hardware-enforced mask of always-zero bits in mcountinhibit register
cycle_undefined	Boolean	Specify that the cycle CSR is undefined
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
tdata1_mask	Uns64	Specify hardware-enforced mask of writable bits in Trigger Module tdata1 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)
<b>Trigger</b>		

tinfo_undefined	Boolean	Specify that the tinfo CSR is undefined
tcontrol_undefined	Boolean	Specify that the tcontrol CSR is undefined
mcontext_undefined	Boolean	Specify that the mcontext CSR is undefined
scontext_undefined	Boolean	Specify that the scontext CSR is undefined
mscontext_undefined	Boolean	Specify that the mscontext CSR is undefined (Debug Version 0.14.0 and later)
amo_trigger	Boolean	Specify whether AMO load/store operations activate triggers
no_hit	Boolean	Specify that tdata1.hit is unimplemented
trigger_num	Uns32	Specify the number of implemented hardware triggers
tinfo	Uns32	Override tinfo register (for all triggers)
mcontext_bits	Uns32	Specify the number of implemented bits in mcontext
mvalue_bits	Uns32	Specify the number of implemented bits in textra.mvalue (if zero, textra.mselect is tied to zero)
mcontrol_maskmax	Uns32	Specify mcontrol.maskmax value
<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>Fast Interrupt</b>		
CLICLEVELS	Uns32	Specify number of interrupt levels implemented by CLIC, or 0 if CLIC absent

Table 8.1: Parameters that can be set in: Hart

## 8.1 Parameters with enumerated types

### 8.1.1 Parameter user\_version

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

Table 8.2: Values for Parameter user\_version

### 8.1.2 Parameter priv\_version

Set to this value	Description
1.10	Privileged Architecture Version 1.10
1.11	Deprecated and equivalent to 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 8.3: Values for Parameter priv\_version

### 8.1.3 Parameter debug\_version

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

0.13.2-DRAFT	RISC-V External Debug Support Version 0.13.2-DRAFT
0.14.0-DRAFT	RISC-V External Debug Support Version 0.14.0-DRAFT
1.0.0-STABLE	RISC-V External Debug Support Version 1.0.0-STABLE

Table 8.4: Values for Parameter debug\_version

#### 8.1.4 Parameter rnmi\_version

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

Table 8.5: Values for Parameter rnmi\_version

#### 8.1.5 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 8.6: Values for Parameter debug\_mode

### 8.2 Parameter values

These are the current parameter values.

Name	Value
<b>Fundamental</b>	
variant	RVB64I
user_version	20191213
priv_version	20190608
numHarts	0
endian	none
enable_expanded	F
endianFixed	F
misa_MXL	2
misa_Extensions	0x100
add_Extensions	
sub_Extensions	
misa_Extensions_mask	0x100
add_Extensions_mask	
sub_Extensions_mask	
add_implicit_Extensions	
sub_implicit_Extensions	
Zicsr	T
Zifencei	T
Zicbom	F

Zicbop	F
Zicboz	F
<b>Debug</b>	
debug_version	1.0.0-STABLE
debug_mode	none
<b>Interrupts_Exceptions</b>	
rnmi_version	none
mtvec_is_ro	F
tvec_align	0
ecode_mask	0x7fffffffffffff
ecode_nmi	0
tval_zero	F
tval_zero_ebreak	F
tval_ii_code	T
reset_address	0
nmi_address	0
CLINT_address	0
local_int_num	0
unimp_int_mask	0
force_mideleg	0
no_ideleg	0
no_e deleg	0
external_int_id	F
<b>Simulation Artifact</b>	
use_hw_reg_names	F
verbose	F
traceVolatile	F
enable_CSR_bus	F
CSR_remap	
<b>Memory</b>	
unaligned	F
PMP_grain	0
PMP_registers	0
PMP_max_page	0
PMP_decompose	F
<b>Instruction_CSR_Behavior</b>	
wfi_is_nop	F
counteren_mask	0xffffffff
noinhibit_mask	0
cycle_undefined	F
time_undefined	F
instret_undefined	F
hpmcounter_undefined	F
<b>CSR Masks</b>	
mtvec_mask	0

tdata1_mask	0xffffffffffff
mip_mask	0x337
mtvec_sext	F
MXL_writable	F
<b>Trigger</b>	
tinfo_undefined	F
tcontrol_undefined	F
mcontext_undefined	F
scontext_undefined	F
mscontext_undefined	F
amo_trigger	F
no_hit	F
trigger_num	4
tinfo	125
mcontext_bits	13
mvalue_bits	13
mcontrol_maskmax	63
<b>CSR Defaults</b>	
mvendorid	0
marchid	0
mimpid	0
mhartid	0
mtvec	0
<b>Fast Interrupt</b>	
CLICLEVELS	0

Table 8.7: Parameter values

## Chapter 9

# Execution Modes

Mode	Code	Description
Machine	3	Machine mode

Table 9.1: Modes implemented in: Hart

## Chapter 10

# Exceptions

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

Table 10.1: Exceptions implemented in: Hart



# Chapter 11

## Hierarchy of the model

A CPU core may be configured to instance many processors of a Symmetrical Multi Processor (SMP). A CPU core may also have sub elements within a processor, for example hardware threading blocks.

OVP processor models can be written to include SMP blocks and to have many levels of hierarchy. Some OVP CPU models may have a fixed hierarchy, and some may be configured by settings in a configuration register. Please see the register definitions of this model.

This model documentation shows the settings and hierarchy of the default settings for this model variant.

### 11.1 Level 1: Hart

This level in the model hierarchy has 4 commands.

This level in the model hierarchy has 3 register groups:

Group name	Registers
Core	33
Machine_Control_and_Status	133
Integration_support	1

Table 11.1: Register groups

This level in the model hierarchy has no children.

# Chapter 12

## Model Commands

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

### 12.1 Level 1: Hart

#### 12.1.1 getCSRIndex

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

Argument	Type	Description
-name	String	CSR name

Table 12.1: getCSRIndex command arguments

#### 12.1.2 isync

specify instruction address range for synchronous execution

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

Table 12.2: isync command arguments

#### 12.1.3 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 12.3: itrace command arguments

## 12.1.4 listCSRs

### 12.1.4.1 Argument description

List all CSRs in index order

# Chapter 13

## Registers

### 13.1 Level 1: Hart

#### 13.1.1 Core

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

Name	Bits	Initial-Hex	RW	Description
zero	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 13.1: Registers at level 1, type:Hart group:Core

### 13.1.2 Machine\_Control\_and\_Status

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

Name	Bits	Initial-Hex	RW	Description
mstatus	64	1800	rw	Machine Status
misa	64	80000000 00000100	rw	ISA and Extensions
mie	64	0	rw	Machine Interrupt Enable
mtvec	64	0	rw	Machine Trap-Vector Base-Address
mcountinhibit	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
scontext	64	0	rw	Trigger Supervisor Context
tselect	64	0	rw	Trigger Register Select
tdata1	64	0	rw	Trigger Data 1
tdata2	64	0	rw	Trigger Data 2
tdata3	64	0	rw	Trigger Data 3
tinfo	64	7d	rw	Trigger Info
tcontrol	64	0	rw	Trigger Control
mcontext	64	0	rw	Trigger Machine Context
mscontext	64	0	rw	Trigger Machine Context Alias
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
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
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 13.2: Registers at level 1, type:Hart group:Machine\_Control\_and\_Status

### 13.1.3 Integration support

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

Name	Bits	Initial-Hex	RW	Description
commercial	8	0	r-	Commercial feature in use

Table 13.3: Registers at level 1, type:Hart group:Integration\_support