



OVP Guide to Using Processor Models

Model specific information for RISC-V_RV64GCN

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



| | |
|----------|--|
| Author | Imperas Software Limited |
| Version | 20200630.0 |
| Filename | OVP_Model_Specific_Information_riscv_RV64GCN.pdf |
| Created | 2 July 2020 |
| Status | OVP Standard Release |

Copyright Notice

Copyright (c) 2020 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.

Contents

| | | |
|----------|---|-----------|
| 1 | Overview | 1 |
| 1.1 | Description | 1 |
| 1.2 | Licensing | 1 |
| 1.3 | Extensions | 1 |
| 1.3.1 | Available (But Not Enabled) Extensions | 2 |
| 1.4 | General Features | 2 |
| 1.5 | Floating Point Features | 4 |
| 1.6 | CLIC | 4 |
| 1.6.1 | CLIC Common Parameters | 5 |
| 1.6.2 | CLIC Internal-Implementation Parameters | 5 |
| 1.6.3 | CLIC External-Implementation Net Port Interface | 5 |
| 1.7 | Load-Reserved/Store-Conditional Locking | 6 |
| 1.8 | Active Atomic Operation Indication | 7 |
| 1.9 | Interrupts | 7 |
| 1.10 | Debug Mode | 8 |
| 1.10.1 | Debug State Entry | 8 |
| 1.10.2 | Debug State Exit | 9 |
| 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 | 10 |
| 1.11 | Debug Mask | 10 |
| 1.12 | Integration Support | 10 |
| 1.12.1 | CSR Register External Implementation | 10 |
| 1.12.2 | LR/SC Active Address | 10 |
| 1.13 | Limitations | 11 |
| 1.14 | Verification | 11 |
| 1.15 | References | 11 |
| 2 | Configuration | 13 |
| 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 |

| | | |
|-----------|--|-----------|
| 3 | All Variants in this model | 14 |
| 4 | Bus Master Ports | 15 |
| 5 | Bus Slave Ports | 16 |
| 6 | Net Ports | 17 |
| 7 | FIFO Ports | 18 |
| 8 | Formal Parameters | 19 |
| 8.1 | Parameters with enumerated types | 20 |
| 8.1.1 | Parameter user_version | 20 |
| 8.1.2 | Parameter priv_version | 20 |
| 8.1.3 | Parameter mstatus_fs_mode | 21 |
| 8.1.4 | Parameter debug_mode | 21 |
| 9 | Execution Modes | 22 |
| 10 | Exceptions | 23 |
| 11 | Hierarchy of the model | 24 |
| 11.1 | Level 1: Hart | 24 |
| 12 | Model Commands | 25 |
| 12.1 | Level 1: Hart | 25 |
| 12.1.1 | dumpTLB | 25 |
| 12.1.1.1 | Argument description | 25 |
| 12.1.2 | isync | 25 |
| 12.1.3 | itrace | 25 |
| 13 | Registers | 26 |
| 13.1 | Level 1: Hart | 26 |
| 13.1.1 | Core | 26 |
| 13.1.2 | Floating_point | 27 |
| 13.1.3 | User_Control_and_Status | 27 |
| 13.1.4 | Supervisor_Control_and_Status | 28 |
| 13.1.5 | Machine_Control_and_Status | 29 |
| 13.1.6 | Integration_support | 31 |

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 RV64GCN 64-bit processor model

1.2 Licensing

This Model is released under the Open Source Apache 2.0

1.3 Extensions

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

misa bit 0: extension A (atomic instructions)

misa bit 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/64I/128I base ISA

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

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

misa bit 18: extension S (Supervisor mode)

misa bit 20: extension U (User mode)

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

Legacy parameter “misa_Extensions_mask” can also be used. This Uns32-valued parameter specifies all writable bits in the misa Extensions field, replacing any value 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.

1.3.1 Available (But Not Enabled) 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) (NOT ENABLED)

misa bit 4: RV32E base ISA (NOT ENABLED)

misa bit 21: extension V (vector extension) (NOT ENABLED)

misa bit 23: extension X (non-standard extensions present) (NOT ENABLED)

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

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

1.4 General Features

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

Values written to “mtvec” are masked using the value 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.

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

Values written to “stvec” are masked using the value 0xffffffffffffd. A different mask of writable bits may be specified using parameter “stvec_mask” if required. parameter “tvec_align” may be used to specify additional hardware-enforced base address alignment in the same manner as for the “mtvec” register, described above.

Values written to “utvec” are masked using the value 0xffffffffffffd. A different mask of writable bits may be specified using parameter “ucdtvec_mask” if required. parameter “tvec_align” may be used to specify additional hardware-enforced base address alignment in the same manner as for the “mtvec” register, described above.

On reset, the model will restart at address 0x0. A different reset address may be specified using parameter “reset_address” if required.

On an NMI, the model will restart at address 0x0. A different NMI address may be specified using parameter “nmi_address” if required.

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

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

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

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

A 16-bit ASID is implemented. Use parameter “ASID_bits” to specify a different implemented ASID size if required.

This variant supports address translation modes 0, 8 and 9. Use parameter “Sv_modes” to specify a bit mask of different modes if required.

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.

16 PMP entries are implemented by this variant. Use parameter “PMP_registers” to specify a different number of PMP entries; set the parameter to 0 to disable the PMP unit. The PMP grain size (G) is 0, meaning that PMP regions as small as 4 bytes are implemented. Use parameter “PMP_grain” to specify a different grain size if required.

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 Floating Point Features

The D extension is enabled in this variant independently of the F extension. Set parameter “d_requires_f” to “T” to specify that the D extension requires the F extension to be enabled.

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”.

1.6 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 (see references). When “CLICLEVELS” is non-zero, further parameters are made available to configure other aspects of the CLIC, as described below.

The model can be configured either to use an internal CLIC model (if parameter “externalCLIC” is False) or to present a net interface to allow the CLIC to be implemented externally in a platform component (if parameter “externalCLIC” is True). When the CLIC is implemented internally, net ports for standard interrupts and additional local interrupts are available. When the CLIC is implemented externally, a net port interface allowing the highest-priority pending interrupt to be delivered is instead present. This is described below.

1.6.1 CLIC Common Parameters

This section describes parameters applicable whether the CLIC is implemented internally or externally. These are:

“CLICANDBASIC”: this Boolean parameter indicates whether both CLIC and basic interrupt controller are present (if True) or whether only the CLIC is present (if False).

“CLICXNXTI”: this Boolean parameter indicates whether xnxti CSRs are implemented (if True) or unimplemented (if False).

“CLICXCSW”: this Boolean parameter indicates whether xscratchsw and xscratchswl CSRs registers are implemented (if True) or unimplemented (if False).

“mclibase”: this parameter specifies the CLIC base address in physical memory.

“tvt_undefined”: this Boolean parameter indicates whether xtvt CSRs registers are implemented (if True) or unimplemented (if False). If the registers are unimplemented then the model will use basic mode vectored interrupt semantics based on the xtvec CSRs instead of Selective Hardware Vectoring semantics described in the specification.

“intthresh_undefined”: this Boolean parameter indicates whether xintthresh CSRs registers are implemented (if True) or unimplemented (if False).

“mclibase_undefined”: this Boolean parameter indicates whether the mclibase CSR register is implemented (if True) or unimplemented (if False).

1.6.2 CLIC Internal-Implementation Parameters

This section describes parameters applicable only when the CLIC is implemented internally. These are:

“CLICCFGMBITS”: this Uns32 parameter indicates the number of bits implemented in clic-cfg.nmbits, and also indirectly defines CLICPRIVMODES. For cores which implement only Machine mode, or which implement Machine and User modes but not the N extension, the parameter is absent (“CLICCFGMBITS” must be zero in these cases).

“CLICCFGLBITS”: this Uns32 parameter indicates the number of bits implemented in clic-cfg.nlbits.

“CLICSELHVEC”: this Boolean parameter indicates whether Selective Hardware Vectoring is supported (if True) or unsupported (if False).

1.6.3 CLIC External-Implementation Net Port Interface

When the CLIC is externally implemented, net ports are present allowing the external CLIC model to supply the highest-priority pending interrupt and to be notified when interrupts are handled. These are:

“irq_id_i”: this input should be written with the id of the highest-priority pending interrupt.

“irq_lev_i”: this input should be written with the highest-priority interrupt level.

“`irq_sec_i`”: this 2-bit input should be written with the highest-priority interrupt security state (00:User, 01:Supervisor, 11:Machine).

“`irq_shv_i`”: this input port should be written to indicate whether the highest-priority interrupt should be direct (0) or vectored (1). If the “`vtv_undefined` parameter” is False, vectored interrupts will use selective hardware vectoring, as described in the CLIC specification. If “`vtv_undefined`” is True, vectored interrupts will behave like basic mode vectored interrupts.

“`irq_id_i`”: this input should be written with the id of the highest-priority pending interrupt.

“`irq_i`”: this input should be written with 1 to indicate that the external CLIC is presenting an interrupt, or 0 if no interrupt is being presented.

“`irq_ack_o`”: this output is written by the model on entry to the interrupt handler (i.e. when the interrupt is taken). It will be written as an instantaneous pulse (i.e. written to 1, then immediately 0).

“`irq_id_o`”: this output is written by the model with the id of the interrupt currently being handled. It is valid during the instantaneous `irq_ack_o` pulse.

“`sec_lvl_o`”: this output signal indicates the current secure status of the processor, as a 2-bit value (00=User, 01:Supervisor, 11=Machine).

1.7 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.

1.8 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

1.9 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 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 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 (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`);
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 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.

1.12 Integration Support

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

1.12.1 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.

1.12.2 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.

1.13 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.

The processor fully supports the architecturally-specified floating-point instructions.

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

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

1.14 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 Foundations 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.

1.15 References

The Model details are based upon the following specifications:

RISC-V Instruction Set Manual, Volume I: User-Level ISA (User Architecture Version 20190305-Base-Ratification)

RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Privileged Architecture Version 20190405-Priv-MSU-Ratification)

RISC-V Core-Local Interrupt Controller (CLIC) Version 0.9-draft-20191208

RISC-V External Debug Support Version 0.14.0-DRAFT

Chapter 2

Configuration

2.1 Location

This model's VLN 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 | |
| RV32IMAC | |
| RV32G | |
| RV32GC | |
| RV32GCB | |
| RV32GCN | |
| RV32GCV | |
| RV32E | |
| RV32EC | |
| RV64I | |
| RV64IM | |
| RV64IMC | |
| RV64IMAC | |
| RV64G | |
| RV64GC | |
| RV64GCB | |
| RV64GCN | (described in this document) |
| RV64GCV | |
| RVB32I | |
| RVB32E | |
| RVB64I | |

Table 3.1: All Variants in this model

Chapter 4

Bus Master Ports

This model has these bus master ports.

| Name | min | max | Connect? | Description |
|-------------|-----|-----|-----------|-----------------|
| INSTRUCTION | 32 | 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.

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

Table 6.1: Net Ports

Chapter 7

FIFO Ports

This model has no FIFO ports.

Chapter 8

Formal Parameters

| Name | Type | Description |
|-------------------|-------------|--|
| variant | Enumeration | Selects variant (either a generic UISA or a specific model) |
| user_version | Enumeration | Specify required User Architecture version (2.2, 2.3 or 20190305) |
| priv_version | Enumeration | Specify required Privileged Architecture version (1.10, 1.11, 20190405 or master) |
| mstatus_fs_mode | Enumeration | Specify conditions causing update of mstatus.FS to dirty (write_1, write_any or always_dirty) |
| debug_mode | Enumeration | Specify how Debug mode is implemented (none, vector, interrupt or halt) |
| debug_address | Uns64 | Specify address to which to jump to enter debug in vectored mode |
| dexc_address | Uns64 | Specify address to which to jump on debug exception in vectored mode |
| verbose | Boolean | Specify verbose output messages |
| numHarts | Uns32 | Specify the number of hart contexts in a multiprocessor |
| updatePTEA | Boolean | Specify whether hardware update of PTE A bit is supported |
| updatePTED | Boolean | Specify whether hardware update of PTE D bit is supported |
| unaligned | Boolean | Specify whether the processor supports unaligned memory accesses |
| unalignedAMO | Boolean | Specify whether the processor supports unaligned memory accesses for AMO instructions |
| wfi_is_nop | Boolean | Specify whether WFI should be treated as a NOP (if not, halt while waiting for interrupts) |
| 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 |
| counteren_mask | Uns32 | Specify hardware-enforced mask of writable bits in mcounteren/scounteren registers |
| mtvec_mask | Uns64 | Specify hardware-enforced mask of writable bits in mtvec register |
| stvec_mask | Uns64 | Specify hardware-enforced mask of writable bits in stvec register |
| utvec_mask | Uns64 | Specify hardware-enforced mask of writable bits in utvec register |
| 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_ii_code | Boolean | Specify whether mtval/stval contain faulting instruction bits on illegal instruction exception |
| cycle_undefined | Boolean | Specify that the cycle CSR is undefined (reads to it are emulated by a Machine mode trap) |
| time_undefined | Boolean | Specify that the time CSR is undefined (reads to it are emulated by a Machine mode trap) |
| instret_undefined | Boolean | Specify that the instret CSR is undefined (reads to it are emulated by a Machine mode trap) |
| enable_CSR_bus | Boolean | Add artifact CSR bus port, allowing CSR registers to be externally implemented |

| | | |
|----------------------|---------|---|
| 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 |
| xret_preserves_lr | Boolean | Whether an xRET instruction preserves the value of LR |
| ASID_bits | Uns32 | Specify the number of implemented ASID bits |
| lr_sc_grain | Uns32 | Specify byte granularity of ll/sc lock region (constrained to a power of two) |
| reset_address | Uns64 | Override reset vector address |
| nmi_address | Uns64 | Override NMI vector address |
| 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 |
| Sv_modes | Uns32 | Specify bit mask of implemented Sv modes (e.g. 1<<8 is Sv39) |
| local_int_num | Uns32 | Specify number of supplemental local interrupts |
| unimp_int_mask | Uns64 | Specify mask of unimplemented interrupts (e.g. 1<<9 indicates Supervisor external interrupt unimplemented) |
| force_mideleg | Uns64 | Specify mask of interrupts always delegated to lower-priority execution level from Machine execution level |
| force_sideleg | Uns64 | Specify mask of interrupts always delegated to User execution level from Supervisor execution level |
| no_ideleg | Uns64 | Specify mask of interrupts that cannot be delegated to lower-priority execution levels |
| no_edeleg | Uns64 | Specify mask of exceptions that cannot be delegated to lower-priority execution levels |
| external_int_id | Boolean | Whether to add nets allowing External Interrupt ID codes to be forced |
| endian | Endian | Model endian |
| misa_MXL | Uns32 | Override default value of misa.MXL |
| misa_MXL_mask | Uns32 | Override mask of writable bits in misa.MXL |
| misa_Extensions | Uns32 | Override default value of misa.Extensions |
| add_Extensions | String | Add extensions specified by letters to misa.Extensions (for example, specify “VD” to add V and D features) |
| misa_Extensions_mask | Uns32 | Override mask of writable bits in misa.Extensions |
| add_Extensions_mask | String | Add extensions specified by letters to mask of writable bits in misa.Extensions (for example, specify “VD” to add V and D features) |
| 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 |
| mstatus_FS | Uns32 | Override default value of mstatus.FS (initial state of floating point unit) |
| CLICLEVELS | Uns32 | Specify number of interrupt levels implemented by CLIC, or 0 if CLIC absent |

Table 8.1: Parameters that can be set in: Hart

8.1 Parameters with enumerated types

8.1.1 Parameter user_version

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

Table 8.2: Values for Parameter user_version

8.1.2 Parameter priv_version

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

Table 8.3: Values for Parameter `priv_version`

8.1.3 Parameter `mstatus_fs_mode`

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

Table 8.4: Values for Parameter `mstatus_fs_mode`

8.1.4 Parameter `debug_mode`

| Set to this value | Description |
|------------------------|---|
| <code>none</code> | Debug mode not implemented |
| <code>vector</code> | Debug mode implemented by execution at vector |
| <code>interrupt</code> | Debug mode implemented by interrupt |
| <code>halt</code> | Debug mode implemented by halt |

Table 8.5: Values for Parameter `debug_mode`

Chapter 9

Execution Modes

| Mode | Code | Description |
|-------------|------|-----------------|
| User | 0 | User mode |
| Supervisor | 1 | Supervisor mode |
| 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 |
| EnvironmentCallFromUMode | 8 | ECALL instruction executed in User mode |
| EnvironmentCallFromSMode | 9 | ECALL instruction executed in Supervisor mode |
| EnvironmentCallFromMMode | 11 | ECALL instruction executed in Machine mode |
| InstructionPageFault | 12 | Page fault at fetch address |
| LoadPageFault | 13 | Page fault at load address |
| StoreAMOPageFault | 15 | Page fault at store/atomic memory operation address |
| USWInterrupt | 64 | User software interrupt |
| SSWInterrupt | 65 | Supervisor software interrupt |
| MSWInterrupt | 67 | Machine software interrupt |
| UTimerInterrupt | 68 | User timer interrupt |
| STimerInterrupt | 69 | Supervisor timer interrupt |
| MTimerInterrupt | 71 | Machine timer interrupt |
| UExternalInterrupt | 72 | User external interrupt |
| SExternalInterrupt | 73 | Supervisor external 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 3 commands.

This level in the model hierarchy has 6 register groups:

| Group name | Registers |
|-------------------------------|-----------|
| Core | 33 |
| Floating_point | 32 |
| User_Control_and_Status | 43 |
| Supervisor_Control_and_Status | 12 |
| Machine_Control_and_Status | 99 |
| Integration_support | 2 |

Table 11.1: Register groups

This level in the model hierarchy has no children.

Chapter 12

Model Commands

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

12.1 Level 1: Hart

12.1.1 dumpTLB

12.1.1.1 Argument description

show TLB contents

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 12.1: 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 |
| -off | Boolean | disable instruction tracing |
| -on | Boolean | enable instruction tracing |
| -registerchange | Boolean | show registers changed by this instruction |
| -registers | Boolean | show registers after each trace |

Table 12.2: itrace command arguments

Chapter 13

Registers

13.1 Level 1: Hart

13.1.1 Core

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

| Name | Bits | Initial-Hex | RW | Description |
|------|------|-------------|----|-----------------|
| zero | 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 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 13.2: Registers at level 1, type:Hart group:Floating_point

13.1.3 User_Control_and_Status

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

| Name | Bits | Initial-Hex | RW | Description |
|----------|------|-------------|----|-----------------------------------|
| ustatus | 64 | 0 | rw | User Status |
| fflags | 64 | 0 | rw | Floating-Point Flags |
| frm | 64 | 0 | rw | Floating-Point Rounding Mode |
| fcsr | 64 | 0 | rw | Floating-Point Control and Status |
| uie | 64 | 0 | rw | User Interrupt Enable |
| utvec | 64 | 0 | rw | User Trap-Vector Base-Address |
| uscratch | 64 | 0 | rw | User Scratch |

| | | | | |
|--------------|----|---|----|--------------------------------|
| uepc | 64 | 0 | rw | User Exception Program Counter |
| ucause | 64 | 0 | rw | User Cause |
| utval | 64 | 0 | rw | User Trap Value |
| uip | 64 | 0 | rw | User Interrupt Pending |
| 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 13.3: Registers at level 1, type:Hart group:User_Control_and_Status

13.1.4 Supervisor_Control_and_Status

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

| Name | Bits | Initial-Hex | RW | Description |
|------------|------|-------------|----|---|
| sstatus | 64 | 2 00000000 | rw | Supervisor Status |
| sedeleg | 64 | 0 | rw | Supervisor Exception Delegation |
| sideleg | 64 | 0 | rw | Supervisor Interrupt Delegation |
| sie | 64 | 0 | rw | Supervisor Interrupt Enable |
| stvec | 64 | 0 | rw | Supervisor Trap-Vector Base-Address |
| scounteren | 64 | 0 | rw | Supervisor Counter Enable |
| sscratch | 64 | 0 | rw | Supervisor Scratch |
| sepc | 64 | 0 | rw | Supervisor Exception Program Counter |
| scause | 64 | 0 | rw | Supervisor Cause |
| stval | 64 | 0 | rw | Supervisor Trap Value |
| sip | 64 | 0 | rw | Supervisor Interrupt Pending |
| satp | 64 | 0 | rw | Supervisor Address Translation and Protection |

Table 13.4: Registers at level 1, type:Hart group:Supervisor_Control_and_Status

13.1.5 Machine_Control_and_Status

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

| Name | Bits | Initial-Hex | RW | Description |
|---------------|------|----------------------|----|---|
| mstatus | 64 | a 00000000 | rw | Machine Status |
| misa | 64 | 80000000 0014312d | rw | ISA and Extensions |
| medeleg | 64 | 0 | rw | Machine Exception Delegation |
| mideleg | 64 | 0 | rw | Machine Interrupt Delegation |
| mie | 64 | 0 | rw | Machine Interrupt Enable |
| mtvec | 64 | 0 | rw | Machine Trap-Vector Base-Address |
| mcounteren | 64 | 0 | rw | Machine Counter Enable |
| 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 |
| tselect | 64 | - | rw | Debug/Trace Trigger Register Select (not implemented) |
| tdata1 | 64 | - | rw | Debug/Trace Trigger Data 1 (not implemented) |
| tdata2 | 64 | - | rw | Debug/Trace Trigger Data 2 (not implemented) |
| tdata3 | 64 | - | rw | Debug/Trace Trigger Data 3 (not implemented) |
| mcycle | 64 | 0 | rw | Machine Cycle Counter |
| minstret | 64 | 0 | rw | Machine Instructions Retired |
| mhpmcounter3 | 64 | 0 | rw | Machine Performance Monitor Counter 3 |
| mhpmcounter4 | 64 | 0 | rw | Machine Performance Monitor Counter 4 |
| mhpmcounter5 | 64 | 0 | rw | Machine Performance Monitor Counter 5 |
| mhpmcounter6 | 64 | 0 | rw | Machine Performance Monitor Counter 6 |
| mhpmcounter7 | 64 | 0 | rw | Machine Performance Monitor Counter 7 |
| mhpmcounter8 | 64 | 0 | rw | Machine Performance Monitor Counter 8 |
| mhpmcounter9 | 64 | 0 | rw | Machine Performance Monitor Counter 9 |
| mhpmcounter10 | 64 | 0 | rw | Machine Performance Monitor Counter 10 |
| mhpmcounter11 | 64 | 0 | rw | Machine Performance Monitor Counter 11 |
| mhpmcounter12 | 64 | 0 | rw | Machine Performance Monitor Counter 12 |
| mhpmcounter13 | 64 | 0 | rw | Machine Performance Monitor Counter 13 |
| mhpmcounter14 | 64 | 0 | rw | Machine Performance Monitor Counter 14 |
| mhpmcounter15 | 64 | 0 | rw | Machine Performance Monitor Counter 15 |
| mhpmcounter16 | 64 | 0 | rw | Machine Performance Monitor Counter 16 |
| mhpmcounter17 | 64 | 0 | rw | Machine Performance Monitor Counter 17 |
| mhpmcounter18 | 64 | 0 | rw | Machine Performance Monitor Counter 18 |
| mhpmcounter19 | 64 | 0 | rw | Machine Performance Monitor Counter 19 |
| mhpmcounter20 | 64 | 0 | rw | Machine Performance Monitor Counter 20 |
| mhpmcounter21 | 64 | 0 | rw | Machine Performance Monitor Counter 21 |
| mhpmcounter22 | 64 | 0 | rw | Machine Performance Monitor Counter 22 |
| mhpmcounter23 | 64 | 0 | rw | Machine Performance Monitor Counter 23 |
| mhpmcounter24 | 64 | 0 | rw | Machine Performance Monitor Counter 24 |
| mhpmcounter25 | 64 | 0 | rw | Machine Performance Monitor Counter 25 |
| mhpmcounter26 | 64 | 0 | rw | Machine Performance Monitor Counter 26 |
| mhpmcounter27 | 64 | 0 | rw | Machine Performance Monitor Counter 27 |
| mhpmcounter28 | 64 | 0 | rw | Machine Performance Monitor Counter 28 |
| mhpmcounter29 | 64 | 0 | rw | Machine Performance Monitor Counter 29 |
| mhpmcounter30 | 64 | 0 | rw | Machine Performance Monitor Counter 30 |
| mhpmcounter31 | 64 | 0 | rw | Machine Performance Monitor Counter 31 |
| mvendorid | 64 | 0 | r- | Vendor ID |
| marchid | 64 | 0 | r- | Architecture ID |
| mimpid | 64 | 0 | r- | Implementation ID |
| mhartid | 64 | 0 | r- | Hardware Thread ID |

Table 13.5: Registers at level 1, type:Hart group:Machine.Control.and.Status

13.1.6 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 13.6: Registers at level 1, type:Hart group:Integration_support