OVP Guide to Using Processor Models

Model specific information for OpenHwGroup_CV32E40P

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

Author	Imperas Software Limited
Version	20230724.0
Filename	OVP_Model_Specific_Information_openhwgroup_riscv_CV32E40P.pdf
Created	24 July 2023
Status	OVP Standard Release

Model Release Status

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

Contents

Chapter 1

Overview

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

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

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

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

1.1 Description

RISC-V CV32E40P 32-bit processor model

1.2 Licensing

This Model is released under the Open Source Apache 2.0

1.3 Extensions

1.3.1 Extensions Enabled by Default

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

misa bit 2: extension C (compressed instructions)

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

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

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

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

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

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

1.3.2 Enabling Other Extensions

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

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

To add features from this list to the visible set in the misa register, use parameter "add_Extensions". This is a string containing identification letters of features to enable; for example, value "DV" indicates that double-precision floating point and the Vector Extension should be enabled, if they are currently absent and are available on this variant.

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

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

1.3.3 Disabling Extensions

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

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

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

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

1.4 General Features

1.4.1 mtvec CSR

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

Values written to “mtvec” are masked using the value 0xffffff01. 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 0x1. 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”.

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

1.4.4 WFI

WFI will halt the processor until an interrupt occurs. It can instead be configured as a NOP by setting parameter "wfi_is_nop" to True.

The nominal time limit for WFI instructions can be set by parameter "TW_time_limit". In this variant, the time limit is 0 cycles.

Output signal "core_wfi_mode" indicates whether the processor is currently in WFI state.

Input signal "restart_wfi" will cause the processor to restart from WFI state when high.

Parameter "wfi_resume_not_trap" is 0 on this variant, meaning that pending wakeup events when WFI is executed will not prevent a trap occurring. if "wfi_resume_not_trap" is set to 1 then pending wakeup events when WFI is executed will cause the WFI to be treated as a NOP.

1.4.5 time CSR

The "time" CSR is not implemented in this variant and reads of it will cause Illegal Instruction traps. Set parameter "time_undefined" to False to instead specify that "time" is implemented.

1.4.6 mcycle CSR

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

1.4.7 minstret CSR

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

1.4.8 mhpmcounter CSR

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

1.4.9 Unaligned Accesses

Unaligned memory accesses are supported by this variant. Set parameter "unaligned" to "F" to disable such accesses.

Address misaligned exceptions are higher priority than page fault or access fault exceptions on this variant. Set parameter "unaligned_low_pri" to "T" to specify that they are lower priority instead.

1.4.10 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.

Accesses to unimplemented PMP registers cause Illegal Instruction exceptions on this variant. Set parameter "PMP_undefined" to False to indicate that these registers are hard-wired to zero instead.

1.4.11 Time and Timers

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

1. The "time" CSR is implemented ("time_undefined" is False);
2. The "Sstc" extension is present ("Sstc" is True);
3. The internal CLINT model is enabled ("CLINT_address" is non-zero).

For cases 1 and 2, a 64-bit input port "mtime" is present. If this port is connected, it must be driven periodically by an external source with the current time value, which is visible in the "time" CSR and used for timer calculations by the "Sstc" extension. If the port is not connected, the value of time is internally derived with a period specified by the "mtime_Hz" parameter (0Hz by default).

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

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

1.5 Compressed Extension

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

1.5.1 Compressed Extension Parameters

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

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

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

1.5.2 Legacy Version 1.10

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

1.5.3 Version 0.70.1

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

1.5.4 Version 0.70.5

Version 0.70.5, with these changes compared to version 0.70.1:

- access to jt and jalt instructions is enabled by Smstateen.

- jvt.base is WARL and fewer bits than the maximum can be implemented

1.5.5 Version 1.0.0-RC5.7

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

- encodings of jt and jalt instructions changed.

- Zcmb and Zcmpe subsets removed.

1.5.6 Version 1.0

Ratified version 1.0, identical to version 1.0.0-RC5.7.

1.6 Privileged Architecture

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

1.6.1 Legacy Version 1.10

1.10 version of May 7 2017.

1.6.2 Version 20190608

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

- mcountinhibit CSR defined;

- pages are never executable in Supervisor mode if page table entry U bit is 1;

- mstatus.TW is writable if any lower-level privilege mode is implemented (previously, it was just if Supervisor mode was implemented);

1.6.3 Version 20211203

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

- mstatush, mseccfg, mseccfgh, menvcfg, menvcfgh, senvcfg, henvcfg, henvcfgh and mconfigptr CSRs defined;

- xret instructions clear mstatus.MPRV when leaving Machine mode if new mode is less privileged than M-mode;

- maximum number of PMP registers increased to 64;

- data endian is now configurable.

1.6.4 Version 1.12

Official 1.12 version, identical to 20211203.

1.6.5 Version master

Unstable master version, currently identical to 1.12.

1.7 Unprivileged Architecture

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

1.7.1 Legacy Version 2.2

2.2 version of May 7 2017.

1.7.2 Version 20191213

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

- floating point fmin/fmax instruction behavior modified to comply with IEEE 754-201x.

- numerous other optional behaviors can be separately enabled using Z-prefixed parameters.

1.8 Other Extensions

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

1.8.1 Zmmul

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

1.8.2 Zicsr

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

1.8.3 Zifencei

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

1.8.4 Zicbom

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

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

1.8.5 Zicbop

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

1.8.6 Zicboz

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

If Zicboz is present, the cache block size is given by parameter "cmoz_bytes".

1.8.7 Smstateen

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

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

1.8.8 Zawrs

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

1.9 CLIC

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

1.10 Advanced Interrupt Architecture

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

1.11 Interrupts

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

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

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

Parameter “external_int_id” can be used to enable extra interrupt ID input ports on each hart. If the parameter is True then when an external interrupt is taken the value on the ID port is sampled and used to fill the Exception Code field in the relevant “xcause” CSR. For Machine External interrupts, the extra interrupt ID port is called “MExternalInterruptID”; for Supervisor External interrupts, the extra interrupt ID port is called “SExternalInterruptID”.

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.

1.12 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 four 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.

4. If set to value "inject", then operations that would cause entry to Debug mode result in the processor continuing to execute from the current address in Debug mode. The harness should detect that Debug mode has been entered by monitoring the "DM" integration support register, and inject Debug-mode instructions one at a time using function opProcessorSimulateInstruction. Debug mode is exited by either an explicit write of False to the "DM" register or by execution of an injected "dret" instruction, as described by "Debug State Exit" below.

1.12.1 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) - in this case, dcsr.cause will report a cause of trigger (2);

2. By writing a 1 then 0 to net "haltreq" (using opNetWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate) - in this case, dcsr.cause will report a cause of haltreq (3);

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) - in this case, dcsr.cause will report a cause of resethaltreq (5) or haltreq (3), depending on the value of parameter "no_resethaltreq";

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 this case, dcsr.cause will report a cause of ebreak (1);

5. By executing a single instruction when Debug mode entry for the current processor mode is enabled by dcsr.step - in this case, dcsr.cause will report a cause of step (4);

6. By a Trigger Module trigger, when that trigger is configured to enter Debug mode - in this case, dcsr.cause will report a cause of trigger (2).

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.

1.12.2 Debug State Exit

Exit from Debug mode can be performed in either 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.

1.12.3 Debug Registers

When Debug mode is enabled, registers “dcsr” and “dpc” are implemented as described in the specification. Registers “dscratch0” and “dscratch1” may also be implemented (see parameters below). Implemented registers 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.

Parameter “dscratch0_undefined” is used to specify whether the “dscratch0” CSR is undefined, in which case accesses to it trap to Machine mode. In this variant, “dscratch0_undefined” is 0.

Parameter “dscratch1_undefined” is used to specify whether the “dscratch1” CSR is undefined, in which case accesses to it trap to Machine mode. In this variant, “dscratch1_undefined” is 0.

1.12.4 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.

1.12.5 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.

1.12.6 Debug Event Priorities

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

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

1.12.7 Debug Ports

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

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

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

1.12.8 Debug Mode Versions

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

1.12.9 Version 0.13.2-DRAFT

0.13.2-DRAFT version of March 22 2019.

1.12.10 Version 0.14.0-DRAFT

0.14.0-DRAFT version of November 6 2020.

1.12.11 Version 1.0.0-STABLE

1.0.0-STABLE version of February 9 2022.

1.12.12 Version 1.0-STABLE

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

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

1.13 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).

1.13.1 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.

1.13.2 Trigger Module Parameters

Parameter “trigger_num” is used to specify the number of implemented triggers. In this variant, “trigger_num” is 1.

Parameter “tinfo” is used to specify the value of the read-only “tinfo” register, which indicates the trigger types supported and also version information which controls the behavior of “mcontrol6”. In this variant, “tinfo” is 0x04.

Parameter “trigger_match” is used to specify the legal “match” values for triggers of types 2 and 6. This parameter is a bitmask with 1 bits corresponding to legal values; for example, a “trigger_match” of 0xd, means that triggers of types 0, 2 and 3 are supported. In this variant, “trigger_match” is 0xffff.

Parameter “tdata2_undefined” is used to specify whether the “tdata2” register is undefined, in which case reads of it trap to Machine mode. In this variant, “tdata2_undefined” is 0.

Parameter “tdata3_undefined” is used to specify whether the “tdata3” register is undefined, in which case reads of it trap to Machine mode. In this variant, “tdata3_undefined” is 0.

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 1.

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 “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” bits in tdata1 are unimplemented. In this variant, “no_hit” is 1.

Parameter “mcontext_bits” is used to specify the number of writable bits in the “mcontext” register.

In this variant, “mcontext_bits” is 0.

Parameter “mvalue_bits” is used to specify the number of writable bits in the “mvalue” field in “textra32”/“textra64” registers; if zero, the “mselect” field is tied to zero. In this variant, “mvalue_bits” is 0.

Parameter “mcontrol_maskmax” is used to specify the value of field “maskmax” in the “mcontrol” register. In this variant, “mcontrol_maskmax” is 0.

1.14 Debug Mask

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

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

Value 0x004: enable debugging of interrupt state;

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

1.15 Integration Support

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

1.15.1 Command “setPMA -attributes <attrs>-lo <addr>-hi <addr>”

This command allows PMA attributes to be set for the address range lo:hi. The required attributes are described by the “attrs” string, which can contain any combination of these characters:

“r”: allow read access

“w”: allow write access

“x”: allow execute access

“a”: disallow unaligned accesses

“A”: disallow RVMC_USER1 accesses (often AMO and LR/SC)

“P”: disallow RVMC_USER2 accesses (often push/pop)

“1”: allow 1-byte accesses

“2”: allow 2-byte accesses

“4”: allow 4-byte accesses

“8”: allow 8-byte accesses

<space>, “-”: ignored, use for formatting

The command may be used multiple times, in which case PMA attributes for later commands override those specified for earlier ones where ranges overlap. A common idiom is to deny all access to the entire memory range in the first command before adding back permissions for subregions with subsequent commands.

1.15.2 Command "getCSRIndex -name <name>"

This command returns the index number of a named CSR, or -1 if that CSR does not exist.

1.15.3 Command "listCSRs"

This command lists all implemented CSRs in index order.

1.15.4 CSR Register External Implementation

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

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

An artifact net, "readcsr", can also be used to override the value apparently read from a CSR without resorting to the CSR bus. When a CSR is read into a GPR that is not "x0", this net is written with a value encoding the CSR number (in bits 11:0) and destination GPR number (in bits 20:16). To use this net:

1. Install a net monitor callback on "readcsr" using opNetWriteMonitorAdd;

2. When the callback is activated, extract the encoded CSR and GPR numbers;

3. If the CSR number corresponds to a CSR of interest, find the OP register corresponding to the GPR using opProcessorRegByIndex;

4. Use opProcessorRegWrite to modify the GPR value.

1.15.5 External Stimulation of Illegal Instruction Trap

Artifact input net port "illegalinstr" allows Illegal Instruction traps to be raised externally. On a rising edge of the signal connected to this port, the hart will immediately take an Illegal Instruction trap with "xepc" set to the current program counter.

As a special case, if the hart is currently stalled by a WFI instruction ("wfi_is_nop" is False), it will be restarted and take either an Illegal Instruction or Virtual Instruction trap, based on the current processor mode and the governing TW bit.

1.16 Instruction Disassembly

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

If parameter "use_hw_reg_names" is True, instruction disassembly shows hardware names x0-x31. If "use_hw_reg_names" is False, ABI names are shown instead.

If parameter "no_pseudo_inst" is True, instruction disassembly always shows true instructions. If "no_pseudo_inst" is False, pseudo-instructions are shown instead where applicable.

If parameter "show_c_prefix" is True, instruction disassembly of 16-bit instructions will include a compressed prefix (e.g. "c." or "cm."). If "show_c_prefix" is False, the compressed prefix will be omitted.

1.17 Limitations

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

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

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

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

1.18 Verification

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

Also reference tests have been used from various sources including:

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

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

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

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

http://valtrix.in/sting from Valtrix

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

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

1.19 References

The Model details are based upon the following specifications:

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

RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Privileged Architecture Version Ratified-IMFDQC-and-Priv-v1.11)

RISC-V External Debug Support (RISC-V External Debug Support Version 0.13.2-DRAFT)

Chapter 2

Openhwgroup-Specific Extensions

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

- PULP_XPULP Features

- PULP_XCLUSTER Features

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

2.1 Parameter: extensions/PULP_XPULP

The PULP_XPULP CORE-V Instruction Set Extensions may be enabled on RV32 cores by setting the parameter extension_CVE4P/PULP_XPULP to True.

By default the XPULP instructions use the V2 conforming encodings.

2.2 Parameter: extensions/PULP_XPULP_V1

The non-conforming V1 encodings implemented in version 1.0.0, may be selected by setting the parameter extension_CVE4P/PULP_V1 to True. The V1 encodings are based on this release:

- https://github.com/openhwgroup/cv32e40p/releases/tag/cv32e40p_v1.0.0_doc

2.3 Parameter: extensions/PULP_XPULP_V2

The current V2 encodings implemented in version 1.2.0 and later are thedefault if no version option is specified, or may be explicitly selected by setting the parameter extension_CVE4P/PULP_V2 to True. The V2 encodings are based on this release:

- https://github.com/openhwgroup/cv32e40p/releases/tag/cv32e40p_v1.2.1_doc

(Note: The obsolete V2 encodings specified in the version 1.1.0 documentation are not supported)

2.4 Parameter: extensions/PULP_CLUSTER

The PULP_CLUSTER CORE-V Instruction Set Extensions may be enabled on RV32 cores by setting the parameter extension_CVE4P/PULP_CLUSTER to True.

2.5 Extension: PULP_Xfinx

There is no PULP-specific Zfinx extension implemented, but the standard RiscV Zfinx extension may be enabled by:

1) Enabling the F extension using the add_Extensions parameter

1) Specifying the Zfinx version using the zfinx_version parameter.

2.6 PULP_XPULP Extension Status

The XPULP extension is enabled on this variant.

2.6.1 PULP_XPULP Features:

The following features are enabled by the PULP_XPULP parameter:

- Post-Incrementing Load & Store Instructions and Register-Register Load & Store Instructions

- Hardware Loops and the lpcount0/1, lpstart0/1 and lpend0/1 CSRs

- ALU Instructions

- Multiply-Accumulate Instructions

- SIMD Instructions

- uhartid and privlv custom CSRs

2.6.2 PULP_XPULP Limitations:

Valid behaviour of the hardware loops is fully modeled, however reporting errors in hardware loop configurations is TBD.

2.6.3 PULP_XPULP References:

The cv32e40p documentation version 1.3.2 was used as the reference documentation for the PULP_XPULP extension, and may be found here:

- https://docs.openhwgroup.org/projects/cv32e40p-user-manual/en/cv32e40p_v1.3.2/

2.7 PULP_CLUSTER Extension Status

The CLUSTER extension is NOT enabled on this variant.

2.7.1 PULP_CLUSTER Features:

The following features are enabled by the PULP_CLUSTER parameter:

- cv.elw instruction is enabled

- wfi instruction always executes as a NOP when enabled

- The output net port core_sleep_o is added (see below)

- The input net port pulp_clock_Nen_i is added (see below)

2.7.2 PULP_CLUSTER Sleep Unit Interface:

When the PULP_CLUSTER feature is enabled the core_sleep_o and pulp_clock_Nen_i signals to interface to a n Event Unit are supported.

core_sleep_o: this is an output signal that is set to 1 during memory loads that use the cv.elw instruction. An Event Unit needs to be instantiated in the platform that behaves specially for reads when this signal is high.

pulp_clock_Nen_i: the Event Unit should drive the pulp_clock_Nen_i signal high when core_sleep_o is high if the load is not allowed to proceed. This will block the hart until the signal is lowered. On restart, the hart will retry the load, which must now be allowed to succeed.

If you wish to model your own event unit to support the cv.elw instruction behavior in your platform, please contact Imperas for support.

Set parameter extension_CVE4P/PULP_CLUSTER to enable it.

Chapter 3

Configuration

3.1 Location

This model's VLNV is openhwgroup.ovpworld.org/processor/CVE4P/1.0.
The model source is usually at:
$IMPERAS_HOME/ImperasLib/source/openhwgroup.ovpworld.org/processor/CVE4P/1.0
The model binary is usually at:
$IMPERAS_HOME/lib/$IMPERAS_ARCH/ImperasLib/openhwgroup.ovpworld.org/processor/CVE4P/1.0

3.2 GDB Path

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

3.3 Semi-Host Library

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

3.4 Processor Endian-ness

This is a LITTLE endian model.

3.5 QuantumLeap Support

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

3.6 Processor ELF code

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

Chapter 4

All Variants in this model

This model has these variants

Variant	Description
CV32E40P	(described in this document)

Table 4.1: All Variants in this model

Chapter 5

Bus Master Ports

This model has these bus master ports.

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

Table 5.1: Bus Master Ports

Chapter 6

Bus Slave Ports

This model has no bus slave ports.

Chapter 7

Net Ports

This model has these net ports.

Name	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
LocalInterrupt0	input	optional	Local interrupt 0
LocalInterrupt1	input	optional	Local interrupt 1
LocalInterrupt2	input	optional	Local interrupt 2
LocalInterrupt3	input	optional	Local interrupt 3
LocalInterrupt4	input	optional	Local interrupt 4
LocalInterrupt5	input	optional	Local interrupt 5
LocalInterrupt6	input	optional	Local interrupt 6
LocalInterrupt7	input	optional	Local interrupt 7
LocalInterrupt8	input	optional	Local interrupt 8
LocalInterrupt9	input	optional	Local interrupt 9
LocalInterrupt10	input	optional	Local interrupt 10
LocalInterrupt11	input	optional	Local interrupt 11
LocalInterrupt12	input	optional	Local interrupt 12
LocalInterrupt13	input	optional	Local interrupt 13
LocalInterrupt14	input	optional	Local interrupt 14
LocalInterrupt15	input	optional	Local interrupt 15
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
DM	output	optional	Debug state indication
haltreq	input	optional	haltreq (Debug halt request)

resethaltreq	input	optional	resethaltreq (Debug halt request after reset)
illegalinstr	input	optional	Artifact signal raising Illegal Instruction on rising edge
deferint	input	optional	Artifact signal causing interrupts to be held off when high
coverpoint	output	optional	Artifact port written with coverage point identifier
readcsr	output	optional	Artifact port written with CSR/GPR information when CSR is read
core_wfi_mode	output	optional	WFI is active
restart_wfi	input	optional	Artifact signal causing restart from WFI state when high

Table 7.1: Net Ports

Chapter 8

FIFO Ports

This model has no FIFO ports.

Chapter 9

Formal Parameters

Name	Type	Description
Fundamental		
user_version	Enumeration	Specify required User Architecture version
	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
priv_version	Enumeration	Specify required Privileged Architecture version
	1.10	Privileged Architecture Version 1.10
	1.11	Privileged Architecture Version 1.11, equivalent to 20190608
	20190405	Deprecated and equivalent to 20190608
	20190608	Privileged Architecture Version Ratified-IMFDQC-and-Priv-v1.11
	20211203	Privileged Architecture Version 20211203
	1.12	Privileged Architecture Version 1.12, equivalent to 20211203
	master	Privileged Architecture Master Branch as of commit 6bdeb58 (this is subject to change)
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
Compressed_Extension		
compress_version	Enumeration	Specify required Compressed Architecture version
	legacy	Compressed Architecture absent or legacy version
	0.70.1	Compressed Architecture Version 0.70.1
	0.70.5	Compressed Architecture Version 0.70.5
	1.0.0-RC5.7	Compressed Architecture Version 1.0.0-RC5.7
	1.0	Compressed Architecture Version 1.0

Zcea_version	Enumeration	Specify version of Zcea implemented (legacy only)
	none	Zcea not implemented
	0.50.1	Zcea version 0.50.1
Zceb_version	Enumeration	Specify version of Zceb implemented (legacy only)
	none	Zceb not implemented
	0.50.1	Zceb version 0.50.1
Zcee_version	Enumeration	Specify version of Zcee implemented (legacy only)
	none	Zcee not implemented
	1.0.0-rc	Zcee version 1.0.0-rc
Debug_Extension		
debug_version	Enumeration	Specify required Debug Architecture version
	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
	1.0-STABLE	RISC-V External Debug Support Version 1.0-STABLE
debug_mode	Enumeration	Specify how Debug mode is implemented
	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
	inject	Debug mode implemented using injected instructions
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
debug_eret_mode	Enumeration	Specify behavior for MRET, SRET or URET in Debug mode (nop, jump to dexc_address or trap to dexc_address)
	nop	MRET, SRET or URET in Debug mode is a nop
	jump_to_dexc_address	MRET, SRET or URET in Debug mode jumps to dexc_address
	trap_to_dexc_address	MRET, SRET or URET in Debug mode traps to dexc_address
debug_priority	Enumeration	Specify relative priorities of simultaneous debug events
	asxh	after trigger ->step ->execute address ->haltreq
	ashx	after trigger ->step ->haltreq ->execute address
	ahsx	after trigger ->haltreq ->step ->execute address
	hasx	haltreq ->after trigger ->step ->execute address
	original	legacy alias of asxh
	PR693	legacy alias of ashx
	halt_not_step	legacy alias of ahsx
dcsr_ebreak_mask	Uns32	Specify mask of dcsr.ebreak fields that reset to 1 (ebreak instructions enter Debug mode)
no_resethaltreq	Boolean	Specify that haltreq (code 3) should be reported when resethaltreq signal is applied at reset
dscratch0_undefined	Boolean	Specify that the dscratch0 CSR is undefined
dscratch1_undefined	Boolean	Specify that the dscratch1 CSR is undefined
Interrupts_Exceptions		
rnmi_version	Enumeration	Specify required RNMI Architecture version
	none	RNMI not implemented
	0.2.1	RNMI version 0.2.1
	0.4	RNMI version 0.4
mtvec_is_ro	Boolean	Specify whether mtvec CSR is read-only
tvec_align	Uns32	Specify hardware-enforced alignment of mtvec/stvec/utvec when Vectored interrupt mode enabled
ecode_mask	Uns64	Specify hardware-enforced mask of writable bits in xcause.ExceptionCode
ecode_nmi	Uns64	Specify xcause.ExceptionCode for NMI
nmi_is_latched	Boolean	Specify whether NMI input is latched on rising edge (if False, it is level-sensitive)
tval_zero	Boolean	Specify whether mtval/stval/utval are hard wired to zero
tval_zero_ebreak	Boolean	Specify whether mtval/stval/utval are set to zero by an ebreak

tval_ii_code	Boolean	Specify whether mtval/stval contain faulting instruction bits on illegal instruction exception
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_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
Simulation_Artifact		
use_hw_reg_names	Boolean	Specify whether to use hardware register names x0-x31 and f0-f31 instead of ABI register names
no_pseudo_inst	Boolean	Specify whether pseudo-instructions should not be reported in trace and disassembly
show_c_prefix	Boolean	Specify whether compressed instruction prefix should be reported in trace and disassembly
verbose	Boolean	Specify verbose output messages
traceVolatile	Boolean	Specify whether volatile registers (e.g. minstret) should be shown in change trace
enable_CSR_bus	Boolean	Add artifact CSR bus port, allowing CSR registers to be externally implemented
CSR_remap	String	Comma-separated list of CSR number mappings, each of the form <csrName>=<number>
Memory		
unaligned_low_pri	Boolean	Specify whether address misaligned exceptions are lower priority than page or access fault exceptions
unaligned	Boolean	Specify whether the processor supports unaligned memory accesses
Instruction_CSR_Behavior		
wfi_is_nop	Boolean	Specify whether WFI should be treated as a NOP (if not, halt while waiting for interrupts)
wfi_resume_not_trap	Boolean	Specify whether pending wakeup events should cause WFI to be treated as a NOP instead of taking a trap
TW_time_limit	Uns32	Specify nominal cycle timeout for instructions controlled by mstatus.TW
counteren_mask	Uns32	Specify hardware-enforced mask of writable bits in mcounteren/scounteren registers
noinhibit_mask	Uns32	Specify hardware-enforced mask of always-zero bits in mcountinhibit register
mcycle_undefined	Boolean	Specify that the mcycle CSR is undefined
time_undefined	Boolean	Specify that the time CSR is undefined
minstret_undefined	Boolean	Specify that the minstret CSR is undefined
mhpmcounter_undefined	Boolean	Specify that the mhpmcounter CSRs are undefined
CSR_Masks		
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
Trigger		
tdata2_undefined	Boolean	Specify that the tdata2 CSR is undefined
tdata3_undefined	Boolean	Specify that the tdata3 CSR is undefined

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* bits are unimplemented
trigger_num	Uns32	Specify the number of implemented hardware triggers
tinfo	Uns32	Override tinfo register (for all triggers)
trigger_match	Uns32	Specify legal "match" values for triggers of type 2 and 6 (bitmask)
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
chain_tval	Enumeration	Specify which trigger provides xtval when triggers are chained
	first	first matching trigger provides xtval
	last	last matching trigger provides xtval
	first_non_epc	first matching trigger provides xtval (prefer non-epc)
	last_non_epc	last matching trigger provides xtval (prefer non-epc)
PMP Configuration		
PMP_grain	Uns32	Specify PMP region granularity, G (0 =>4 bytes, 1 =>8 bytes, etc)
PMP_registers	Uns32	Specify the number of implemented PMP address registers
PMP_max_page	Uns32	Specify the maximum size of PMP region to map if non-zero (may improve performance; constrained to a power of two)
PMP_decompose	Boolean	Whether unaligned PMP accesses are decomposed into separate aligned accesses
PMP_undefined	Boolean	Whether accesses to unimplemented PMP registers are undefined (if True) or write ignored and zero (if False)
PMP_maskparams	Boolean	Enable parameters to change the read-only masks for PMP CSRs
PMP_initialparams	Boolean	Enable parameters to change the reset values for PMP CSRs
Other_Extensions		
Smstateen	Boolean	Specify that Smstateen is implemented
Zihintntl	Boolean	Specify that Zihintntl is implemented (instruction decode only, implemented as NOP)
Zicond	Boolean	Specify that Zicond is implemented
Zicsr	Boolean	Specify that Zicsr is implemented
Zifencei	Boolean	Specify that Zifencei is implemented
Zicbom	Boolean	Specify that Zicbom is implemented
Zicbop	Boolean	Specify that Zicbop is implemented
Zicboz	Boolean	Specify that Zicboz is implemented
Zawrs	Boolean	Specify that Zawrs is implemented
Zmmul	Boolean	Specify that Zmmul is implemented
CSR_Defaults		
mvendorid	Uns64	Override mvendorid register
marchid	Uns64	Override marchid register
mimpid	Uns64	Override mimpid register
mhartid	Uns64	Override mhartid register (or first mhartid of an incrementing sequence if this is an SMP variant)
mtvec	Uns64	Override mtvec register
Fast_Interrupt		
CLICLEVELS	Uns32	Specify number of interrupt levels implemented by CLIC, or 0 if CLIC absent
AIA_Interrupts		
Smaia	Boolean	Specify that Smaia CSRs are present

Table 9.1: Parameters that can be set in: Hart

9.1 Extension Parameters

Name	Type	Description
debug	Boolean	debug flags
mcountinhibit_reset	Uns32	reset value of mcountinhibit
tdata1_reset	Uns32	reset value of tdata1
dcsr_reset	Uns32	reset value of dcsr
PULP_XPULP	Boolean	Enable XPULP features (CORE-V Extensions, excluding cv.elw
PULP_V1	Boolean	Enable obsolete V1 encodings for PULP opcodes
PULP_V2	Boolean	Enable current V2 encodings for PULP opcodes (default if neither PULP_V1 or PULP_V2 are specified
PULP_CLUSTER	Boolean	Enable PULP Cluster Extension

Table 9.2: Parameters for extension_CVE4P

9.2 Parameters with enumerated types

9.2.1 Parameter user_version

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

Table 9.3: Values for Parameter user_version

9.2.2 Parameter priv_version

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

Table 9.4: Values for Parameter priv_version

9.2.3 Parameter compress_version

Set to this value	Description
legacy	Compressed Architecture absent or legacy version
0.70.1	Compressed Architecture Version 0.70.1
0.70.5	Compressed Architecture Version 0.70.5
1.0.0-RC5.7	Compressed Architecture Version 1.0.0-RC5.7
1.0	Compressed Architecture Version 1.0

Table 9.5: Values for Parameter compress_version

9.2.4 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
1.0-STABLE	RISC-V External Debug Support Version 1.0-STABLE

Table 9.6: Values for Parameter debug_version

9.2.5 Parameter rnmi_version

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

Table 9.7: Values for Parameter rnmi_version

9.2.6 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
inject	Debug mode implemented using injected instructions

Table 9.8: Values for Parameter debug_mode

9.2.7 Parameter debug_eret_mode

Set to this value	Description
nop	MRET, SRET or URET in Debug mode is a nop
jump_to_dexc_address	MRET, SRET or URET in Debug mode jumps to dexc_address
trap_to_dexc_address	MRET, SRET or URET in Debug mode traps to dexc_address

Table 9.9: Values for Parameter debug_eret_mode

9.2.8 Parameter debug_priority

Set to this value	Description
asxh	after trigger ->step ->execute address ->haltreq
ashx	after trigger ->step ->haltreq ->execute address
ahsx	after trigger ->haltreq ->step ->execute address
hasx	haltreq ->after trigger ->step ->execute address
original	legacy alias of asxh
PR693	legacy alias of ashx
halt_not_step	legacy alias of ahsx

Table 9.10: Values for Parameter debug_priority

9.2.9 Parameter chain_tval

Set to this value	Description
first	first matching trigger provides xtval
last	last matching trigger provides xtval
first_non_epc	first matching trigger provides xtval (prefer non-epc)
last_non_epc	last matching trigger provides xtval (prefer non-epc)

Table 9.11: Values for Parameter chain_tval

9.2.10 Parameter Zcea_version

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

Table 9.12: Values for Parameter Zcea_version

9.2.11 Parameter Zceb_version

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

Table 9.13: Values for Parameter Zceb_version

9.2.12 Parameter Zcee_version

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

Table 9.14: Values for Parameter Zcee_version

9.3 Parameter values and limits

These are the formal parameter limits and actual parameter values

Name	Min	Max	Default	Actual
Fundamental				
variant			CV32E40P	CV32E40P
user_version			20191213	20191213
priv_version			20190608	20190608
endian				none
enable_expanded			t	f
endianFixed			t	f
misa_MXL	1	2	1	1

misa_Extensions	0	67108863	8392964	0x801104
add_Extensions				
sub_Extensions				
misa_Extensions_mask	0	67108863	0	0
add_Extensions_mask				
sub_Extensions_mask				
add_implicit_Extensions				
sub_implicit_Extensions				
Compressed_Extension				
compress_version			legacy	legacy
Zcea_version			none	none
Zceb_version			none	none
Zcee_version			none	none
Debug_Extension				
debug_version			0.13.2-DRAFT	0.13.2-DRAFT
debug_mode			vector	vector
debug_address	0x0	0xffffffffffffffff	0x1a110800	0x1a110800
dexc_address	0x0	0xffffffffffffffff	0x1a111000	0x1a111000
debug_eret_mode			jump_to_dexc_address	jump_to_dexc_address
debug_priority			asxh	asxh
dcsr_ebreak_mask	0	63	0	0
no_resethaltreq			t	f
dscratch0_undefined			t	f
dscratch1_undefined			t	f
Interrupts_Exceptions				
rnmi_version			none	none
mtvec_is_ro			t	f
tvec_align	0	65536	0	0
ecode_mask	0x0	0xffffffffffffffff	0x1f	31
ecode_nmi	0x0	0xffffffffffffffff	0x0	0
nmi_is_latched			t	t
tval_zero			t	t
tval_zero_ebreak			t	f
tval_ii_code			t	f
reset_address	0x0	0xffffffffffffffff	0x0	0
nmi_address	0x0	0xffffffffffffffff	0x0	0
CLINT_address	0x0	0xffffffffffffffff	0x0	0
local_int_num	0	16	16	16
unimp_int_mask	0x0	0xffffffffffffffff	0x0	0
force_mideleg	0x0	0xffffffffffffffff	0x0	0
no_ideleg	0x0	0xffffffffffffffff	0x0	0
no_edeleg	0x0	0xffffffffffffffff	0x0	0
external_int_id			t	f
Simulation_Artifact				
use_hw_reg_names			t	f

no_pseudo_inst			t	f
show_c_prefix			t	f
verbose			t	f
traceVolatile			t	f
enable_CSR_bus			t	f
CSR_remap				
Memory				
unaligned_low_pri			t	f
unaligned			t	t
Instruction_CSR_Behavior				
wfi_is_nop			t	f
wfi_resume_not_trap			t	f
TW_time_limit	0	4294967295	0	0
counteren_mask	0	4294967295	4294967295	0xffffffff
noinhibit_mask	0	4294967295	4294967280	0xfffffff0
mcycle_undefined			t	f
time_undefined			t	t
minstret_undefined			t	f
mhpmcounter_undefined			t	f
CSR_Masks				
mtvec_mask	0x0	0xffffffffffffffff	0xffffff01	0xffffff01
tdata1_mask	0x0	0xffffffffffffffff	0x4	4
mip_mask	0x0	0xffffffffffffffff	0x337	0x337
mtvec_sext			t	f
Trigger				
tdata2_undefined			t	f
tdata3_undefined			t	f
tinfo_undefined			t	f
tcontrol_undefined			t	t
mcontext_undefined			t	f
scontext_undefined			t	f
mscontext_undefined			t	f
amo_trigger			t	f
no_hit			t	t
trigger_num	0	255	1	1
tinfo	0	16842751	4	4
trigger_match	1	65535	65535	0xffff
mcontext_bits	0	32	0	0
mvalue_bits	0	6	0	0
mcontrol_maskmax	0	63	0	0
chain_tval			first	first
PMP Configuration				
PMP_grain	0	29	0	0
PMP_registers	0	16	0	0
PMP_max_page	0	4294967295	0	0

PMP_decompose			t	f
PMP_undefined			t	t
PMP_maskparams			t	f
PMP_initialparams			t	f
Other_Extensions				
Smstateen			t	f
Zihintntl			t	f
Zicond			t	f
Zicsr			t	t
Zifencei			t	t
Zicbom			t	f
Zicbop			t	f
Zicboz			t	f
Zawrs			t	f
Zmmul			t	f
CSR_Defaults				
mvendorid	0x0	0xffffffffffffffff	0x602	0x602
marchid	0x0	0xffffffffffffffff	0x4	4
mimpid	0x0	0xffffffffffffffff	0x0	0
mhartid	0x0	0xffffffffffffffff	0x0	0
mtvec	0x0	0xffffffffffffffff	0x1	1
Fast_Interrupt				
CLICLEVELS	0	256	0	0
AIA_Interrupts				
Smaia			t	f
extension_CVE4P				
debug			t	f
mcountinhibit_reset	0	4294967295	13	13
tdata1_reset	0	4294967295	671092800	0x28001040
dcsr_reset	0	4294967295	1073741827	0x40000003
PULP_XPULP			t	t
PULP_V1			t	f
PULP_V2			t	f
PULP_CLUSTER			t	f

Table 9.15: Parameter values and limits

Chapter 10

Execution Modes

Mode	Code	Description
Machine	3	Machine mode
Debug	6	Debug mode

Table 10.1: Modes implemented in: Hart

Chapter 11

Exceptions

Exception	Code	Description
InstructionAccessFault	1	No access permission for fetch
IllegalInstruction	2	Undecoded, unimplemented or disabled instruction
Breakpoint	3	EBREAK instruction executed
LoadAccessFault	5	No access permission for load
StoreAMOAccessFault	7	No access permission for store/atomic memory operation
EnvironmentCallFromMMode	11	ECALL instruction executed in Machine mode
MSWInterrupt	67	Machine software interrupt
MTimerInterrupt	71	Machine timer interrupt
MExternalInterrupt	75	Machine external interrupt
LocalInterrupt0	80	Local interrupt 0 (id 16)
LocalInterrupt1	81	Local interrupt 1 (id 17)
LocalInterrupt2	82	Local interrupt 2 (id 18)
LocalInterrupt3	83	Local interrupt 3 (id 19)
LocalInterrupt4	84	Local interrupt 4 (id 20)
LocalInterrupt5	85	Local interrupt 5 (id 21)
LocalInterrupt6	86	Local interrupt 6 (id 22)
LocalInterrupt7	87	Local interrupt 7 (id 23)
LocalInterrupt8	88	Local interrupt 8 (id 24)
LocalInterrupt9	89	Local interrupt 9 (id 25)
LocalInterrupt10	90	Local interrupt 10 (id 26)
LocalInterrupt11	91	Local interrupt 11 (id 27)
LocalInterrupt12	92	Local interrupt 12 (id 28)
LocalInterrupt13	93	Local interrupt 13 (id 29)
LocalInterrupt14	94	Local interrupt 14 (id 30)
LocalInterrupt15	95	Local interrupt 15 (id 31)
GenericNMI	4294967295	Generic NMI

Table 11.1: Exceptions implemented in: Hart

Chapter 12

Hierarchy of the model

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

12.1 Level 1: Hart

This level in the model hierarchy has 6 commands.
This level in the model hierarchy has 3 register groups:

Group name	Registers
Core	33
Machine_Control_and_Status	187
Integration_support	3

Table 12.1: Register groups

This level in the model hierarchy has no children.

Chapter 13

Model Commands

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

13.1 Level 1: Hart

13.1.1 debugflags

show or modify the processor debug flags

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

Table 13.1: debugflags command arguments

13.1.2 getCSRIndex

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

Argument	Type	Description
-name	String	CSR name

Table 13.2: getCSRIndex command arguments

13.1.3 isync

specify instruction address range for synchronous execution

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

Table 13.3: isync command arguments

13.1.4 itrace

enable or disable instruction tracing

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

Table 13.4: itrace command arguments

13.1.5 listCSRs

13.1.5.1 Argument description

List all CSRs in index order

13.1.6 setPMA

Set PMA region permissions and legal access sizes

Argument	Type	Description
-attributes	String	region attributes (string containing r, w, x, a, A, P, 1, 2, 4 or 8)
-hi	Uns64	high address
-lo	Uns64	low address

Table 13.5: setPMA command arguments

Chapter 14

Registers

14.1 Level 1: Hart

14.1.1 Core

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

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

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

14.1.2 Machine_Control_and_Status

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

Name	Bits	Initial-Hex	RW	Description
mstatus	32	1800	rw	Machine Status
misa	32	40801104	rw	ISA and Extensions
mie	32	0	rw	Machine Interrupt Enable
mtvec	32	1	rw	Machine Trap-Vector Base-Address
mcountinhibit	32	d	rw	Machine Counter Inhibit
mhpmevent3	32	0	rw	Machine Performance Monitor Event Select 3
mhpmevent4	32	0	rw	Machine Performance Monitor Event Select 4
mhpmevent5	32	0	rw	Machine Performance Monitor Event Select 5
mhpmevent6	32	0	rw	Machine Performance Monitor Event Select 6
mhpmevent7	32	0	rw	Machine Performance Monitor Event Select 7
mhpmevent8	32	0	rw	Machine Performance Monitor Event Select 8
mhpmevent9	32	0	rw	Machine Performance Monitor Event Select 9
mhpmevent10	32	0	rw	Machine Performance Monitor Event Select 10
mhpmevent11	32	0	rw	Machine Performance Monitor Event Select 11
mhpmevent12	32	0	rw	Machine Performance Monitor Event Select 12
mhpmevent13	32	0	rw	Machine Performance Monitor Event Select 13
mhpmevent14	32	0	rw	Machine Performance Monitor Event Select 14
mhpmevent15	32	0	rw	Machine Performance Monitor Event Select 15
mhpmevent16	32	0	rw	Machine Performance Monitor Event Select 16
mhpmevent17	32	0	rw	Machine Performance Monitor Event Select 17
mhpmevent18	32	0	rw	Machine Performance Monitor Event Select 18
mhpmevent19	32	0	rw	Machine Performance Monitor Event Select 19
mhpmevent20	32	0	rw	Machine Performance Monitor Event Select 20
mhpmevent21	32	0	rw	Machine Performance Monitor Event Select 21
mhpmevent22	32	0	rw	Machine Performance Monitor Event Select 22
mhpmevent23	32	0	rw	Machine Performance Monitor Event Select 23
mhpmevent24	32	0	rw	Machine Performance Monitor Event Select 24
mhpmevent25	32	0	rw	Machine Performance Monitor Event Select 25
mhpmevent26	32	0	rw	Machine Performance Monitor Event Select 26
mhpmevent27	32	0	rw	Machine Performance Monitor Event Select 27
mhpmevent28	32	0	rw	Machine Performance Monitor Event Select 28
mhpmevent29	32	0	rw	Machine Performance Monitor Event Select 29
mhpmevent30	32	0	rw	Machine Performance Monitor Event Select 30
mhpmevent31	32	0	rw	Machine Performance Monitor Event Select 31
mscratch	32	0	rw	Machine Scratch
mepc	32	0	rw	Machine Exception Program Counter
mcause	32	0	rw	Machine Cause
mtval	32	0	rw	Machine Trap Value
mip	32	0	rw	Machine Interrupt Pending
tselect	32	0	rw	Trigger Register Select
tdata1	32	28001040	rw	Trigger Data 1
tdata2	32	0	rw	Trigger Data 2
tdata3	32	0	rw	Trigger Data 3
tinfo	32	4	rw	Trigger Info
mcontext	32	0	rw	Trigger Machine Context
scontext	32	0	rw	Trigger Supervisor Context
dcsr	32	40000003	rw	Debug Control and Status
dpc	32	0	rw	Debug PC

dscratch0	32	0	rw	Debug Scratch 0
dscratch1	32	0	rw	Debug Scratch 1
mcycle	32	0	rw	Machine Cycle Counter
minstret	32	0	rw	Machine Instructions Retired
mhpmcounter3	32	0	rw	Machine Performance Monitor Counter 3
mhpmcounter4	32	0	rw	Machine Performance Monitor Counter 4
mhpmcounter5	32	0	rw	Machine Performance Monitor Counter 5
mhpmcounter6	32	0	rw	Machine Performance Monitor Counter 6
mhpmcounter7	32	0	rw	Machine Performance Monitor Counter 7
mhpmcounter8	32	0	rw	Machine Performance Monitor Counter 8
mhpmcounter9	32	0	rw	Machine Performance Monitor Counter 9
mhpmcounter10	32	0	rw	Machine Performance Monitor Counter 10
mhpmcounter11	32	0	rw	Machine Performance Monitor Counter 11
mhpmcounter12	32	0	rw	Machine Performance Monitor Counter 12
mhpmcounter13	32	0	rw	Machine Performance Monitor Counter 13
mhpmcounter14	32	0	rw	Machine Performance Monitor Counter 14
mhpmcounter15	32	0	rw	Machine Performance Monitor Counter 15
mhpmcounter16	32	0	rw	Machine Performance Monitor Counter 16
mhpmcounter17	32	0	rw	Machine Performance Monitor Counter 17
mhpmcounter18	32	0	rw	Machine Performance Monitor Counter 18
mhpmcounter19	32	0	rw	Machine Performance Monitor Counter 19
mhpmcounter20	32	0	rw	Machine Performance Monitor Counter 20
mhpmcounter21	32	0	rw	Machine Performance Monitor Counter 21
mhpmcounter22	32	0	rw	Machine Performance Monitor Counter 22
mhpmcounter23	32	0	rw	Machine Performance Monitor Counter 23
mhpmcounter24	32	0	rw	Machine Performance Monitor Counter 24
mhpmcounter25	32	0	rw	Machine Performance Monitor Counter 25
mhpmcounter26	32	0	rw	Machine Performance Monitor Counter 26
mhpmcounter27	32	0	rw	Machine Performance Monitor Counter 27
mhpmcounter28	32	0	rw	Machine Performance Monitor Counter 28
mhpmcounter29	32	0	rw	Machine Performance Monitor Counter 29
mhpmcounter30	32	0	rw	Machine Performance Monitor Counter 30
mhpmcounter31	32	0	rw	Machine Performance Monitor Counter 31
mcycleh	32	0	rw	Machine Cycle Counter High
minstreth	32	0	rw	Machine Instructions Retired High
mhpmcounterh3	32	0	rw	Machine Performance Monitor Counter High 3
mhpmcounterh4	32	0	rw	Machine Performance Monitor Counter High 4
mhpmcounterh5	32	0	rw	Machine Performance Monitor Counter High 5
mhpmcounterh6	32	0	rw	Machine Performance Monitor Counter High 6
mhpmcounterh7	32	0	rw	Machine Performance Monitor Counter High 7
mhpmcounterh8	32	0	rw	Machine Performance Monitor Counter High 8
mhpmcounterh9	32	0	rw	Machine Performance Monitor Counter High 9
mhpmcounterh10	32	0	rw	Machine Performance Monitor Counter High 10
mhpmcounterh11	32	0	rw	Machine Performance Monitor Counter High 11
mhpmcounterh12	32	0	rw	Machine Performance Monitor Counter High 12
mhpmcounterh13	32	0	rw	Machine Performance Monitor Counter High 13
mhpmcounterh14	32	0	rw	Machine Performance Monitor Counter High 14
mhpmcounterh15	32	0	rw	Machine Performance Monitor Counter High 15
mhpmcounterh16	32	0	rw	Machine Performance Monitor Counter High 16
mhpmcounterh17	32	0	rw	Machine Performance Monitor Counter High 17
mhpmcounterh18	32	0	rw	Machine Performance Monitor Counter High 18
mhpmcounterh19	32	0	rw	Machine Performance Monitor Counter High 19
mhpmcounterh20	32	0	rw	Machine Performance Monitor Counter High 20
mhpmcounterh21	32	0	rw	Machine Performance Monitor Counter High 21
mhpmcounterh22	32	0	rw	Machine Performance Monitor Counter High 22
mhpmcounterh23	32	0	rw	Machine Performance Monitor Counter High 23

mhpmcounterh24	32	0	rw	Machine Performance Monitor Counter High 24
mhpmcounterh25	32	0	rw	Machine Performance Monitor Counter High 25
mhpmcounterh26	32	0	rw	Machine Performance Monitor Counter High 26
mhpmcounterh27	32	0	rw	Machine Performance Monitor Counter High 27
mhpmcounterh28	32	0	rw	Machine Performance Monitor Counter High 28
mhpmcounterh29	32	0	rw	Machine Performance Monitor Counter High 29
mhpmcounterh30	32	0	rw	Machine Performance Monitor Counter High 30
mhpmcounterh31	32	0	rw	Machine Performance Monitor Counter High 31
cycle	32	0	r-	Cycle Counter
instret	32	0	r-	Instructions Retired
hpmcounter3	32	0	r-	Performance Monitor Counter 3
hpmcounter4	32	0	r-	Performance Monitor Counter 4
hpmcounter5	32	0	r-	Performance Monitor Counter 5
hpmcounter6	32	0	r-	Performance Monitor Counter 6
hpmcounter7	32	0	r-	Performance Monitor Counter 7
hpmcounter8	32	0	r-	Performance Monitor Counter 8
hpmcounter9	32	0	r-	Performance Monitor Counter 9
hpmcounter10	32	0	r-	Performance Monitor Counter 10
hpmcounter11	32	0	r-	Performance Monitor Counter 11
hpmcounter12	32	0	r-	Performance Monitor Counter 12
hpmcounter13	32	0	r-	Performance Monitor Counter 13
hpmcounter14	32	0	r-	Performance Monitor Counter 14
hpmcounter15	32	0	r-	Performance Monitor Counter 15
hpmcounter16	32	0	r-	Performance Monitor Counter 16
hpmcounter17	32	0	r-	Performance Monitor Counter 17
hpmcounter18	32	0	r-	Performance Monitor Counter 18
hpmcounter19	32	0	r-	Performance Monitor Counter 19
hpmcounter20	32	0	r-	Performance Monitor Counter 20
hpmcounter21	32	0	r-	Performance Monitor Counter 21
hpmcounter22	32	0	r-	Performance Monitor Counter 22
hpmcounter23	32	0	r-	Performance Monitor Counter 23
hpmcounter24	32	0	r-	Performance Monitor Counter 24
hpmcounter25	32	0	r-	Performance Monitor Counter 25
hpmcounter26	32	0	r-	Performance Monitor Counter 26
hpmcounter27	32	0	r-	Performance Monitor Counter 27
hpmcounter28	32	0	r-	Performance Monitor Counter 28
hpmcounter29	32	0	r-	Performance Monitor Counter 29
hpmcounter30	32	0	r-	Performance Monitor Counter 30
hpmcounter31	32	0	r-	Performance Monitor Counter 31
cycleh	32	0	r-	Cycle Counter High
instreth	32	0	r-	Instructions Retired High
hpmcounterh3	32	0	r-	Performance Monitor High 3
hpmcounterh4	32	0	r-	Performance Monitor High 4
hpmcounterh5	32	0	r-	Performance Monitor High 5
hpmcounterh6	32	0	r-	Performance Monitor High 6
hpmcounterh7	32	0	r-	Performance Monitor High 7
hpmcounterh8	32	0	r-	Performance Monitor High 8
hpmcounterh9	32	0	r-	Performance Monitor High 9
hpmcounterh10	32	0	r-	Performance Monitor High 10
hpmcounterh11	32	0	r-	Performance Monitor High 11
hpmcounterh12	32	0	r-	Performance Monitor High 12
hpmcounterh13	32	0	r-	Performance Monitor High 13
hpmcounterh14	32	0	r-	Performance Monitor High 14
hpmcounterh15	32	0	r-	Performance Monitor High 15
hpmcounterh16	32	0	r-	Performance Monitor High 16
hpmcounterh17	32	0	r-	Performance Monitor High 17

hpmcounterh18	32	0	r-	Performance Monitor High 18
hpmcounterh19	32	0	r-	Performance Monitor High 19
hpmcounterh20	32	0	r-	Performance Monitor High 20
hpmcounterh21	32	0	r-	Performance Monitor High 21
hpmcounterh22	32	0	r-	Performance Monitor High 22
hpmcounterh23	32	0	r-	Performance Monitor High 23
hpmcounterh24	32	0	r-	Performance Monitor High 24
hpmcounterh25	32	0	r-	Performance Monitor High 25
hpmcounterh26	32	0	r-	Performance Monitor High 26
hpmcounterh27	32	0	r-	Performance Monitor High 27
hpmcounterh28	32	0	r-	Performance Monitor High 28
hpmcounterh29	32	0	r-	Performance Monitor High 29
hpmcounterh30	32	0	r-	Performance Monitor High 30
hpmcounterh31	32	0	r-	Performance Monitor High 31
lpstart0*	32	0	rw	HW Loop Start Address 0
lpend0*	32	0	rw	HW Loop End Address 0
lpcount0*	32	0	rw	HW Loop Count Address 0
lpstart1*	32	0	rw	HW Loop Start Address 1
lpend1*	32	0	rw	HW Loop End Address 1
lpcount1*	32	0	rw	HW Loop Count Address 1
uhartid*	32	0	r-	User Hardware Thread ID
privlv*	32	3	r-	Privilege Level
zfinx*	32	0	r-	ZFINX ISA
mvendorid	32	602	r-	Vendor ID
marchid	32	4	r-	Architecture ID
mimpid	32	0	r-	Implementation ID
mhartid	32	0	r-	Hardware Thread ID

Table 14.2: Registers at level 1, type:Hart group:Machine_Control_and_Status

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

14.1.3 Integration_support

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

Name	Bits	Initial-Hex	RW	Description
DM	8	0	rw	Debug mode active
commercial	8	0	r-	Commercial feature in use
ASYNCPE	8	0	r-	Asynchronous Event Pending & Enabled

Table 14.3: Registers at level 1, type:Hart group:Integration_support