OVP Guide to Using Processor Models

Model specific information for OpenHwGroup_CV32E40X

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

Author	Imperas Software Limited
Version	20230201.0
Filename	OVP_Model_Specific_Information_openhwgroup_riscv_CV32E40X.pdf
Created	1 February 2023
Status	OVP Standard Release

Copyright Notice

Right to Copy Documentation

Destination Control Statement

Disclaimer

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 CV32E40X 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)

In addition, the model has the following architectural extensions implicitly enabled (not shown in the misa CSR Extensions field):

misa bit 1: extension B (bit manipulation extension)

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

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

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

1.3.2 Enabling Other Extensions

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

misa bit 0: extension A (atomic instructions)

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

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

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

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

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

1.3.3 Disabling Extensions

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

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

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 0xffffff81. 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.

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

This variant implements the compressed extension with version specified in the References section of this document. Note that parameter "compress_version" can be used to select the required architecture version. See the following sections for detailed information about differences between each supported version.

1.5.1 Compressed Extension Parameters

Parameter "Zca" is used to specify that basic C extension instructions are present. By default, "Zca" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zcf" is used to specify that floating point load/store instructions are present. By default, "Zcf" is set to 0 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zcb" is used to specify that additional simple operation instructions are present. By default, "Zcb" is set to 1 in this variant. Updates to this parameter require a commercial product

license.

Parameter "Zcmb" is used to specify that load/store byte/half instructions are present. By default, "Zcmb" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zcmp" is used to specify that push/pop and double move instructions are present. By default, "Zcmp" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zcmpe" is used to specify that E-extension push/pop instructions are present. By default, "Zcmpe" is set to 0 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zcmt" is used to specify that table jump instructions are present. By default, "Zcmt" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "jvt_mask" is used to specify writable bits in the jvt CSR. By default, "jvt_mask" is set to 0xfffffc00 in this variant.

1.5.2 Legacy Version 1.10

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

1.5.3 Version 0.70.1

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

1.5.4 Version 0.70.5

Version 0.70.5, with these changes compared to version 0.70.1:

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

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

1.5.5 Version 1.0.0-RC5.7

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

- encodings of jt and jalt instructions changed.

- Zcmb and Zcmpe subsets removed.

1.6 Privileged Architecture

This variant implements the Privileged Architecture with version specified in the References section of this document. Note that parameter "priv_version" can be used to select the required architecture

version; see the following sections for detailed information about differences between each supported version.

1.6.1 Legacy Version 1.10

1.10 version of May 7 2017.

1.6.2 Version 20190608

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

- mcountinhibit CSR defined;

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

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

1.6.3 Version 20211203

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

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

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

- maximum number of PMP registers increased to 64;

- data endian is now configurable.

1.6.4 Version 1.12

Official 1.12 version, identical to 20211203.

1.6.5 Version master

Unstable master version, currently identical to 1.12.

1.7 Unprivileged Architecture

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

1.7.1 Legacy Version 2.2

2.2 version of May 7 2017.

1.7.2 Version 20191213

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

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

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

1.8 Bit-Manipulation Extension

This variant implements the Bit-Manipulation extension with version specified in the References section of this document. Note that parameter "bitmanip_version" can be used to select the required version of this extension. See section "Bit-Manipulation Extension Versions" for detailed information about differences between each supported version.

1.8.1 Bit-Manipulation Extension Parameters

Parameter "Zbb" is used to specify that the base instructions are present. By default, "Zbb" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zba" is used to specify that address calculation instructions are present. By default, "Zba" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zbc" is used to specify that carryless operation instructions are present. By default, "Zbc" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zbe" is used to specify that bit deposit/extract instructions are present. By default, "Zbe" is set to 0 in this variant. Updates to this parameter require a commercial product license. This parameter is ignored for version 1.0.0, which does not implement that subset.

Parameter "Zbf" is used to specify that bit field place instructions are present. By default, "Zbf" is set to 0 in this variant. Updates to this parameter require a commercial product license. This parameter is ignored for version 1.0.0, which does not implement that subset.

Parameter "Zbm" is used to specify that bit matrix operation instructions are present. By default, "Zbm" is set to 0 in this variant. Updates to this parameter require a commercial product license. This parameter is ignored for version 1.0.0, which does not implement that subset.

Parameter "Zbp" is used to specify that permutation instructions are present. By default, "Zbp" is set to 0 in this variant. Updates to this parameter require a commercial product license. This parameter is ignored for version 1.0.0, which does not implement that subset.

Parameter "Zbr" is used to specify that CRC32 instructions are present. By default, "Zbr" is set to 0 in this variant. Updates to this parameter require a commercial product license. This parameter

is ignored for version 1.0.0, which does not implement that subset.

Parameter "Zbs" is used to specify that single bit instructions are present. By default, "Zbs" is set to 1 in this variant. Updates to this parameter require a commercial product license.

Parameter "Zbt" is used to specify that ternary instructions are present. By default, "Zbt" is set to 0 in this variant. Updates to this parameter require a commercial product license. This parameter is ignored for version 1.0.0, which does not implement that subset.

1.8.2 Bit-Manipulation Extension Versions

The Bit-Manipulation Extension 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 previous versions of the specification. The differing features of these are listed below, in chronological order.

1.8.3 Version 0.90

Stable 0.90 version of June 10 2019.

1.8.4 Version 0.91

Stable 0.91 version of August 29 2019, with these changes compared to version 0.90:

- change encodings of bmatxor, grev, grevw, grevi and greviw;

- add gorc, gorcw, gorci, gorciw, bfp and bfpw instructions.

1.8.5 Version 0.92

Stable 0.92 version of November 8 2019, with these changes compared to version 0.91:

- add packh, packu and packuw instructions;

- add sext.b and sext.h instructions;

- change encoding and behavior of bfp and bfpw instructions;

- change encoding of bdep and bdepw instructions.

1.8.6 Version 0.93-draft

Draft 0.93 version of January 29 2020, with these changes compared to version 0.92:

- add sh1add, sh2add, sh3add, sh1addu, sh2addu and sh3addu instructions;

- move slo, sloi, sro and sroi to Zbp subset;

- add orc16 to Zbb subset.

1.8.7 Version 0.93

Stable 0.93 version of January 10 2021, with these changes compared to version 0.93-draft:

- assignments of instructions to Z extension groups changed;

- exchange encodings of max and minu instructions;

- add xperm.[nbhw] instructions;

- instructions named *u.w renamed to *.uw;

- instruction add.uw zero-extends argument rs1, not rs2;

- instructions named sb* renamed to b*;

- instructions named pcnt* renamed to cpop*;

- instructions subu.w, addiwu, addwu, subwu, clmulw, clmulrw and clmulhw removed;

- instructions slo, sro, sloi, sroi, slow, srow, sloiw and sroiw removed from all Z extension groups and are therefore never implemented;

- instructions bext/bdep renamed to bcompress/bdecompress (this change is documented under the draft 0.94 version but is required to resolve an instruction name conflict introduced by instruction renames above);

1.8.8 Version 0.94

Stable 0.94 version of January 20 2021, with these changes compared to version 0.93:

- instructions bset[i]w, bclr[i]w, binv[i]w and bextw removed.

1.8.9 Version 1.0.0

Stable 1.0.0 version of June 6 2021, with these changes compared to version 0.94:

- instructions with immediate shift operands now follow base architecture semantics to determine operand legality instead of masking to XLEN-1;

- only subsets Zba, Zbb, Zbc and Zbs may be enabled;

- if the B extension is present, it is implicitly always enabled and not subject to control by misa.B, which is zero.

1.8.10 Version master

Unstable master version, with these changes compared to version 1.0.0:

- any subset may be enabled;

- xperm.n, xperm.b, xperm.h and xperm.w instructions renamed xperm4, xperm8, xperm16 and xperm32.

1.9 Other Extensions

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

1.9.1 Zmmul

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

1.9.2 Zicsr

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

1.9.3 Zifencei

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

1.9.4 Zicbom

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

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

1.9.5 Zicbop

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

1.9.6 Zicboz

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

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

1.9.7 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.

1.9.8 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.

1.10 CLIC

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

1.11 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.

1.12 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.

1.13 Debug Mode

The model can be configured to implement Debug mode using parameter "debug_mode". This implements features described in Chapter 4 of the RISC-V External Debug Support specification with version specified by parameter "debug_version" (see References). Some aspects of this mode are not defined in the specification because they are implementation-specific; the model provides infrastructure to allow implementation of a Debug Module using a custom harness. Features added are described below.

Parameter "debug_mode" can be used to specify three different behaviors, as follows:

1. If set to value "vector", then operations that would cause entry to Debug mode result in the processor jumping to the address specified by the "debug_address" parameter. It will execute at this address, in Debug mode, until a "dret" instruction causes return to non-Debug mode. Any exception generated during this execution will cause a jump to the address specified by the "dexc_address" parameter.

2. If set to value "interrupt", then operations that would cause entry to Debug mode result in the processor simulation call (e.g. opProcessorSimulate) returning, with a stop reason of OP_SR_INTERRUPT. In this usage scenario, the Debug Module is implemented in the simulation harness.

3. If set to value "halt", then operations that would cause entry to Debug mode result in the processor halting. Depending on the simulation environment, this might cause a return from the simulation call with a stop reason of OP_SR_HALT, or debug mode might be implemented by another platform component which then restarts the debugged processor again.

1.13.1 Debug State Entry

The specification does not define how Debug mode is implemented. In this model, Debug mode is enabled by a Boolean pseudo-register, "DM". When "DM" is True, the processor is in Debug mode. When "DM" is False, mode is defined by "mstatus" in the usual way.

Entry to Debug mode can be performed in any of these ways:

1. By writing True to register "DM" (e.g. using opProcessorRegWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate), dcsr cause will be reported as trigger;

2. By writing a 1 then 0 to net "haltreq" (using opNetWrite) followed by simulation of at least one cycle (e.g. using opProcessorSimulate);

3. By writing a 1 to net "resethaltreq" (using opNetWrite) while the "reset" signal undergoes a negedge transition, followed by simulation of at least one cycle (e.g. using opProcessorSimulate);

4. By executing an "ebreak" instruction when Debug mode entry for the current processor mode is enabled by dcsr.ebreakm, dcsr.ebreaks or dcsr.ebreaku.

In all cases, the processor will save required state in "dpc" and "dcsr" and then perform actions described above, depending in the value of the "debug_mode" parameter.

1.13.2 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.

1.13.3 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.

1.13.4 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.

1.13.5 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.

1.13.6 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.

1.13.7 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).

1.13.8 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.

1.13.9 Version 0.13.2-DRAFT

0.13.2-DRAFT version of March 22 2019.

1.13.10 Version 0.14.0-DRAFT

0.14.0-DRAFT version of November 6 2020.

1.13.11 Version 1.0.0-STABLE

1.0.0-STABLE version of February 9 2022.

1.13.12 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.

1.14 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).

1.14.1 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.

1.14.2 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. In this variant, "tinfo" is 0x60.

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 0x000d.

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

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

Parameter "mcontext_undefined" is used to specify whether the "mcontext" register is undefined, in which case accesses to it trap to Machine mode. In this variant, "mcontext_undefined" is 1.

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 1.

Parameter "mscontext_undefined" is used to specify whether the "mscontext" register is undefined, in which case accesses to it trap to Machine mode. In this variant, "mscontext_undefined" is 1.

Parameter "amo_trigger" is used to specify whether load/store triggers are activated for AMO instructions. In this variant, "amo_trigger" is 0.

Parameter "no_hit" is used to specify whether the "hit" bit in tdata1 is unimplemented. In this variant, "no_hit" is 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.

1.15 Debug Mask

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

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

Value 0x004: enable debugging of interrupt state.

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

1.16 Integration Support

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

1.16.1 CSR Register External Implementation

If parameter "enable_CSR_bus" is True, an artifact 16-bit bus "CSR" is enabled. Slave callbacks installed on this bus can be used to implement modified CSR behavior (use opBusSlaveNew or icmMapExternalMemory, depending on the client API). A CSR with index 0xABC is mapped on

the bus at address 0xABC0; as a concrete example, implementing CSR "time" (number 0xC01) externally requires installation of a read callback at address 0xC010 on the CSR bus.

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

1.17 Instruction Disassembly

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

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

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

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

1.18 Limitations

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

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

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

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

THIS IS A STARTING POINT AS THE SPECS DEVELOP More detail to be added once confirmed Awaiting information for: PMA (bespoke model requiring specification), ISA_P (exists in other models, to be added when ratified).Default Bit Manipulation setting = B_NONE

in order to get ZBA_ZBB_ZBS

–override root/cpu/Zba=1

–override root/cpu/Zbb=1

–override root/cpu/Zbs=1

in order to get ZBA_ZBB_ZBC_ZBS

–override root/cpu/Zba=1

–override root/cpu/Zbb=1

–override root/cpu/Zbc=1

–override root/cpu/Zbs=1

1.19 Verification

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

Also reference tests have been used from various sources including:

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

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

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

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

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

http://valtrix.in/sting from Valtrix

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

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

1.20 References

The Model details are based upon the following specifications:

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

RISC-V Instruction Set Manual, Volume II: Privileged Architecture (Privileged Architecture Version 1.12, equivalent to 20211203)

RISC-V “C” Compressed Extension (Compressed Architecture Version 0.70.5)

RISC-V “B” Bit Manipulation Extension (Bit Manipulation Architecture Version 1.0.0)

RISC-V External Debug Support (RISC-V External Debug Support Version 1.0.0-STABLE)

RTL https://docs.openhwgroup.org/projects/cv32e40x-user-manual/en/latest/preface.html#release-

0-4-0

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:

- Static 16 entry PMA

- Custom bus fault extensions

- WFE instruction and wu_wfe_i signal

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/PMA_NUM_REGIONS

TBD

2.2 PMA Extension Status

The PMA extension is not enabled on this variant. Use parameter extension/PMA_NUM_REGIONS to enable it.

2.3 Custom External Exceptions

Custom external exceptions are enabled on this variant. The following table lists the net ports that will trigger the corresponding exception when a rising edge is detected on a net connected to the port:

Name	Description
InstructionBusFault	External Instruction Bus Fault Exception
wu_wfe_i	Wake from WFE state

Table 2.1: External Exception Net Ports

Chapter 3

Configuration

3.1 Location

This model's VLNV is openhwgroup.ovpworld.org/processor/CVE4X/1.0.
The model source is usually at:
$IMPERAS_HOME/ImperasLib/source/openhwgroup.ovpworld.org/processor/CVE4X/1.0
The model binary is usually at:
$IMPERAS_HOME/lib/$IMPERAS_ARCH/ImperasLib/openhwgroup.ovpworld.org/processor/CVE4X/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
CV32E40X	(described in this document)
CV32E40X_DEV	

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)
deferint	input	optional	Artifact signal causing interrupts to be held off when high
InstructionBusFault	input	optional	External Instruction Bus Fault Exception
wu_wfe_i	input	optional	Wake from WFE state

Table 7.1: Net Ports

Chapter 8

FIFO Ports

This model has no FIFO ports.

Chapter 9

Formal Parameters

Name	Type	Description
Fundamental		
variant	Enumeration	Selects variant (either a generic UISA or a specific model)
user_version	Enumeration	Specify required User Architecture version (2.2, 2.3, 20190305 or 20191213)
priv_version	Enumeration	Specify required Privileged Architecture version (1.10, 1.11, 20190405, 20190608, 20211203, 1.12 or master)
Smepmp_version	Enumeration	Specify required Smepmp Architecture version (none, 0.9.5 or 1.0)
endian	Endian	Model endian
enable_expanded	Boolean	Specify that 48-bit and 64-bit expanded instructions are supported
endianFixed	Boolean	Specify that data endianness is fixed (mstatus.{MBE,SBE,UBE} fields are read-only)
misa_MXL	Uns32	Override default value of misa.MXL
misa_Extensions	Uns32	Override default value of misa.Extensions
add_Extensions	String	Add extensions specified by letters to misa.Extensions (for example, specify "VD" to add V and D features)
sub_Extensions	String	Remove extensions specified by letters from misa.Extensions (for example, specify "VD" to remove V and D features)
misa_Extensions_mask	Uns32	Override mask of writable bits in misa.Extensions
add_Extensions_mask	String	Add extensions specified by letters to mask of writable bits in misa.Extensions (for example, specify "VD" to add V and D features)
sub_Extensions_mask	String	Remove extensions specified by letters from mask of writable bits in misa.Extensions (for example, specify "VD" to remove V and D features)
add_implicit_Extensions	String	Add extensions specified by letters to implicitly-present extensions not visible in misa.Extensions
sub_implicit_Extensions	String	Remove extensions specified by letters from implicitly-present extensions not visible in misa.Extensions
Bit_Manipulation_Extension		
bitmanip_version	Enumeration	Specify required Bit Manipulation Architecture version (0.90, 0.91, 0.92, 0.93-draft, 0.93, 0.94, 1.0.0 or master)
Zba	Boolean	Specify that Zba is implemented
Zbb	Boolean	Specify that Zbb is implemented
Zbc	Boolean	Specify that Zbc is implemented
Zbe	Boolean	Specify that Zbe is implemented (ignored if version 1.0.0)
Zbf	Boolean	Specify that Zbf is implemented (ignored if version 1.0.0)
Zbm	Boolean	Specify that Zbm is implemented (ignored if version 1.0.0)
Zbp	Boolean	Specify that Zbp is implemented (ignored if version 1.0.0)
Zbr	Boolean	Specify that Zbr is implemented (ignored if version 1.0.0)
Zbs	Boolean	Specify that Zbs is implemented
Zbt	Boolean	Specify that Zbt is implemented (ignored if version 1.0.0)
Compressed_Extension		

compress_version	Enumeration	Specify required Compressed Architecture version (legacy, 0.70.1, 0.70.5 or 1.0.0-RC5.7)
Zca	Boolean	Specify that Zca is implemented
Zcb	Boolean	Specify that Zcb is implemented
Zcf	Boolean	Specify that Zcf is implemented
Zcmb	Boolean	Specify that Zcmb is implemented
Zcmp	Boolean	Specify that Zcmp is implemented
Zcmpe	Boolean	Specify that Zcmpe is implemented
Zcmt	Boolean	Specify that Zcmt is implemented
Debug_Extension		
debug_version	Enumeration	Specify required Debug Architecture version (0.13.2-DRAFT, 0.14.0-DRAFT, 1.0.0-STABLE or 1.0-STABLE)
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
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, jump_to_dexc_address or trap_to_dexc_address)
debug_priority	Enumeration	Specify relative priorities of simultaneous debug events (asxh, ashx, ahsx, hasx, original, PR693 or halt_not_step)
dcsr_ebreak_mask	Uns32	Specify mask of dcsr.ebreak fields that reset to 1 (ebreak instructions enter Debug mode)
Interrupts_Exceptions		
rnmi_version	Enumeration	Specify required RNMI Architecture version (none, 0.2.1 or 0.4)
mtvec_is_ro	Boolean	Specify whether mtvec CSR is read-only
tvec_align	Uns32	Specify hardware-enforced alignment of mtvec/stvec/utvec when Vectored interrupt mode enabled
ecode_mask	Uns64	Specify hardware-enforced mask of writable bits in xcause.ExceptionCode
ecode_nmi	Uns64	Specify xcause.ExceptionCode for NMI
nmi_is_latched	Boolean	Specify whether NMI input is latched on rising edge (if False, it is level-sensitive)
tval_zero	Boolean	Specify whether mtval/stval/utval are hard wired to zero
tval_zero_ebreak	Boolean	Specify whether mtval/stval/utval are set to zero by an ebreak
tval_ii_code	Boolean	Specify whether mtval/stval contain faulting instruction bits on illegal instruction exception
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
Memory		
push_pop_constraint	Enumeration	Specify memory constraint for PUSH/POP instructions (none, user1 or user2)
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
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>
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
jvt_mask	Uns64	Specify hardware-enforced mask of writable bits in Zcmt jvt 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
envcfg_mask	Uns64	Specify hardware-enforced mask of writable bits in envcfg registers
mtvec_sext	Boolean	Specify whether mtvec is sign-extended from most-significant bit
Trigger		
tinfo_undefined	Boolean	Specify that the tinfo CSR is undefined
tcontrol_undefined	Boolean	Specify that the tcontrol CSR is undefined
mcontext_undefined	Boolean	Specify that the mcontext CSR is undefined
scontext_undefined	Boolean	Specify that the scontext CSR is undefined
mscontext_undefined	Boolean	Specify that the mscontext CSR is undefined (Debug Version 0.14.0 and later)
amo_trigger	Boolean	Specify whether AMO load/store operations activate triggers
no_hit	Boolean	Specify that tdata1.hit is unimplemented
trigger_num	Uns32	Specify the number of implemented hardware triggers
tinfo	Uns32	Override tinfo register (for all triggers)
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
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)
mconfigptr	Uns64	Override mconfigptr register
mtvec	Uns64	Override mtvec register
mseccfg	Uns64	Override mseccfg 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
PMA_NUM_REGIONS	Uns32	number of PMA regions
word_addr_low0	Uns32	PMA region 0 low bound
word_addr_high0	Uns32	PMA region 0 high bound
main0	Boolean	PMA region 0 main
bufferable0	Boolean	PMA region 0 bufferable
cacheable0	Boolean	PMA region 0 cacheable
atomic0	Boolean	PMA region 0 atomic
word_addr_low1	Uns32	PMA region 1 low bound
word_addr_high1	Uns32	PMA region 1 high bound
main1	Boolean	PMA region 1 main
bufferable1	Boolean	PMA region 1 bufferable
cacheable1	Boolean	PMA region 1 cacheable
atomic1	Boolean	PMA region 1 atomic

word_addr_low2	Uns32	PMA region 2 low bound
word_addr_high2	Uns32	PMA region 2 high bound
main2	Boolean	PMA region 2 main
bufferable2	Boolean	PMA region 2 bufferable
cacheable2	Boolean	PMA region 2 cacheable
atomic2	Boolean	PMA region 2 atomic
word_addr_low3	Uns32	PMA region 3 low bound
word_addr_high3	Uns32	PMA region 3 high bound
main3	Boolean	PMA region 3 main
bufferable3	Boolean	PMA region 3 bufferable
cacheable3	Boolean	PMA region 3 cacheable
atomic3	Boolean	PMA region 3 atomic
word_addr_low4	Uns32	PMA region 4 low bound
word_addr_high4	Uns32	PMA region 4 high bound
main4	Boolean	PMA region 4 main
bufferable4	Boolean	PMA region 4 bufferable
cacheable4	Boolean	PMA region 4 cacheable
atomic4	Boolean	PMA region 4 atomic
word_addr_low5	Uns32	PMA region 5 low bound
word_addr_high5	Uns32	PMA region 5 high bound
main5	Boolean	PMA region 5 main
bufferable5	Boolean	PMA region 5 bufferable
cacheable5	Boolean	PMA region 5 cacheable
atomic5	Boolean	PMA region 5 atomic
word_addr_low6	Uns32	PMA region 6 low bound
word_addr_high6	Uns32	PMA region 6 high bound
main6	Boolean	PMA region 6 main
bufferable6	Boolean	PMA region 6 bufferable
cacheable6	Boolean	PMA region 6 cacheable
atomic6	Boolean	PMA region 6 atomic
word_addr_low7	Uns32	PMA region 7 low bound
word_addr_high7	Uns32	PMA region 7 high bound
main7	Boolean	PMA region 7 main
bufferable7	Boolean	PMA region 7 bufferable
cacheable7	Boolean	PMA region 7 cacheable
atomic7	Boolean	PMA region 7 atomic
word_addr_low8	Uns32	PMA region 8 low bound
word_addr_high8	Uns32	PMA region 8 high bound
main8	Boolean	PMA region 8 main
bufferable8	Boolean	PMA region 8 bufferable
cacheable8	Boolean	PMA region 8 cacheable
atomic8	Boolean	PMA region 8 atomic
word_addr_low9	Uns32	PMA region 9 low bound
word_addr_high9	Uns32	PMA region 9 high bound
main9	Boolean	PMA region 9 main
bufferable9	Boolean	PMA region 9 bufferable
cacheable9	Boolean	PMA region 9 cacheable
atomic9	Boolean	PMA region 9 atomic
word_addr_low10	Uns32	PMA region 10 low bound
word_addr_high10	Uns32	PMA region 10 high bound
main10	Boolean	PMA region 10 main
bufferable10	Boolean	PMA region 10 bufferable
cacheable10	Boolean	PMA region 10 cacheable
atomic10	Boolean	PMA region 10 atomic
word_addr_low11	Uns32	PMA region 11 low bound
word_addr_high11	Uns32	PMA region 11 high bound

main11	Boolean	PMA region 11 main
bufferable11	Boolean	PMA region 11 bufferable
cacheable11	Boolean	PMA region 11 cacheable
atomic11	Boolean	PMA region 11 atomic
word_addr_low12	Uns32	PMA region 12 low bound
word_addr_high12	Uns32	PMA region 12 high bound
main12	Boolean	PMA region 12 main
bufferable12	Boolean	PMA region 12 bufferable
cacheable12	Boolean	PMA region 12 cacheable
atomic12	Boolean	PMA region 12 atomic
word_addr_low13	Uns32	PMA region 13 low bound
word_addr_high13	Uns32	PMA region 13 high bound
main13	Boolean	PMA region 13 main
bufferable13	Boolean	PMA region 13 bufferable
cacheable13	Boolean	PMA region 13 cacheable
atomic13	Boolean	PMA region 13 atomic
word_addr_low14	Uns32	PMA region 14 low bound
word_addr_high14	Uns32	PMA region 14 high bound
main14	Boolean	PMA region 14 main
bufferable14	Boolean	PMA region 14 bufferable
cacheable14	Boolean	PMA region 14 cacheable
atomic14	Boolean	PMA region 14 atomic
word_addr_low15	Uns32	PMA region 15 low bound
word_addr_high15	Uns32	PMA region 15 high bound
main15	Boolean	PMA region 15 main
bufferable15	Boolean	PMA region 15 bufferable
cacheable15	Boolean	PMA region 15 cacheable
atomic15	Boolean	PMA region 15 atomic

Table 9.2: Parameters for extension_CVE4X

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 bitmanip_version

Set to this value	Description
0.90	Bit Manipulation Architecture Version v0.90-20190610
0.91	Bit Manipulation Architecture Version v0.91-20190829
0.92	Bit Manipulation Architecture Version v0.92-20191108
0.93-draft	Bit Manipulation Architecture Version 0.93-draft-20200129
0.93	Bit Manipulation Architecture Version v0.93-20210110
0.94	Bit Manipulation Architecture Version v0.94-20210120
1.0.0	Bit Manipulation Architecture Version 1.0.0
master	Bit Manipulation Master Branch as of commit 1f56afe (this is subject to change)

Table 9.5: Values for Parameter bitmanip_version

9.2.4 Parameter compress_version

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

Table 9.6: Values for Parameter compress_version

9.2.5 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 9.7: Values for Parameter debug_version

9.2.6 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 9.8: Values for Parameter rnmi_version

9.2.7 Parameter Smepmp_version

Set to this value	Description
none	Smepmp not implemented
0.9.5	Smepmp version 0.9.5 (deprecated and identical to 1.0)
1.0	Smepmp version 1.0

Table 9.9: Values for Parameter Smepmp_version

9.2.8 Parameter debug_mode

Set to this value	Description
none	Debug mode not implemented
vector	Debug mode implemented by execution at vector
interrupt	Debug mode implemented by interrupt
halt	Debug mode implemented by halt

Table 9.10: Values for Parameter debug_mode

9.2.9 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 9.11: Values for Parameter debug_eret_mode

9.2.10 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 9.12: Values for Parameter debug_priority

9.2.11 Parameter push_pop_constraint

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

Table 9.13: Values for Parameter push_pop_constraint

9.3 Parameter values

These are the current parameter values.

Name	Value
Fundamental	
variant	CV32E40X

user_version	20191213
priv_version	1.12
Smepmp_version	none
endian	none
enable_expanded	F
endianFixed	F
misa_MXL	1
misa_Extensions	0x1106
add_Extensions	
sub_Extensions	
misa_Extensions_mask	0
add_Extensions_mask	
sub_Extensions_mask	
add_implicit_Extensions	
sub_implicit_Extensions	
Bit_Manipulation_Extension	
bitmanip_version	1.0.0
Zba	T
Zbb	T
Zbc	T
Zbe	T
Zbf	T
Zbm	T
Zbp	T
Zbr	T
Zbs	T
Zbt	T
Compressed_Extension	
compress_version	0.70.5
Zca	T
Zcb	T
Zcf	F
Zcmb	T
Zcmp	T
Zcmpe	F
Zcmt	T
Debug_Extension	
debug_version	1.0.0-STABLE
debug_mode	vector
debug_address	0x1a110800
dexc_address	0x1a111000
debug_eret_mode	trap_to_dexc_address
debug_priority	ahsx
dcsr_ebreak_mask	0
Interrupts_Exceptions	

rnmi_version	none
mtvec_is_ro	F
tvec_align	0
ecode_mask	0x7ff
ecode_nmi	0
nmi_is_latched	T
tval_zero	T
tval_zero_ebreak	F
tval_ii_code	F
reset_address	0
nmi_address	0
CLINT_address	0
local_int_num	16
unimp_int_mask	0
force_mideleg	0
no_ideleg	0
no_edeleg	0
external_int_id	F
Memory	
push_pop_constraint	user1
unaligned_low_pri	F
unaligned	T
Simulation_Artifact	
use_hw_reg_names	F
no_pseudo_inst	F
show_c_prefix	F
verbose	F
traceVolatile	F
enable_CSR_bus	F
CSR_remap	
Instruction_CSR_Behavior	
wfi_is_nop	F
wfi_resume_not_trap	F
TW_time_limit	0
counteren_mask	0xffffffff
noinhibit_mask	0
mcycle_undefined	F
time_undefined	T
minstret_undefined	F
mhpmcounter_undefined	F
CSR_Masks	
mtvec_mask	0xffffff81
jvt_mask	0xfffffc00
tdata1_mask	0xffffffffffffffff
mip_mask	0x337

envcfg_mask	0
mtvec_sext	F
Trigger	
tinfo_undefined	F
tcontrol_undefined	F
mcontext_undefined	T
scontext_undefined	T
mscontext_undefined	T
amo_trigger	F
no_hit	T
trigger_num	1
tinfo	96
trigger_match	13
mcontext_bits	0
mvalue_bits	0
mcontrol_maskmax	0
PMP Configuration	
PMP_grain	0
PMP_registers	0
PMP_max_page	0
PMP_decompose	F
PMP_undefined	T
PMP_maskparams	F
PMP_initialparams	F
Other_Extensions	
Smstateen	F
Zihintntl	F
Zicond	F
Zicsr	T
Zifencei	T
Zicbom	F
Zicbop	F
Zicboz	F
Zawrs	F
Zmmul	F
CSR_Defaults	
mvendorid	0x602
marchid	20
mimpid	0
mhartid	0
mconfigptr	0
mtvec	1
mseccfg	0
Fast_Interrupt	
CLICLEVELS	0

AIA_Interrupts	
Smaia	F
extension_CVE4X	
debug*	F
mcountinhibit_reset*	13
tdata1_reset*	0x68001044
dcsr_reset*	0x40000013
PMA_NUM_REGIONS*	0
word_addr_low0*	0
word_addr_high0*	0
main0*	F
bufferable0*	F
cacheable0*	F
atomic0*	F
word_addr_low1*	0
word_addr_high1*	0
main1*	F
bufferable1*	F
cacheable1*	F
atomic1*	F
word_addr_low2*	0
word_addr_high2*	0
main2*	F
bufferable2*	F
cacheable2*	F
atomic2*	F
word_addr_low3*	0
word_addr_high3*	0
main3*	F
bufferable3*	F
cacheable3*	F
atomic3*	F
word_addr_low4*	0
word_addr_high4*	0
main4*	F
bufferable4*	F
cacheable4*	F
atomic4*	F
word_addr_low5*	0
word_addr_high5*	0
main5*	F
bufferable5*	F
cacheable5*	F
atomic5*	F
word_addr_low6*	0

word_addr_high6*	0
main6*	F
bufferable6*	F
cacheable6*	F
atomic6*	F
word_addr_low7*	0
word_addr_high7*	0
main7*	F
bufferable7*	F
cacheable7*	F
atomic7*	F
word_addr_low8*	0
word_addr_high8*	0
main8*	F
bufferable8*	F
cacheable8*	F
atomic8*	F
word_addr_low9*	0
word_addr_high9*	0
main9*	F
bufferable9*	F
cacheable9*	F
atomic9*	F
word_addr_low10*	0
word_addr_high10*	0
main10*	F
bufferable10*	F
cacheable10*	F
atomic10*	F
word_addr_low11*	0
word_addr_high11*	0
main11*	F
bufferable11*	F
cacheable11*	F
atomic11*	F
word_addr_low12*	0
word_addr_high12*	0
main12*	F
bufferable12*	F
cacheable12*	F
atomic12*	F
word_addr_low13*	0
word_addr_high13*	0
main13*	F
bufferable13*	F

cacheable13*	F
atomic13*	F
word_addr_low14*	0
word_addr_high14*	0
main14*	F
bufferable14*	F
cacheable14*	F
atomic14*	F
word_addr_low15*	0
word_addr_high15*	0
main15*	F
bufferable15*	F
cacheable15*	F
atomic15*	F

Table 9.14: Parameter values

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

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
InstructionAddressMisaligned	0	Fetch from unaligned address
InstructionAccessFault	1	No access permission for fetch
IllegalInstruction	2	Undecoded, unimplemented or disabled instruction
Breakpoint	3	EBREAK instruction executed
LoadAddressMisaligned	4	Load from unaligned address
LoadAccessFault	5	No access permission for load
StoreAMOAddressMisaligned	6	Store/atomic memory operation at unaligned address
StoreAMOAccessFault	7	No access permission for store/atomic memory operation
EnvironmentCallFromMMode	11	ECALL instruction executed in Machine mode
InstructionPageFault	12	Page fault at fetch address
LoadPageFault	13	Page fault at load address
StoreAMOPageFault	15	Page fault at store/atomic memory operation address
InstructionBusFault	24	Instruction Bus Fault
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 5 commands.
This level in the model hierarchy has 3 register groups:

Group name	Registers
Core	33
Machine_Control_and_Status	180
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
-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

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
jvt	32	0	rw	Table Jump Base and Control
mstatus	32	1800	rw	Machine Status
misa	32	40001104	rw	ISA and Extensions
mie	32	0	rw	Machine Interrupt Enable
mtvec*	32	1	rw	Machine Trap-Vector Base-Address
mstatush	32	0	rw	Machine Status High
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	68001044	rw	Trigger Data 1
tdata2	32	0	rw	Trigger Data 2
tdata3	32	0	rw	Trigger Data 3
tinfo	32	60	rw	Trigger Info
tcontrol*	32	0	rw	Trigger Control
dcsr	32	40000013	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
mvendorid	32	602	r-	Vendor ID
marchid	32	14	r-	Architecture ID
mimpid	32	0	r-	Implementation ID
mhartid	32	0	r-	Hardware Thread ID
mconfigptr	32	0	r-	Configuration Data Structure

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