



OVP Guide to Using Processor Models

Model specific information for SiFive_X280

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



Author	Imperas Software Limited
Version	20211118.0
Filename	OVP_Model_Specific_Information_sifive_riscv_X280.pdf
Created	31 December 2021
Status	OVP Standard Release

Copyright Notice

Copyright (c) 2021 Imperas Software Limited. All rights reserved. This software and documentation contain information that is the property of Imperas Software Limited. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Imperas Software Limited, or as expressly provided by the license agreement.

Right to Copy Documentation

The license agreement with Imperas permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the readers responsibility to determine the applicable regulations and to comply with them.

Disclaimer

IMPERAS SOFTWARE LIMITED, AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Model Release Status

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

Contents

1	Overview	1
1.1	Description	1
1.2	Licensing	1
1.3	Extensions	2
1.3.1	Extensions Enabled by Default	2
1.3.2	Disabling Extensions	2
1.4	General Features	3
1.4.1	Multicore Features	3
1.4.2	mtvec CSR	3
1.4.3	stvec CSR	3
1.4.4	Reset	3
1.4.5	NMI	3
1.4.6	WFI	4
1.4.7	cycle CSR	4
1.4.8	time CSR	4
1.4.9	instret CSR	4
1.4.10	hpmcounter CSRs	4
1.4.11	Virtual Memory	4
1.4.12	Unaligned Accesses	5
1.4.13	PMP	5
1.4.14	LR/SC Granule	5
1.5	Compressed Extension	5
1.6	Floating Point Features	6
1.7	Privileged Architecture	6
1.7.1	Legacy Version 1.10	6
1.7.2	Version 20190608	7
1.7.3	Version master	7
1.8	Unprivileged Architecture	7
1.8.1	Legacy Version 2.2	7
1.8.2	Version 20191213	7
1.9	Bit-Manipulation Extension	8
1.9.1	Bit-Manipulation Extension Parameters	8
1.9.2	Bit-Manipulation Extension Versions	8
1.9.3	Version 0.90	9
1.9.4	Version 0.91	9
1.9.5	Version 0.92	9
1.9.6	Version 0.93-draft	9

1.9.7	Version 0.93	9
1.9.8	Version 0.94	10
1.9.9	Version 1.0.0	10
1.9.10	Version master	10
1.10	Vector Extension	10
1.10.1	Vector Extension Parameters	10
1.10.2	Vector Extension Features	12
1.10.3	Vector Extension Versions	12
1.10.4	Version 0.7.1-draft-20190605	12
1.10.5	Version 0.7.1-draft-20190605+	12
1.10.6	Version 0.8-draft-20190906	12
1.10.7	Version 0.8-draft-20191004	13
1.10.8	Version 0.8-draft-20191117	13
1.10.9	Version 0.8-draft-20191118	13
1.10.10	Version 0.8	14
1.10.11	Version 0.9	14
1.10.12	Version 1.0-draft-20210130	14
1.10.13	Version 1.0-rc1-20210608	15
1.10.14	Version master	15
1.11	Other Extensions	15
1.11.1	Zmmul	15
1.11.2	Zicsr	15
1.11.3	Zifencei	16
1.11.4	Zicbom	16
1.11.5	Zicbop	16
1.11.6	Zicboz	16
1.11.7	Svnapot	16
1.11.8	Svpbmt	16
1.11.9	Svinval	17
1.12	Load-Reserved/Store-Conditional Locking	17
1.13	Active Atomic Operation Indication	17
1.14	Interrupts	18
1.15	Debug Mode	18
1.15.1	Debug State Entry	19
1.15.2	Debug State Exit	19
1.15.3	Debug Registers	20
1.15.4	Debug Mode Execution	20
1.15.5	Debug Single Step	20
1.15.6	Debug Ports	20
1.16	Debug Mask	21
1.17	Integration Support	21
1.17.1	CSR Register External Implementation	21
1.17.2	LR/SC Active Address	21
1.17.3	Page Table Walk Introspection	21
1.17.4	Artifact Register “fflags.i”	22
1.18	Limitations	22
1.19	Verification	22
1.20	References	23

2	SiFive-Specific Extensions	24
2.1	SiFive-Specific CSRs	24
2.1.1	bpm (csr num 0x7c0)	24
2.1.2	featureDisable (csr num 0x7c1)	24
2.2	SiFive-Specific Instructions	24
2.2.1	CFLUSH.D.L1	25
2.2.2	CDISCARD.D.L1	25
2.2.3	CEASE	25
3	SiFive-Specific Extensions	26
3.1	SiFive-Specific CSRs	26
3.1.1	bpm (csr num 0x7c0)	26
3.1.2	featureDisable (csr num 0x7c1)	26
3.2	SiFive-Specific Instructions	26
3.2.1	CFLUSH.D.L1	27
3.2.2	CDISCARD.D.L1	27
3.2.3	CEASE	27
4	Configuration	28
4.1	Location	28
4.2	GDB Path	28
4.3	Semi-Host Library	28
4.4	Processor Endian-ness	28
4.5	QuantumLeap Support	28
4.6	Processor ELF code	28
5	All Variants in this model	29
6	Bus Master Ports	30
7	Bus Slave Ports	31
8	Net Ports	32
9	FIFO Ports	34
10	Formal Parameters	35
10.1	Extension Parameters	38
10.2	Parameters with enumerated types	39
10.2.1	Parameter user_version	39
10.2.2	Parameter priv_version	39
10.2.3	Parameter vector_version	39
10.2.4	Parameter bitmanip_version	39
10.2.5	Parameter rnmi_version	40
10.2.6	Parameter fp16_version	40
10.2.7	Parameter mstatus_fs_mode	40
10.2.8	Parameter debug_mode	40
10.2.9	Parameter Zfinx_version	40
10.2.10	Parameter Zcea_version	41

10.2.11 Parameter <code>Zceb_version</code>	41
10.2.12 Parameter <code>Zcee_version</code>	41
10.3 Parameter values	41
11 Execution Modes	46
12 Exceptions	47
13 Hierarchy of the model	49
13.1 Level 1: SMP	49
13.2 Level 2: Hart	49
14 Model Commands	50
14.1 Level 1: SMP	50
14.1.1 <code>isync</code>	50
14.1.2 <code>itrace</code>	50
14.2 Level 2: Hart	51
14.2.1 <code>dumpTLB</code>	51
14.2.1.1 Argument description	51
14.2.2 <code>getCSRIndex</code>	51
14.2.3 <code>isync</code>	51
14.2.4 <code>itrace</code>	51
14.2.5 <code>listCSRs</code>	51
14.2.5.1 Argument description	51
15 Registers	52
15.1 Level 1: SMP	52
15.2 Level 2: Hart	52
15.2.1 Core	52
15.2.2 Floating_point	53
15.2.3 Vector	53
15.2.4 User_Control_and_Status	54
15.2.5 Supervisor_Control_and_Status	55
15.2.6 Machine_Control_and_Status	55
15.2.7 Integration_support	57

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 X280 64-bit processor model

1.2 Licensing

This Model is released under the Open Source Apache 2.0

1.3 Extensions

1.3.1 Extensions Enabled by Default

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

misa bit 0: extension A (atomic instructions)

misa bit 1: extension B (bit manipulation extension)

misa bit 2: extension C (compressed instructions)

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

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

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

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

misa bit 18: extension S (Supervisor mode)

misa bit 20: extension U (User mode)

misa bit 21: extension V (vector 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 Disabling Extensions

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

misa bit 0: extension A (atomic instructions)

misa bit 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 Multicore Features

This is a multicore variant with 1 harts by default. The number of harts may be overridden with the “numHarts” parameter.

1.4.2 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 0x3fffffff. 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 64.

If parameter “mtvec_sext” is True, values written to “mtvec” are sign-extended from the most-significant writable bit. In this variant, “mtvec_sext” is False, indicating that “mtvec” is not sign-extended.

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

1.4.3 stvec CSR

Values written to “stvec” are masked using the value 0xffffffffffff. A different mask of writable bits may be specified using parameter “stvec_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 64.

If parameter “stvec_sext” is True, values written to “stvec” are sign-extended from the most-significant writable bit. In this variant, “stvec_sext” is False, indicating that “stvec” is not sign-extended.

1.4.4 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.

1.4.5 NMI

On an NMI, the model will restart at address 0x0; a different NMI address may be specified using parameter “nmi_address” or applied using optional input port “nmi_addr” if required. The

cause reported on an NMI is 0x0 by default; a different cause may be specified using parameter “`ecode_nmi`” or applied using optional input port “`nmi_cause`” if required.

If parameter “`rnmi_version`” is not “`none`”, resumable NMIs are supported, managed by additional CSRs “`mnscratch`”, “`mnepc`”, “`mncause`” and “`mnstatus`”, following the indicated version of the Resumable NMI extension proposal. In this variant, “`rnmi_version`” is “`none`”.

1.4.6 WFI

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

1.4.7 cycle CSR

The “`cycle`” CSR is implemented in this variant. Set parameter “`cycle_undefined`” to True to instead specify that “`cycle`” is unimplemented and reads of it should cause Illegal Instruction traps.

1.4.8 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.

1.4.9 instret CSR

The “`instret`” CSR is implemented in this variant. Set parameter “`instret_undefined`” to True to instead specify that “`instret`” is unimplemented and reads of it should cause Illegal Instruction traps.

1.4.10 hpmcounter CSRs

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

1.4.11 Virtual Memory

This variant supports address translation modes 0 (bare), 8 (Sv39) and 9 (Sv48). Use parameter “`Sv_modes`” to specify a bit mask of different implemented modes if required; for example, setting “`Sv_modes`” to $(1 < 0) + (1 < 8)$ indicates that mode 0 (bare) and mode 8 (Sv39) are implemented. These indices correspond to writable values in the `satp.MODE` CSR field.

A 0-bit ASID is implemented. Use parameter “`ASID_bits`” to specify a different implemented ASID size if required.

TLB behavior is controlled by parameter “ASIDCacheSize”. If this parameter is 0, then an unlimited number of TLB entries will be maintained concurrently. If this parameter is non-zero, then only TLB entries for up to “ASIDCacheSize” different ASIDs will be maintained concurrently initially; as new ASIDs are used, TLB entries for less-recently used ASIDs are deleted, which improves model performance in some cases. If the model detects that the TLB entry cache is too small (entry ejections are very frequent), it will increase the cache size automatically. In this variant, “ASIDCacheSize” is 8.

1.4.12 Unaligned Accesses

Unaligned memory accesses are not supported by this variant. Set parameter “unaligned” to “T” to enable such accesses.

Unaligned memory accesses are not supported for AMO instructions by this variant. Set parameter “unalignedAMO” to “T” to enable such accesses.

1.4.13 PMP

8 PMP entries are implemented by this variant. Use parameter “PMP_registers” to specify a different number of PMP entries; set the parameter to 0 to disable the PMP unit. The PMP grain size (G) is 0, meaning that PMP regions as small as 4 bytes are implemented. Use parameter “PMP_grain” to specify a different grain size if required. Unaligned PMP accesses are not decomposed into separate aligned accesses; use parameter “PMP_decompose” to modify this behavior if required.

1.4.14 LR/SC Granule

LR/SC instructions are implemented with a 64-byte reservation granule. A different granule size may be specified using parameter “lr_sc_grain”.

1.5 Compressed Extension

Standard compressed instructions are present in this variant.

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

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

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

1.6 Floating Point Features

16-bit floating point is implemented (IEEE 754 half precision implemented format). Use parameter “Zfh” to disable this if required.

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

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

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

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

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

In this variant, “mstatus_fs_mode” is set to “always_dirty”.

Parameter “Zfhmin” is 0 on this variant, meaning that all half-precision operations are supported. if “Zfhmin” is set to 1 then a minimal set of half-precision operations are supported.

For Zhinx, specify both “Zfh” and “Zfinx_version”.

For Zhinxmin, specify both “Zfhmin” and “Zfinx_version”.

1.7 Privileged Architecture

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

1.7.1 Legacy Version 1.10

1.10 version of May 7 2017.

1.7.2 Version 20190608

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

- mcountinhibit CSR defined;
- pages are never executable in Supervisor mode if page table entry U bit is 1;
- mstatus.TW is writable if any lower-level privilege mode is implemented (previously, it was just if Supervisor mode was implemented);

1.7.3 Version master

Unstable master version corresponding to evolving 1.12 specification, with these changes compared to version 20190608:

- mstatush, mseccfg, mseccfgh, menvcfg, menvcfgh, senvcfg, henvcfg, henvcfgh and mconfigptr CSRs defined;
- xret instructions clear mstatus.MPRV when leaving Machine mode if new mode is less privileged than M-mode;
- maximum number of PMP registers increased to 64;
- data endian is now configurable.

1.8 Unprivileged Architecture

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

1.8.1 Legacy Version 2.2

2.2 version of May 7 2017.

1.8.2 Version 20191213

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

- floating point fmin/fmax instruction behavior modified to comply with IEEE 754-201x.
- numerous other optional behaviors can be separately enabled using Z-prefixed parameters.

1.9 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.

1.9.1 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 0 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 0 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.

1.9.2 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.

1.9.3 Version 0.90

Stable 0.90 version of June 10 2019.

1.9.4 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.

1.9.5 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.

1.9.6 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.

1.9.7 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;
- instructions named sb* renamed to b*;
- instructions named pcnt* renamed to cpop*;

- instructions `subu.w`, `addiw`, `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);

1.9.8 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.

1.9.9 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.

1.9.10 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`.

1.10 Vector Extension

This variant implements the RISC-V base vector extension with version specified in the References section of this document. Note that parameter “`vector_version`” can be used to select the required version, including the unstable “`master`” version corresponding to the active specification. See section “Vector Extension Versions” for detailed information about differences between each supported version.

1.10.1 Vector Extension Parameters

Parameter `ELEN` is used to specify the maximum size of a single vector element in bits (32 or 64). By default, `ELEN` is set to 64 in this variant.

Parameter `VLEN` is used to specify the number of bits in a vector register (a power of two in the range 32 to 65536). By default, `VLEN` is set to 512 in this variant.

Parameter `SLEN` is used to specify the striping distance (a power of two in the range 32 to 65536). By default, `SLEN` is set to 512 in this variant.

Parameter `EEW_index` is used to specify the maximum supported `EEW` for index load/store instructions (a power of two in the range 8 to `ELEN`). By default, `EEW_index` is set to 64 in this variant.

Parameter `SEW_min` is used to specify the minimum supported `SEW` (a power of two in the range 8 to `ELEN`). By default, `SEW_min` is set to 8 in this variant.

Parameter `Zvlsseg` is used to specify whether the `Zvlsseg` extension is implemented. By default, `Zvlsseg` is set to 1 in this variant.

Parameter `Zvamo` is used to specify whether the `Zvamo` extension is implemented. By default, `Zvamo` is set to 1 in this variant.

Parameter `Zvediv` will be used to specify whether the `Zvediv` extension is implemented. This is not currently supported.

Parameter `Zvqmac` is used to specify whether the `Zvqmac` extension is implemented (from version 0.8-draft-20191117 only). By default, `Zvqmac` is set to 0 in this variant.

Parameter `Zve32x` is used to specify whether the `Zve32x` extension is implemented. By default, `Zve32x` is set to 0 in this variant.

Parameter `Zve32f` is used to specify whether the `Zve32f` extension is implemented. By default, `Zve32f` is set to 0 in this variant.

Parameter `Zve64x` is used to specify whether the `Zve64x` extension is implemented. By default, `Zve64x` is set to 0 in this variant.

Parameter `Zve64f` is used to specify whether the `Zve64f` extension is implemented. By default, `Zve64f` is set to 0 in this variant.

Parameter `Zve64d` is used to specify whether the `Zve64d` extension is implemented. By default, `Zve64d` is set to 0 in this variant.

Parameter `require_vstart0` is used to specify whether non-interruptible vector instructions require `vstart=0`. By default, `require_vstart0` is set to 0 in this variant.

Parameter `align_whole` is used to specify whether whole-register load and store instructions require alignment to the encoded `EEW`. By default, `align_whole` is set to 0 in this variant.

Parameter `vill_trap` is used to specify whether attempts to write illegal values to `vtype` cause an Illegal Instruction trap. By default, `vill_trap` is set to 0 in this variant.

Parameter `agnostic_ones` is used to specify whether agnostic fields are filled with all-ones (from Vector Extension version 0.9 only). By default, `agnostic_ones` is set to 0 in this variant, meaning mask tails, vector tail elements and vector masked-off elements all show undisturbed behavior.

1.10.2 Vector Extension Features

The model implements the base vector extension with a maximum ELEN of 64. Striping, masking and polymorphism are all fully supported. Zvlsseg and Zvamo extensions are fully supported. The Zvediv extension specification is subject to change and therefore not yet supported.

Single precision and double precision floating point types are supported if those types are also supported in the base architecture (i.e. the corresponding D and F features must be present and enabled). Vector floating point operations may only be executed if the base floating point unit is also enabled (i.e. mstatus.FS must be non-zero). Attempting to execute vector floating point instructions when mstatus.FS is 0 will cause an Illegal Instruction exception.

The model assumes that all vector memory operations must be aligned to the memory element size. Unaligned accesses will cause a Load/Store Address Alignment exception.

By default, the processor starts with vector extension disabled (mstatus.VS=0). Use parameter “mstatus_VS” to force mstatus.VS to a non-zero value for the vector extension to be enabled from the start.

1.10.3 Vector Extension Versions

The Vector 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.

1.10.4 Version 0.7.1-draft-20190605

Stable 0.7.1 version of June 10 2019.

1.10.5 Version 0.7.1-draft-20190605+

Version 0.7.1, with some 0.8 and custom features. Not intended for general use.

1.10.6 Version 0.8-draft-20190906

Stable 0.8 draft of September 6 2019, with these changes compared to version 0.7.1-draft-20190605:

- tail vector and scalar elements preserved, not zeroed;
- vext.s.v, vmford.vv and vmford.vf instructions removed;
- encodings for vfmv.f.s, vfmv.s.f, vmv.s.x, vpopc.m, vfirst.m, vmsbf.m, vmsif.m, vmsof.m, viota.m and vid.v instructions changed;
- overlap constraints for slideup and slidedown instructions relaxed to allow overlap of destination and mask when SEW=1;
- 64-bit vector AMO operations replaced with SEW-width vector AMO operations;

- vsetvl and vsetvli instructions when $rs1 = x0$ preserve the current vl instead of selecting the maximum possible vl;
- instruction vfnctv.rod.f.f.w added (to allow narrowing floating point conversions with jamming semantics);
- instructions that transfer values between vector registers and general purpose registers (vmv.s.x and vmv.x.s) sign-extend the source if required (previously, it was zero-extended).

1.10.7 Version 0.8-draft-20191004

Stable 0.8 draft of October 4 2019, with these changes compared to version 0.8-draft-20190906:

- vwmaccsu and vwmaccus instruction encodings exchanged;
- vwsmaccsu and vwsmaccus instruction encodings exchanged.

1.10.8 Version 0.8-draft-20191117

Stable 0.8 draft of November 17 2019, with these changes compared to version 0.8-draft-20191004:

- indexed load/store instructions zero-extend offsets (previously, they were sign-extended);
- vslide1up/vslide1down instructions sign-extend XLEN values to SEW length (previously, they were zero-extended);
- vadc/vsbc instruction encodings require $vm=0$ (previously, they required $vm=1$);
- vmadc/vmsbc instruction encodings allow both $vm=0$, implying carry input is used, and $vm=1$, implying carry input is zero (previously, only $vm=1$ was permitted, implying carry input is used);
- vaaddu.vv, vaaddu.vx, vasubu.vv and vasubu.vx instructions added;
- vaadd.vv and vaadd.vx, instruction encodings changed;
- vaadd.vi instruction removed;
- all widening saturating scaled multiply-add instructions removed;
- vqmaccu.vv, vqmaccu.vx, vqmacc.vv, vqmacc.vx, vqmacc.vx, vqmaccsu.vx and vqmaccus.vx instructions added;
- CSR vlenb added (vector register length in bytes);
- load/store whole register instructions added;
- whole register move instructions added.

1.10.9 Version 0.8-draft-20191118

Stable 0.8 draft of November 18 2019, with these changes compared to version 0.8-draft-20191117:

- vsetvl/vsetvli with $rd!=zero$ and $rs1=zero$ sets vl to the maximum vector length.

1.10.10 Version 0.8

Stable 0.8 official release (commit 9a65519), with these changes compared to version 0.8-draft-20191118:

- vector context status in mstatus register is now implemented;
- whole register load and store operations have been restricted to a single register only;
- whole register move operations have been restricted to aligned groups of 1, 2, 4 or 8 registers only.

1.10.11 Version 0.9

Stable 0.9 official release (commit cb7d225), with these significant changes compared to version 0.8:

- mstatus.VS and sstatus.VS fields moved to bits 10:9;
- new CSR vcsr added and fields VXSAT and VXRM relocated there from CSR fcsr;
- vslide1up.vf, vslide1down.vf, vfcvt.rtz.xu.f.v, vfcvt.rtz.x.f.v, vfwcvt.rtz.xu.f.v, vfwcvt.rtz.x.f.v, vfnvcvt.rtz.xu.f.v, vfnvcvt.rtz.x.f.v, vzext.vf2, vsex.vf2, vzext.vf4, vsex.vf4, vzext.vf8 and vsex.vf8 instructions added;
- fractional LMUL support added, controlled by an extended vtype.vlmul CSR field;
- vector tail agnostic and vector mask agnostic fields added to the vtype CSR;
- all vector load/store instructions replaced with new instructions that explicitly encode EEW of data or index;
- whole register load and store operation encodings changed;
- VFUNARY0 and VFUNARY1 encodings changed;
- MLEN is always 1;
- for implementations with SLEN != VLEN, striping is applied horizontally rather than the previous vertical striping;
- vmsbf.m, vmsif.m and vmsof.m no longer allow overlap of destination with source or mask registers.

1.10.12 Version 1.0-draft-20210130

Stable 1.0-draft-20210130 official release (commit 8e768b0), with these changes compared to version 0.9:

- SLEN=VLEN register layout is mandatory;
- ELEN>VLEN is now supported for LMUL>1;
- whole register moves and load/stores now have element size hints;
- whole register load and store operations now permit use of aligned groups of 1, 2, 4 or 8 registers.

- overlap constraints for different source/destination EEW changed;
- instructions `vfqrt7.v`, `vfrec7.v` and `vrgatherei16.vv` added;
- CSR `vtype` format changed to make `vlmul` bits contiguous.
- `vsetvli x0, x0, imm` instruction is reserved if it would cause `vl` to change;
- ordered/unordered indexed vector memory instructions added;
- instructions `vle1.v`, `vse1.v` and `vsetivli` added.

1.10.13 Version 1.0-rc1-20210608

Stable 1.0-rc1-20210608 official release (commit 795a4dd), with these changes compared to version 1.0-draft-20210130:

- instructions `vle1.v/vse1.v` renamed `vlm.v/vsm.v`;
- instructions `vfredsum.vs/vfwredsum.vs` renamed `vfredusum.vs/vfwredusum.vs`;
- whole-register load/store instructions now use the EEW encoded in the instruction to determine element size (previously, this was a hint and element size 8 was used).

1.10.14 Version master

Unstable master version as of 20 September 2021 (commit 3570f99) with these changes compared to version 1.0-rc1-20210608:

- instruction `vpopc.m` renamed `vcpop.m`;
- instruction `vmandnot.mm` renamed `vmandn.mm`;
- instruction `vmornot.mm` renamed `vmorn.mm`.

1.11 Other Extensions

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

1.11.1 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.

1.11.2 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.

1.11.3 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.

1.11.4 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.

1.11.5 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).

1.11.6 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”.

1.11.7 Svnapot

Parameter “Svnapot_page_mask” is 0x0 on this variant, meaning that NAPOT Translation Contiguity is not implemented. if “Svnapot_page_mask” is non-zero then NAPOT Translation Contiguity is enabled for page sizes indicated by that mask value when page table entry bit 63 is set.

If Svnapot is present, “Svnapot_page_mask” is a mask of page sizes for which contiguous pages can be created. For example, a value of 0x10000 implies that 64KiB contiguous pages are supported.

1.11.8 Svpbmt

Parameter “Svpbmt” is 0 on this variant, meaning that page-based memory types are not implemented. if “Svpbmt” is set to 1 then page-based memory types are indicated by page table entry bits 62:61.

Note that except for their effect on Page Faults, the encoded memory types do not alter the behavior of this model, which always implements strongly-ordered non-cacheable semantics.

1.11.9 Svinval

Parameter “Svinval” is 0 on this variant, meaning that fine-grained address-translation cache invalidation instructions are not implemented. If “Svinval” is set to 1 then fine-grained address-translation cache invalidation instructions `sinval.vma`, `sfence.w.inval` and `sfence.inval.ir` are implemented.

1.12 Load-Reserved/Store-Conditional Locking

By default, LR/SC locking is implemented automatically by the model and simulator, with a reservation granule defined by the “`lr_sc_grain`” parameter. It is also possible to implement locking externally to the model in a platform component, using the “`LR_address`”, “`SC_address`” and “`SC_valid`” net ports, as described below.

The “`LR_address`” output net port is written by the model with the address used by a load-reserved instruction as it executes. This port should be connected as an input to the external lock management component, which should record the address, and also that an LR/SC transaction is active.

The “`SC_address`” output net port is written by the model with the address used by a store-conditional instruction as it executes. This should be connected as an input to the external lock management component, which should compare the address with the previously-recorded load-reserved address, and determine from this (and other implementation-specific constraints) whether the store should succeed. It should then immediately write the Boolean success/fail code to the “`SC_valid`” input net port of the model. Finally, it should update state to indicate that an LR/SC transaction is no longer active.

It is also possible to write zero to the “`SC_valid`” input net port at any time outside the context of a store-conditional instruction, which will mark any active LR/SC transaction as invalid.

Irrespective of whether LR/SC locking is implemented internally or externally, taking any exception or interrupt or executing exception-return instructions (e.g. `MRET`) will always mark any active LR/SC transaction as invalid.

1.13 Active Atomic Operation Indication

The “`AMO_active`” output net port is written by the model with a code indicating any current atomic memory operation while the instruction is active. The written codes are:

0: no atomic instruction active

1: `AMOMIN` active

2: `AMOMAX` active

- 3: AMOMINU active
- 4: AMOMAXU active
- 5: AMOADD active
- 6: AMOXOR active
- 7: AMOOR active
- 8: AMOAND active
- 9: AMOSWAP active
- 10: LR active
- 11: SC active

1.14 Interrupts

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

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

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

Parameter “external_int_id” can be used to enable extra interrupt ID input ports on each hart. If the parameter is True then when an external interrupt is applied the value on the ID port is sampled and used to fill the Exception Code field in the “mcause” CSR (or the equivalent CSR for other execution levels). For Machine mode, the extra interrupt ID port is called “MExternalInterruptID”.

The “deferint” port is an active-high artifact input that, when written to 1, prevents any pending-and-enabled interrupt being taken (normally, such an interrupt would be taken on the next instruction after it becomes pending-and-enabled). The purpose of this signal is to enable alignment with hardware models in step-and-compare usage.

1.15 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.

1.15.1 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.

1.15.2 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.

1.15.3 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.

1.15.4 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.

1.15.5 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.

1.15.6 Debug Ports

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

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

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

1.16 Debug Mask

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

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

Value 0x004: enable debugging of interrupt state.

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

1.17 Integration Support

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

1.17.1 CSR Register External Implementation

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

1.17.2 LR/SC Active Address

Artifact register “LRSCAddress” shows the active LR/SC lock address. The register holds all-ones if there is no LR/SC operation active or if LR/SC locking is implemented externally as described above.

1.17.3 Page Table Walk Introspection

Artifact register “PTWStage” shows the active page table translation stage (0 if no stage active, 1 if HS-stage active, 2 if VS-stage active and 3 if G-stage active). This register is visibly non-zero only in a memory access callback triggered by a page table walk event.

Artifact register “PTWInputAddr” shows the input address of active page table translation. This register is visibly non-zero only in a memory access callback triggered by a page table walk event.

Artifact register “PTWLevel” shows the active level of page table translation (corresponding to index variable “i” in the algorithm described by Virtual Address Translation Process in the RISC-V Privileged Architecture specification). This register is visibly non-zero only in a memory access callback triggered by a page table walk event.

1.17.4 Artifact Register “fflags_i”

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

1.18 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.

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

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

RISC-V “B” Bit Manipulation Extension (Bit Manipulation Architecture Version v0.94-20210120)

RISC-V “V” Vector Extension (Vector Architecture Master Branch as of commit 3570f99 (this is subject to change))

— Mongoose Specs sheet 00.09 - Imperas

— SiFive X280 Core Complex Manual - 21G2.01.00

— Initial configuration definition - work in progress

Chapter 2

SiFive-Specific Extensions

SiFive processors can add various custom extensions to the basic RISC-V architecture. This model implements the following:

2.1 SiFive-Specific CSRs

This section describes SiFive-specific CSRs implemented by this variant. Refer to SiFive reference documentation for more information.

2.1.1 bpm (csr num 0x7c0)

Reading and writing the SiFive custom M-Mode Branch Prediction Mode CSR is supported. Since this register controls only micro-architectural behavior, which is not modeled, the setting of this register has no effect.

2.1.2 featureDisable (csr num 0x7c1)

Reading and writing the SiFive custom M-Mode Feature Disable CSR is supported. Since this register controls only micro-architectural behavior, which is not modeled, the setting of this register has no effect, and all fields are hardwired to 0.

2.2 SiFive-Specific Instructions

This section describes SiFive-specific instructions implemented by this variant. Refer to SiFive reference documentation for more information.

2.2.1 CFLUSH.D.L1

31	20	19	15	14	12	11	7	6	0
111111000000		Rs1		000		00000		1110011 (System)	

Instruction to flush the DCACHE L1 line for the address in Rs1. This instruction will generate exceptions for attempts to flush addresses that are not writable, but since caches are not modeled will otherwise have no effect.

2.2.2 CDISCARD.D.L1

31	20	19	15	14	12	11	7	6	0
111111000010		Rs1		000		00000		1110011 (System)	

Instruction to discard the DCACHE L1 line for the address in Rs1. This instruction will generate exceptions for attempts to flush addresses that are not writable, but since caches are not modeled will otherwise have no effect.

2.2.3 CEASE

31	20	19	15	14	12	11	7	6	0
001100000101		00000		000		00000		1110011 (System)	

Instruction to cease execution until the hart is reset.

Chapter 3

SiFive-Specific Extensions

SiFive processors can add various custom extensions to the basic RISC-V architecture. This model implements the following:

3.1 SiFive-Specific CSRs

This section describes SiFive-specific CSRs implemented by this variant. Refer to SiFive reference documentation for more information.

3.1.1 bpm (csr num 0x7c0)

Reading and writing the SiFive custom M-Mode Branch Prediction Mode CSR is supported. Since this register controls only micro-architectural behavior, which is not modeled, the setting of this register has no effect.

3.1.2 featureDisable (csr num 0x7c1)

Reading and writing the SiFive custom M-Mode Feature Disable CSR is supported. Since this register controls only micro-architectural behavior, which is not modeled, the setting of this register has no effect, and all fields are hardwired to 0.

3.2 SiFive-Specific Instructions

This section describes SiFive-specific instructions implemented by this variant. Refer to SiFive reference documentation for more information.

3.2.1 CFLUSH.D.L1

31	20	19	15	14	12	11	7	6	0
111111000000		Rs1		000		00000		1110011 (System)	

Instruction to flush the DCACHE L1 line for the address in Rs1. This instruction will generate exceptions for attempts to flush addresses that are not writable, but since caches are not modeled will otherwise have no effect.

3.2.2 CDISCARD.D.L1

31	20	19	15	14	12	11	7	6	0
111111000010		Rs1		000		00000		1110011 (System)	

Instruction to discard the DCACHE L1 line for the address in Rs1. This instruction will generate exceptions for attempts to flush addresses that are not writable, but since caches are not modeled will otherwise have no effect.

3.2.3 CEASE

31	20	19	15	14	12	11	7	6	0
001100000101		00000		000		00000		1110011 (System)	

Instruction to cease execution until the hart is reset.

Chapter 4

Configuration

4.1 Location

This model's VLVN is sifive.ovpworld.org/processor/riscv/1.0.

The model source is usually at:

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

The model binary is usually at:

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

4.2 GDB Path

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

4.3 Semi-Host Library

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

4.4 Processor Endian-ness

This is a LITTLE endian model.

4.5 QuantumLeap Support

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

4.6 Processor ELF code

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

Chapter 5

All Variants in this model

This model has these variants

Variant	Description
E20	
E21	
E24	
E31	
E34	
E51	
E76	
S21	
S51	
S54	
S76	
U54	
U74	
X280	(described in this document)
P550	

Table 5.1: All Variants in this model

Chapter 6

Bus Master Ports

This model has these bus master ports.

Name	min	max	Connect?	Description
INSTRUCTION	32	64	mandatory	Instruction bus
DATA	32	64	optional	Data bus

Table 6.1: Bus Master Ports

Chapter 7

Bus Slave Ports

This model has no bus slave ports.

Chapter 8

Net Ports

This model has these net ports.

Name	Type	Connect?	Description
hart0_reset	input	optional	Reset
hart0_reset_addr	input	optional	externally-applied reset address
hart0_nmi	input	optional	NMI
hart0_nmi_cause	input	optional	externally-applied NMI cause
hart0_nmi_addr	input	optional	externally-applied NMI address
hart0_SSWInterrupt	input	optional	Supervisor software interrupt
hart0_MSWInterrupt	input	optional	Machine software interrupt
hart0_STimerInterrupt	input	optional	Supervisor timer interrupt
hart0_MTimerInterrupt	input	optional	Machine timer interrupt
hart0_SExternalInterrupt	input	optional	Supervisor external interrupt
hart0_MExternalInterrupt	input	optional	Machine external interrupt
hart0_LocalInterrupt0	input	optional	Local interrupt 0
hart0_LocalInterrupt1	input	optional	Local interrupt 1
hart0_LocalInterrupt2	input	optional	Local interrupt 2
hart0_LocalInterrupt3	input	optional	Local interrupt 3
hart0_LocalInterrupt4	input	optional	Local interrupt 4
hart0_LocalInterrupt5	input	optional	Local interrupt 5
hart0_LocalInterrupt6	input	optional	Local interrupt 6
hart0_LocalInterrupt7	input	optional	Local interrupt 7
hart0_LocalInterrupt8	input	optional	Local interrupt 8
hart0_LocalInterrupt9	input	optional	Local interrupt 9
hart0_LocalInterrupt10	input	optional	Local interrupt 10
hart0_LocalInterrupt11	input	optional	Local interrupt 11
hart0_LocalInterrupt12	input	optional	Local interrupt 12
hart0_LocalInterrupt13	input	optional	Local interrupt 13
hart0_LocalInterrupt14	input	optional	Local interrupt 14
hart0_LocalInterrupt15	input	optional	Local interrupt 15
hart0_LocalInterrupt16	input	optional	Local interrupt 16
hart0_LocalInterrupt17	input	optional	Local interrupt 17
hart0_LocalInterrupt18	input	optional	Local interrupt 18
hart0_LocalInterrupt19	input	optional	Local interrupt 19

hart0_LocalInterrupt20	input	optional	Local interrupt 20
hart0_LocalInterrupt21	input	optional	Local interrupt 21
hart0_LocalInterrupt22	input	optional	Local interrupt 22
hart0_LocalInterrupt23	input	optional	Local interrupt 23
hart0_LocalInterrupt24	input	optional	Local interrupt 24
hart0_LocalInterrupt25	input	optional	Local interrupt 25
hart0_LocalInterrupt26	input	optional	Local interrupt 26
hart0_LocalInterrupt27	input	optional	Local interrupt 27
hart0_LocalInterrupt28	input	optional	Local interrupt 28
hart0_LocalInterrupt29	input	optional	Local interrupt 29
hart0_LocalInterrupt30	input	optional	Local interrupt 30
hart0_LocalInterrupt31	input	optional	Local interrupt 31
hart0_LocalInterrupt32	input	optional	Local interrupt 32
hart0_LocalInterrupt33	input	optional	Local interrupt 33
hart0_LocalInterrupt34	input	optional	Local interrupt 34
hart0_LocalInterrupt35	input	optional	Local interrupt 35
hart0_LocalInterrupt36	input	optional	Local interrupt 36
hart0_LocalInterrupt37	input	optional	Local interrupt 37
hart0_LocalInterrupt38	input	optional	Local interrupt 38
hart0_LocalInterrupt39	input	optional	Local interrupt 39
hart0_LocalInterrupt40	input	optional	Local interrupt 40
hart0_LocalInterrupt41	input	optional	Local interrupt 41
hart0_LocalInterrupt42	input	optional	Local interrupt 42
hart0_LocalInterrupt43	input	optional	Local interrupt 43
hart0_LocalInterrupt44	input	optional	Local interrupt 44
hart0_LocalInterrupt45	input	optional	Local interrupt 45
hart0_LocalInterrupt46	input	optional	Local interrupt 46
hart0_LocalInterrupt47	input	optional	Local interrupt 47
hart0_irq_ack_o	output	optional	interrupt acknowledge (pulse)
hart0_irq_id_o	output	optional	acknowledged interrupt id (valid during irq_ack_o pulse)
hart0_sec_lvl_o	output	optional	current privilege level
hart0_LR_address	output	optional	Port written with effective address for LR instruction
hart0_SC_address	output	optional	Port written with effective address for SC instruction
hart0_SC_valid	input	optional	SC_address valid input signal
hart0_AMO_active	output	optional	Port written with code indicating active AMO
hart0_deferint	input	optional	Artifact signal causing interrupts to be held off when high

Table 8.1: Net Ports

Chapter 9

FIFO Ports

This model has no FIFO ports.

Chapter 10

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 or master)
numHarts	Uns32	Specify the number of hart contexts in a multiprocessor
endian	Endian	Model endian
enable_expanded	Boolean	Specify that 48-bit and 64-bit expanded instructions are supported
endianFixed	Boolean	Specify that data endianness is fixed (mstatus.{MBE,SBE,UBE} fields are read-only)
misa_MXL	Uns32	Override default value of misa.MXL
misa_Extensions	Uns32	Override default value of misa.Extensions
add_Extensions	String	Add extensions specified by letters to misa.Extensions (for example, specify “VD” to add V and D features)
sub_Extensions	String	Remove extensions specified by letters from misa.Extensions (for example, specify “VD” to remove V and D features)
misa_Extensions_mask	Uns32	Override mask of writable bits in misa.Extensions
add_Extensions_mask	String	Add extensions specified by letters to mask of writable bits in misa.Extensions (for example, specify “VD” to add V and D features)
sub_Extensions_mask	String	Remove extensions specified by letters from mask of writable bits in misa.Extensions (for example, specify “VD” to remove V and D features)
add_implicit_Extensions	String	Add extensions specified by letters to implicitly-present extensions not visible in misa.Extensions
sub_implicit_Extensions	String	Remove extensions specified by letters from implicitly-present extensions not visible in misa.Extensions
Zicsr	Boolean	Specify that Zicsr is implemented
Zifencei	Boolean	Specify that Zifencei is implemented
Zicbom	Boolean	Specify that Zicbom is implemented
Zicbop	Boolean	Specify that Zicbop is implemented
Zicboz	Boolean	Specify that Zicboz is implemented
Zmmul	Boolean	Specify that Zmmul is implemented
Vector		
vector_version	Enumeration	Specify required Vector Architecture version (0.7.1-draft-20190605, 0.7.1-draft-20190605+, 0.8-draft-20190906, 0.8-draft-20191004, 0.8-draft-20191117, 0.8-draft-20191118, 0.8, 0.9, 1.0-draft-20210130, 1.0-rc1-20210608 or master)
fp16_version	Enumeration	Specify required 16-bit floating point format (none, IEEE754 or BFLOAT16)
require_vstart0	Boolean	Whether CSR vstart must be 0 for non-interruptible vector instructions

align_whole	Boolean	Whether whole-register load addresses must be aligned using the encoded EEW
vill_trap	Boolean	Whether illegal vtype values cause trap instead of setting vtype.vill
mstatus_VS	Uns32	Override default value of mstatus.VS (initial state of vector unit)
ELEN	Uns32	Override ELEN (vector extension)
SLEN	Uns32	Override SLEN (vector extension before version 1.0 only)
VLEN	Uns32	Override VLEN (vector extension)
EEW_index	Uns32	Override maximum supported index EEW (vector extension, use ELEN if zero)
SEW_min	Uns32	Override minimum supported SEW (vector extension)
agnostic_ones	Boolean	Specify that vector agnostic elements are set to 1 (vector extension)
Zvlsseg	Boolean	Specify that Zvlsseg is implemented (vector extension)
Zvamo	Boolean	Specify that Zvamo is implemented (vector extension)
Zvediv	Boolean	Specify that Zvediv is implemented (vector extension)
Zvqmac	Boolean	Specify that Zvqmac is implemented (vector extension)
Zve32x	Boolean	Specify that Zve32x is implemented (vector extension)
Zve32f	Boolean	Specify that Zve32f is implemented (vector extension)
Zve64x	Boolean	Specify that Zve64x is implemented (vector extension)
Zve64f	Boolean	Specify that Zve64f is implemented (vector extension)
Zve64d	Boolean	Specify that Zve64d is implemented (vector extension)
Bit Manipulation		
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 (bit manipulation extension)
Zbb	Boolean	Specify that Zbb is implemented (bit manipulation extension)
Zbc	Boolean	Specify that Zbc is implemented (bit manipulation extension)
Zbe	Boolean	Specify that Zbe is implemented (bit manipulation extension; ignored if version 1.0.0)
Zbf	Boolean	Specify that Zbf is implemented (bit manipulation extension; ignored if version 1.0.0)
Zbm	Boolean	Specify that Zbm is implemented (bit manipulation extension; ignored if version 1.0.0)
Zbp	Boolean	Specify that Zbp is implemented (bit manipulation extension; ignored if version 1.0.0)
Zbr	Boolean	Specify that Zbr is implemented (bit manipulation extension; ignored if version 1.0.0)
Zbs	Boolean	Specify that Zbs is implemented (bit manipulation extension)
Zbt	Boolean	Specify that Zbt is implemented (bit manipulation extension; ignored if version 1.0.0)
Interrupts Exceptions		
rnmi_version	Enumeration	Specify required RNMI Architecture version (none or 0.2.1)
mtvec_is_ro	Boolean	Specify whether mtvec CSR is read-only
tvec_align	Uns32	Specify hardware-enforced alignment of mtvec/stvec/utvec when Vectored interrupt mode enabled
ecode_mask	Uns64	Specify hardware-enforced mask of writable bits in xcause.ExceptionCode
ecode_nmi	Uns64	Specify xcause.ExceptionCode for NMI
tval_zero	Boolean	Specify whether mtval/stval/utval are hard wired to zero
tval_zero_ebreak	Boolean	Specify whether mtval/stval/utval are set to zero by an ebreak
tval_ii_code	Boolean	Specify whether mtval/stval contain faulting instruction bits on illegal instruction exception
trap_preserves_lr	Boolean	Whether a trap preserves active LR/SC state
xret_preserves_lr	Boolean	Whether an xret instruction preserves active LR/SC state
reset_address	Uns64	Override reset vector address
nmi_address	Uns64	Override NMI vector address
CLINT_address	Uns64	Specify base address of internal CLINT model (or 0 for no CLINT)
local_int_num	Uns32	Specify number of supplemental local interrupts

unimp_int_mask	Uns64	Specify mask of unimplemented interrupts (e.g. 1<<9 indicates Supervisor external interrupt unimplemented)
force_mideleg	Uns64	Specify mask of interrupts always delegated to lower-priority execution level from Machine execution level
force_sideleg	Uns64	Specify mask of interrupts always delegated to User execution level from Supervisor execution level
no_ideleg	Uns64	Specify mask of interrupts that cannot be delegated to lower-priority execution levels
no_e deleg	Uns64	Specify mask of exceptions that cannot be delegated to lower-priority execution levels
external_int_id	Boolean	Whether to add nets allowing External Interrupt ID codes to be forced
Floating Point		
mstatus_fs_mode	Enumeration	Specify conditions causing update of mstatus.FS to dirty (write_1, write_any or always_dirty)
d_requires_f	Boolean	If D and F extensions are separately enabled in the misa CSR, whether D is enabled only if F is enabled
enable_fflags_i	Boolean	Whether fflags.i artifact register present (shows per-instruction floating point flags)
mstatus_FS	Uns32	Override default value of mstatus.FS (initial state of floating point unit)
Zfh	Boolean	Specify that Zfh is implemented (IEEE half-precision floating point is supported)
Zfhmin	Boolean	Specify that Zfhmin is implemented (restricted IEEE half-precision floating point is supported)
Zfmx_version	Enumeration	Specify version of Zfmx implemented (use integer register file for floating point instructions) (none, 0.4 or 0.41)
Debug		
debug_mode	Enumeration	Specify how Debug mode is implemented (none, vector, interrupt or halt)
Simulation Artifact		
use_hw_reg_names	Boolean	Specify whether to use hardware register names x0-x31 and f0-f31 instead of ABI register names
ABI_d	Boolean	Specify whether D registers are used for parameters (ABI SemiHosting)
verbose	Boolean	Specify verbose output messages
traceVolatile	Boolean	Specify whether volatile registers (e.g. minstret) should be shown in change trace
enable_CSR_bus	Boolean	Add artifact CSR bus port, allowing CSR registers to be externally implemented
CSR_remap	String	Comma-separated list of CSR number mappings, each of the form <csr-Name>=<number>
ASID_cache_size	Uns32	Specifies the number of different ASIDs for which TLB entries are cached; a value of 0 implies no limit
Memory		
updatePTEA	Boolean	Specify whether hardware update of PTE A bit is supported
updatePTED	Boolean	Specify whether hardware update of PTE D bit is supported
unaligned	Boolean	Specify whether the processor supports unaligned memory accesses
unalignedAMO	Boolean	Specify whether the processor supports unaligned memory accesses for AMO instructions
ASID_bits	Uns32	Specify the number of implemented ASID bits
lr_sc_grain	Uns32	Specify byte granularity of ll/sc lock region (constrained to a power of two)
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

Sv_modes	Uns32	Specify bit mask of implemented address translation modes (e.g. (1<<0)+(1<<8) indicates “bare” and “Sv39” modes may be selected in satp.MODE)
Svnapot_page_mask	Uns64	Specify mask of implemented Svnapot intermediate page sizes (e.g. 1<<16 means 64KiB contiguous regions are supported)
Svpbmt	Boolean	Specify that Svpbmt is implemented (page-based memory types)
Svinval	Boolean	Specify that Svinval is implemented (fine-grained address translation cache invalidation)
Instruction_CSR_Behavior		
wfi_is_nop	Boolean	Specify whether WFI should be treated as a NOP (if not, halt while waiting for interrupts)
counteren_mask	Uns32	Specify hardware-enforced mask of writable bits in mcounteren/scounteren registers
noinhibit_mask	Uns32	Specify hardware-enforced mask of always-zero bits in mcountinhibit register
cycle_undefined	Boolean	Specify that the cycle CSR is undefined
time_undefined	Boolean	Specify that the time CSR is undefined
instret_undefined	Boolean	Specify that the instret CSR is undefined
hpmcounter_undefined	Boolean	Specify that the hpmcounter CSRs are undefined
CSR_Masks		
mtvec_mask	Uns64	Specify hardware-enforced mask of writable bits in mtvec register
stvec_mask	Uns64	Specify hardware-enforced mask of writable bits in stvec register
mip_mask	Uns64	Specify hardware-enforced mask of writable bits in mip register
sip_mask	Uns64	Specify hardware-enforced mask of writable bits in sip register
mtvec_sext	Boolean	Specify whether mtvec is sign-extended from most-significant bit
stvec_sext	Boolean	Specify whether stvec is sign-extended from most-significant bit
MXL_writable	Boolean	Specify that misa.MXL is writable (feature under development)
SXL_writable	Boolean	Specify that mstatus.SXL is writable (feature under development)
UXL_writable	Boolean	Specify that mstatus.UXL is writable (feature under development)
Trigger		
trigger_num	Uns32	Specify the number of implemented hardware triggers
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
Compressed		
Zcea_version	Enumeration	Specify version of Zcea implemented (code-size reduction extension) (none or 0.50.1)
Zceb_version	Enumeration	Specify version of Zceb implemented (code-size reduction extension) (none or 0.50.1)
Zcee_version	Enumeration	Specify version of Zcee implemented (code-size reduction extension) (none or 1.0.0-rc)
Fast Interrupt		
CLICLEVELS	Uns32	Specify number of interrupt levels implemented by CLIC, or 0 if CLIC absent

Table 10.1: Parameters that can be set in: SMP

10.1 Extension Parameters

Name	Type	Description
FeatureDisable_Present	Boolean	Specify whether Feature Disable CSR present

BranchPredictionMode_Present	Boolean	Specify whether Branch Prediction Mode CSR present
PowerDial_Present	Boolean	Specify whether PowerDial CSR present
CFLUSH_Present	Boolean	Specify whether CFLUSH.D.L1 and CDISCARD.D.L1 instructions present
CEASE_Present	Boolean	Specify whether CEASE instruction present

Table 10.2: Parameters for sifiveExtensions

10.2 Parameters with enumerated types

10.2.1 Parameter user_version

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

Table 10.3: Values for Parameter user_version

10.2.2 Parameter priv_version

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

Table 10.4: Values for Parameter priv_version

10.2.3 Parameter vector_version

Set to this value	Description
0.7.1-draft-20190605	Vector Architecture Version 0.7.1-draft-20190605
0.7.1-draft-20190605+	Vector Architecture Version 0.7.1-draft-20190605 with custom features (not for general use)
0.8-draft-20190906	Vector Architecture Version 0.8-draft-20190906
0.8-draft-20191004	Vector Architecture Version 0.8-draft-20191004
0.8-draft-20191117	Vector Architecture Version 0.8-draft-20191117
0.8-draft-20191118	Vector Architecture Version 0.8-draft-20191118
0.8	Vector Architecture Version 0.8
0.9	Vector Architecture Version 0.9
1.0-draft-20210130	Vector Architecture Version 1.0-draft-20210130
1.0-rc1-20210608	Vector Architecture Version 1.0-rc1-20210608
master	Vector Architecture Master Branch as of commit 3570f99 (this is subject to change)

Table 10.5: Values for Parameter vector_version

10.2.4 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 10.6: Values for Parameter `bitmanip_version`

10.2.5 Parameter `rnmi_version`

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

Table 10.7: Values for Parameter `rnmi_version`

10.2.6 Parameter `fp16_version`

Set to this value	Description
none	No 16-bit floating point implemented
IEEE754	IEEE 754 half precision implemented
BFLOAT16	BFLOAT16 implemented

Table 10.8: Values for Parameter `fp16_version`

10.2.7 Parameter `mstatus_fs_mode`

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

Table 10.9: Values for Parameter `mstatus_fs_mode`

10.2.8 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 10.10: Values for Parameter `debug_mode`

10.2.9 Parameter `Zfinx_version`

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

Table 10.11: Values for Parameter Zfinx_version

10.2.10 Parameter Zcea_version

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

Table 10.12: Values for Parameter Zcea_version

10.2.11 Parameter Zceb_version

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

Table 10.13: Values for Parameter Zceb_version

10.2.12 Parameter Zcee_version

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

Table 10.14: Values for Parameter Zcee_version

10.3 Parameter values

These are the current parameter values.

Name	Value
Fundamental	
variant	X280
user_version	2.3
priv_version	1.11
numHarts	1
endian	none
enable_expanded	F
endianFixed	F
misa_MXL	2
misa_Extensions	0x34112f
add_Extensions	
sub_Extensions	
misa_Extensions_mask	0
add_Extensions_mask	
sub_Extensions_mask	
add_implicit_Extensions	

sub_implicit_Extensions	
Zicsr	T
Zifencei	T
Zicbom	F
Zicbop	F
Zicboz	F
Zmmul	F
Vector	
vector_version	master
fp16_version	IEEE754
require_vstart0	F
align_whole	F
vill_trap	F
mstatus_VS	0
ELEN	64
SLEN	0x200
VLEN	0x200
EEW_index	0
SEW_min	8
agnostic_ones	F
Zvlsseg	T
Zvamo	T
Zvediv	F
Zvqmac	F
Zve32x	F
Zve32f	F
Zve64x	F
Zve64f	F
Zve64d	F
Bit_Manipulation	
bitmanip_version	0.94
Zba	T
Zbb	T
Zbc	F
Zbe	F
Zbf	F
Zbm	F
Zbp	F
Zbr	F
Zbs	F
Zbt	F
Interrupts_Exceptions	
rnmi_version	none
mtvec_is_ro	F
tvec_align	64

ecode_mask	31
ecode_nmi	0
tval_zero	F
tval_zero_ebreak	F
tval_ii_code	T
trap_preserves_lr	F
xret_preserves_lr	F
reset_address	0
nmi_address	0
CLINT_address	0
local_int_num	48
unimp_int_mask	0
force_mideleg	0
force_sideleg	0
no_ideleg	0
no_e deleg	0
external_int_id	F
Floating Point	
mstatus_fs_mode	always_dirty
d_requires_f	T
enable_fflags_i	F
mstatus_FS	0
Zfh	T
Zfhmin	F
Zfinx_version	none
Debug	
debug_mode	none
Simulation Artifact	
use_hw_reg_names	F
ABI_d	T
verbose	F
traceVolatile	F
enable_CSR_bus	F
CSR_remap	
ASID_cache_size	8
Memory	
updatePTEA	F
updatePTED	F
unaligned	F
unalignedAMO	F
ASID_bits	0
lr_sc_grain	64
PMP_grain	0
PMP_registers	8
PMP_max_page	0

PMP_decompose	F
Sv_modes	0x300
Svnapot_page_mask	0
Svpbmt	F
Svinval	F
Instruction_CSR_Behavior	
wfi_is_nop	F
counteren_mask	0xffffffff
noinhibit_mask	0
cycle_undefined	F
time_undefined	T
instret_undefined	F
hpmcounter_undefined	F
CSR_Masks	
mtvec_mask	0x3fffffff
stvec_mask	0xffffffffffff
mip_mask	0x337
sip_mask	0x103
mtvec_sext	F
stvec_sext	F
MXL_writable	F
SXL_writable	F
UXL_writable	F
Trigger	
trigger_num	0
CSR_Defaults	
mvendorid	0x489
marchid	0x80000007
mimpid	0x4210430
mhartid	0
mtvec	0
Compressed	
Zcea_version	none
Zceb_version	none
Zcee_version	none
Fast Interrupt	
CLICLEVELS	0
sifiveExtensions	
FeatureDisable_Present*	T
BranchPredictionMode_Present*	T
PowerDial_Present*	F
CFLUSH_Present*	T
CEASE_Present*	T

Table 10.15: Parameter values

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

Chapter 11

Execution Modes

Mode	Code	Description
User	0	User mode
Supervisor	1	Supervisor mode
Machine	3	Machine mode

Table 11.1: Modes implemented in: Hart

Chapter 12

Exceptions

Exception	Code	Description
InstructionAddressMisaligned	0	Fetch from unaligned address
InstructionAccessFault	1	No access permission for fetch
IllegalInstruction	2	Undecoded, unimplemented or disabled instruction
Breakpoint	3	EBREAK instruction executed
LoadAddressMisaligned	4	Load from unaligned address
LoadAccessFault	5	No access permission for load
StoreAMOAddressMisaligned	6	Store/atomic memory operation at unaligned address
StoreAMOAccessFault	7	No access permission for store/atomic memory operation
EnvironmentCallFromUMode	8	ECALL instruction executed in User mode
EnvironmentCallFromSMode	9	ECALL instruction executed in Supervisor mode
EnvironmentCallFromMMode	11	ECALL instruction executed in Machine mode
InstructionPageFault	12	Page fault at fetch address
LoadPageFault	13	Page fault at load address
StoreAMOPageFault	15	Page fault at store/atomic memory operation address
SSWInterrupt	65	Supervisor software interrupt
MSWInterrupt	67	Machine software interrupt
STimerInterrupt	69	Supervisor timer interrupt
MTimerInterrupt	71	Machine timer interrupt
SExternalInterrupt	73	Supervisor external interrupt
MExternalInterrupt	75	Machine external interrupt
LocalInterrupt0	80	Local interrupt 0
LocalInterrupt1	81	Local interrupt 1
LocalInterrupt2	82	Local interrupt 2
LocalInterrupt3	83	Local interrupt 3
LocalInterrupt4	84	Local interrupt 4
LocalInterrupt5	85	Local interrupt 5
LocalInterrupt6	86	Local interrupt 6

LocalInterrupt7	87	Local interrupt 7
LocalInterrupt8	88	Local interrupt 8
LocalInterrupt9	89	Local interrupt 9
LocalInterrupt10	90	Local interrupt 10
LocalInterrupt11	91	Local interrupt 11
LocalInterrupt12	92	Local interrupt 12
LocalInterrupt13	93	Local interrupt 13
LocalInterrupt14	94	Local interrupt 14
LocalInterrupt15	95	Local interrupt 15
LocalInterrupt16	96	Local interrupt 16
LocalInterrupt17	97	Local interrupt 17
LocalInterrupt18	98	Local interrupt 18
LocalInterrupt19	99	Local interrupt 19
LocalInterrupt20	100	Local interrupt 20
LocalInterrupt21	101	Local interrupt 21
LocalInterrupt22	102	Local interrupt 22
LocalInterrupt23	103	Local interrupt 23
LocalInterrupt24	104	Local interrupt 24
LocalInterrupt25	105	Local interrupt 25
LocalInterrupt26	106	Local interrupt 26
LocalInterrupt27	107	Local interrupt 27
LocalInterrupt28	108	Local interrupt 28
LocalInterrupt29	109	Local interrupt 29
LocalInterrupt30	110	Local interrupt 30
LocalInterrupt31	111	Local interrupt 31
LocalInterrupt32	112	Local interrupt 32
LocalInterrupt33	113	Local interrupt 33
LocalInterrupt34	114	Local interrupt 34
LocalInterrupt35	115	Local interrupt 35
LocalInterrupt36	116	Local interrupt 36
LocalInterrupt37	117	Local interrupt 37
LocalInterrupt38	118	Local interrupt 38
LocalInterrupt39	119	Local interrupt 39
LocalInterrupt40	120	Local interrupt 40
LocalInterrupt41	121	Local interrupt 41
LocalInterrupt42	122	Local interrupt 42
LocalInterrupt43	123	Local interrupt 43
LocalInterrupt44	124	Local interrupt 44
LocalInterrupt45	125	Local interrupt 45
LocalInterrupt46	126	Local interrupt 46
LocalInterrupt47	127	Local interrupt 47

Table 12.1: Exceptions implemented in: Hart

Chapter 13

Hierarchy of the model

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

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

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

13.1 Level 1: SMP

This level in the model hierarchy has 2 commands.

This level in the model hierarchy has no register groups.

This level in the model hierarchy has one child:

hart0

13.2 Level 2: Hart

This level in the model hierarchy has 5 commands.

This level in the model hierarchy has 7 register groups:

Group name	Registers
Core	33
Floating_point	32
Vector	32
User_Control_and_Status	41
Supervisor_Control_and_Status	10
Machine_Control_and_Status	97
Integration_support	5

Table 13.1: Register groups

This level in the model hierarchy has no children.

Chapter 14

Model Commands

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

14.1 Level 1: SMP

14.1.1 isync

specify instruction address range for synchronous execution

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

Table 14.1: isync command arguments

14.1.2 itrace

enable or disable instruction tracing

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

Table 14.2: itrace command arguments

14.2 Level 2: Hart

14.2.1 dumpTLB

14.2.1.1 Argument description

show TLB contents

14.2.2 getCSRIndex

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

Argument	Type	Description
-name	String	CSR name

Table 14.3: getCSRIndex command arguments

14.2.3 isync

specify instruction address range for synchronous execution

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

Table 14.4: isync command arguments

14.2.4 itrace

enable or disable instruction tracing

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

Table 14.5: itrace command arguments

14.2.5 listCSRs

14.2.5.1 Argument description

List all CSRs in index order

Chapter 15

Registers

15.1 Level 1: SMP

No registers.

15.2 Level 2: Hart

15.2.1 Core

Registers at level:2, type:Hart group:Core

Name	Bits	Initial-Hex	RW	Description
zero	64	0	r-	
ra	64	0	rw	
sp	64	0	rw	stack pointer
gp	64	0	rw	
tp	64	0	rw	
t0	64	0	rw	
t1	64	0	rw	
t2	64	0	rw	
s0	64	0	rw	
s1	64	0	rw	
a0	64	0	rw	
a1	64	0	rw	
a2	64	0	rw	
a3	64	0	rw	
a4	64	0	rw	
a5	64	0	rw	
a6	64	0	rw	
a7	64	0	rw	
s2	64	0	rw	
s3	64	0	rw	
s4	64	0	rw	
s5	64	0	rw	
s6	64	0	rw	
s7	64	0	rw	
s8	64	0	rw	
s9	64	0	rw	
s10	64	0	rw	
s11	64	0	rw	

t3	64	0	rw	
t4	64	0	rw	
t5	64	0	rw	
t6	64	0	rw	
pc	64	0	rw	program counter

Table 15.1: Registers at level 2, type:Hart group:Core

15.2.2 Floating_point

Registers at level:2, type:Hart group:Floating_point

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

Table 15.2: Registers at level 2, type:Hart group:Floating_point

15.2.3 Vector

Registers at level:2, type:Hart group:Vector

Name	Bits	Initial-Hex	RW	Description
v0	512	-	rw	
v1	512	-	rw	

v2	512	-	rw	
v3	512	-	rw	
v4	512	-	rw	
v5	512	-	rw	
v6	512	-	rw	
v7	512	-	rw	
v8	512	-	rw	
v9	512	-	rw	
v10	512	-	rw	
v11	512	-	rw	
v12	512	-	rw	
v13	512	-	rw	
v14	512	-	rw	
v15	512	-	rw	
v16	512	-	rw	
v17	512	-	rw	
v18	512	-	rw	
v19	512	-	rw	
v20	512	-	rw	
v21	512	-	rw	
v22	512	-	rw	
v23	512	-	rw	
v24	512	-	rw	
v25	512	-	rw	
v26	512	-	rw	
v27	512	-	rw	
v28	512	-	rw	
v29	512	-	rw	
v30	512	-	rw	
v31	512	-	rw	

Table 15.3: Registers at level 2, type:Hart group:Vector

15.2.4 User_Control_and_Status

Registers at level:2, type:Hart group:User_Control_and_Status

Name	Bits	Initial-Hex	RW	Description
fflags	64	0	rw	Floating-Point Flags
frm	64	0	rw	Floating-Point Rounding Mode
fcsr	64	0	rw	Floating-Point Control and Status
vstart	64	0	rw	Vector Start Index
vxsat	64	0	rw	Fixed-Point Saturate Flag
vxrm	64	0	rw	Fixed-Point Rounding Mode
vcsr	64	0	rw	Vector Control and Status
cycle	64	0	r-	Cycle Counter
instret	64	0	r-	Instructions Retired
hpmcounter3	64	0	r-	Performance Monitor Counter 3
hpmcounter4	64	0	r-	Performance Monitor Counter 4
hpmcounter5	64	0	r-	Performance Monitor Counter 5
hpmcounter6	64	0	r-	Performance Monitor Counter 6
hpmcounter7	64	0	r-	Performance Monitor Counter 7
hpmcounter8	64	0	r-	Performance Monitor Counter 8
hpmcounter9	64	0	r-	Performance Monitor Counter 9
hpmcounter10	64	0	r-	Performance Monitor Counter 10
hpmcounter11	64	0	r-	Performance Monitor Counter 11

hpmcounter12	64	0	r-	Performance Monitor Counter 12
hpmcounter13	64	0	r-	Performance Monitor Counter 13
hpmcounter14	64	0	r-	Performance Monitor Counter 14
hpmcounter15	64	0	r-	Performance Monitor Counter 15
hpmcounter16	64	0	r-	Performance Monitor Counter 16
hpmcounter17	64	0	r-	Performance Monitor Counter 17
hpmcounter18	64	0	r-	Performance Monitor Counter 18
hpmcounter19	64	0	r-	Performance Monitor Counter 19
hpmcounter20	64	0	r-	Performance Monitor Counter 20
hpmcounter21	64	0	r-	Performance Monitor Counter 21
hpmcounter22	64	0	r-	Performance Monitor Counter 22
hpmcounter23	64	0	r-	Performance Monitor Counter 23
hpmcounter24	64	0	r-	Performance Monitor Counter 24
hpmcounter25	64	0	r-	Performance Monitor Counter 25
hpmcounter26	64	0	r-	Performance Monitor Counter 26
hpmcounter27	64	0	r-	Performance Monitor Counter 27
hpmcounter28	64	0	r-	Performance Monitor Counter 28
hpmcounter29	64	0	r-	Performance Monitor Counter 29
hpmcounter30	64	0	r-	Performance Monitor Counter 30
hpmcounter31	64	0	r-	Performance Monitor Counter 31
vl	64	0	r-	Vector Length
vtype	64	0	r-	Vector Type
vlenb	64	40	r-	Vector Length in Bytes

Table 15.4: Registers at level 2, type:Hart group:User_Control_and_Status

15.2.5 Supervisor_Control_and_Status

Registers at level:2, type:Hart group:Supervisor_Control_and_Status

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

Table 15.5: Registers at level 2, type:Hart group:Supervisor_Control_and_Status

15.2.6 Machine_Control_and_Status

Registers at level:2, type:Hart group:Machine_Control_and_Status

Name	Bits	Initial-Hex	RW	Description
mstatus	64	a 00000000	rw	Machine Status
misa	64	80000000 0034112f	rw	ISA and Extensions
medeleg	64	0	rw	Machine Exception Delegation
mideleg	64	0	rw	Machine Interrupt Delegation
mie	64	0	rw	Machine Interrupt Enable

mtvec	64	0	rw	Machine Trap-Vector Base-Address
mcounteren	64	0	rw	Machine Counter Enable
mcounthinhibit	64	0	rw	Machine Counter Inhibit
mhpmevent3	64	0	rw	Machine Performance Monitor Event Select 3
mhpmevent4	64	0	rw	Machine Performance Monitor Event Select 4
mhpmevent5	64	0	rw	Machine Performance Monitor Event Select 5
mhpmevent6	64	0	rw	Machine Performance Monitor Event Select 6
mhpmevent7	64	0	rw	Machine Performance Monitor Event Select 7
mhpmevent8	64	0	rw	Machine Performance Monitor Event Select 8
mhpmevent9	64	0	rw	Machine Performance Monitor Event Select 9
mhpmevent10	64	0	rw	Machine Performance Monitor Event Select 10
mhpmevent11	64	0	rw	Machine Performance Monitor Event Select 11
mhpmevent12	64	0	rw	Machine Performance Monitor Event Select 12
mhpmevent13	64	0	rw	Machine Performance Monitor Event Select 13
mhpmevent14	64	0	rw	Machine Performance Monitor Event Select 14
mhpmevent15	64	0	rw	Machine Performance Monitor Event Select 15
mhpmevent16	64	0	rw	Machine Performance Monitor Event Select 16
mhpmevent17	64	0	rw	Machine Performance Monitor Event Select 17
mhpmevent18	64	0	rw	Machine Performance Monitor Event Select 18
mhpmevent19	64	0	rw	Machine Performance Monitor Event Select 19
mhpmevent20	64	0	rw	Machine Performance Monitor Event Select 20
mhpmevent21	64	0	rw	Machine Performance Monitor Event Select 21
mhpmevent22	64	0	rw	Machine Performance Monitor Event Select 22
mhpmevent23	64	0	rw	Machine Performance Monitor Event Select 23
mhpmevent24	64	0	rw	Machine Performance Monitor Event Select 24
mhpmevent25	64	0	rw	Machine Performance Monitor Event Select 25
mhpmevent26	64	0	rw	Machine Performance Monitor Event Select 26
mhpmevent27	64	0	rw	Machine Performance Monitor Event Select 27
mhpmevent28	64	0	rw	Machine Performance Monitor Event Select 28
mhpmevent29	64	0	rw	Machine Performance Monitor Event Select 29
mhpmevent30	64	0	rw	Machine Performance Monitor Event Select 30
mhpmevent31	64	0	rw	Machine Performance Monitor Event Select 31
mscratch	64	0	rw	Machine Scratch
mepc	64	0	rw	Machine Exception Program Counter
mcause	64	0	rw	Machine Cause
mtval	64	0	rw	Machine Trap Value
mip	64	0	rw	Machine Interrupt Pending
pmpcfg0	64	0	rw	Physical Memory Protection Configuration 0
pmpcfg2	64	0	rw	Physical Memory Protection Configuration 2
pmpaddr0	64	0	rw	Physical Memory Protection Address 0
pmpaddr1	64	0	rw	Physical Memory Protection Address 1
pmpaddr2	64	0	rw	Physical Memory Protection Address 2
pmpaddr3	64	0	rw	Physical Memory Protection Address 3
pmpaddr4	64	0	rw	Physical Memory Protection Address 4
pmpaddr5	64	0	rw	Physical Memory Protection Address 5
pmpaddr6	64	0	rw	Physical Memory Protection Address 6
pmpaddr7	64	0	rw	Physical Memory Protection Address 7
pmpaddr8	64	0	rw	Physical Memory Protection Address 8
pmpaddr9	64	0	rw	Physical Memory Protection Address 9
pmpaddr10	64	0	rw	Physical Memory Protection Address 10
pmpaddr11	64	0	rw	Physical Memory Protection Address 11
pmpaddr12	64	0	rw	Physical Memory Protection Address 12
pmpaddr13	64	0	rw	Physical Memory Protection Address 13
pmpaddr14	64	0	rw	Physical Memory Protection Address 14
pmpaddr15	64	0	rw	Physical Memory Protection Address 15
bpm*	64	0	rw	SiFive Branch Prediction Mode

featureDisable*	64	0	rw	SiFive FeatureDisable
mcycle	64	0	rw	Machine Cycle Counter
minstret	64	0	rw	Machine Instructions Retired
mhpmcounter3	64	0	rw	Machine Performance Monitor Counter 3
mhpmcounter4	64	0	rw	Machine Performance Monitor Counter 4
mhpmcounter5	64	0	rw	Machine Performance Monitor Counter 5
mhpmcounter6	64	0	rw	Machine Performance Monitor Counter 6
mhpmcounter7	64	0	rw	Machine Performance Monitor Counter 7
mhpmcounter8	64	0	rw	Machine Performance Monitor Counter 8
mhpmcounter9	64	0	rw	Machine Performance Monitor Counter 9
mhpmcounter10	64	0	rw	Machine Performance Monitor Counter 10
mhpmcounter11	64	0	rw	Machine Performance Monitor Counter 11
mhpmcounter12	64	0	rw	Machine Performance Monitor Counter 12
mhpmcounter13	64	0	rw	Machine Performance Monitor Counter 13
mhpmcounter14	64	0	rw	Machine Performance Monitor Counter 14
mhpmcounter15	64	0	rw	Machine Performance Monitor Counter 15
mhpmcounter16	64	0	rw	Machine Performance Monitor Counter 16
mhpmcounter17	64	0	rw	Machine Performance Monitor Counter 17
mhpmcounter18	64	0	rw	Machine Performance Monitor Counter 18
mhpmcounter19	64	0	rw	Machine Performance Monitor Counter 19
mhpmcounter20	64	0	rw	Machine Performance Monitor Counter 20
mhpmcounter21	64	0	rw	Machine Performance Monitor Counter 21
mhpmcounter22	64	0	rw	Machine Performance Monitor Counter 22
mhpmcounter23	64	0	rw	Machine Performance Monitor Counter 23
mhpmcounter24	64	0	rw	Machine Performance Monitor Counter 24
mhpmcounter25	64	0	rw	Machine Performance Monitor Counter 25
mhpmcounter26	64	0	rw	Machine Performance Monitor Counter 26
mhpmcounter27	64	0	rw	Machine Performance Monitor Counter 27
mhpmcounter28	64	0	rw	Machine Performance Monitor Counter 28
mhpmcounter29	64	0	rw	Machine Performance Monitor Counter 29
mhpmcounter30	64	0	rw	Machine Performance Monitor Counter 30
mhpmcounter31	64	0	rw	Machine Performance Monitor Counter 31
mvendorid	64	489	r-	Vendor ID
marchid	64	80000007	r-	Architecture ID
mimpid	64	4210430	r-	Implementation ID
mhartid	64	0	r-	Hardware Thread ID

Table 15.6: Registers at level 2, type:Hart group:Machine_Control_and_Status

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

15.2.7 Integration support

Registers at level:2, type:Hart group:Integration_support

Name	Bits	Initial-Hex	RW	Description
LRSCAddress	64	ffffff ffffff	rw	LR/SC active lock address
commercial	8	0	r-	Commercial feature in use
PTWStage	8	0	r-	PTW active stage (0:none 1:HS 2:VS 3:G)
PTWInputAddr	64	0	r-	PTW input address
PTWLevel	8	0	r-	PTW active level

Table 15.7: Registers at level 2, type:Hart group:Integration_support