RISC-V S-mode Physical Memory Protection (SPMP)

Editor - Dong Du, RISC-V SPMP Task Group

Version 0.9.1, 10/2023: This document is in development. Assume everything can change. See http://riscv.org/spec-state for details.

Table of Contents

Preamble

This document is in the Development state

Assume everything can change. This draft specification will change before being accepted as standard, so implementations made to this draft specification will likely not conform to the future standard.

Copyright and license information

This specification is licensed under the Creative Commons Attribution 4.0 International License (CC-BY 4.0). The full license text is available at creativecommons.org/licenses/by/4.0/.

Copyright 2023 by RISC-V International.

Contributors

The proposed SPMP specifications (non-ratified, under discussion) has been contributed to directly or indirectly by:

- Dong Du, Editor <dd_nirvana@sjtu.edu.cn>
- Bicheng Yang
- Nick Kossifidis
- Andy Dellow
- Manuel Offenberg
- Allen Baum
- Bill Huffman
- Xu Lu
- Wenhao Li
- Yubin Xia
- Joe Xie
- Paul Ku
- Jonathan Behrens
- Robin Zheng
- Zeyu Mi
- Fabrice Marinet

Chapter 1. Introduction

This document describes RISC-V S-mode Physical Memory Protection (SPMP) proposal to provide isolation when MMU is unavailable or disabled. RISC-V based processors recently stimulated great interest in the emerging internet of things (IoT). However, as page-based virtual memory (MMU) is usually unavailable on IoT devices, it is hard to isolate the S-mode OSes (e.g., RTOS) and user-mode applications. To support secure processing and isolate faults of U-mode software, the SPMP is desirable to enable S-mode OS to limit the physical addresses accessible by U-mode software on a hart.

Chapter 2. S-mode Physical Memory Protection (SPMP)

An optional RISC-V S-mode Physical Memory Protection (SPMP) provides per-hart supervisor-mode control registers to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region. The SPMP is also applied to data accesses in M-mode when the MPRV bit in mstatus is set and the MPP field in mstatus contains S or U.

Like PMP, the granularity of SPMP access control settings is platform-specific and, within a platform, may vary by physical memory region. However, the standard SPMP encoding support regions as small as four bytes.

The implementation can perform SPMP checks in parallel with PMA and PMP. The SPMP exception reports have higher priority than PMP or PMA exceptions (e.g., an SPMP exception will be raised if the access violates both SPMP and PMP).

SPMP checks will be applied to all accesses for U mode and S mode, depending on the values in the configuration registers. M-mode accesses are not affected and always pass SPMP permission checks. SPMP registers can always be modified by M-mode and S-mode software. SPMP can grant permissions to U-mode, which has none by default. SPMP can also revoke permissions from S-mode.

2.1. Requirements

1) S mode should be implemented 2) `sstatus.SUM` should be WARL.

2.2. S-mode Physical Memory Protection CSRs

Like PMP, SPMP entries are described by an 8-bit configuration register and one XLEN-bit address register. Some SPMP settings additionally use the address register associated with the preceding SPMP entry. The number of SPMP entries can vary by implementation, and up to 64 SPMP entries are supported in the standard.

The terms, entry and rule, are similar to ePMP.

The SPMP configuration registers are packed into CSRs the same way as PMP. For RV32, 16 CSRs, spmpcfg0-spmpcfg15, hold the configurations spmp0cfg-spmp63cfg for the 64 SPMP entries. For RV64, even numbered CSRs (i.e., spmpcfg0, spmpcfg2, ..., spmpcfg14) hold the configurations for the 64 SPMP entries; odd numbered CSRs (e.g., spmpcfg1) are illegal. Figures 1 and 2 demonstrate the first 16 entries of SPMP. The layout of the rest entries is similar.

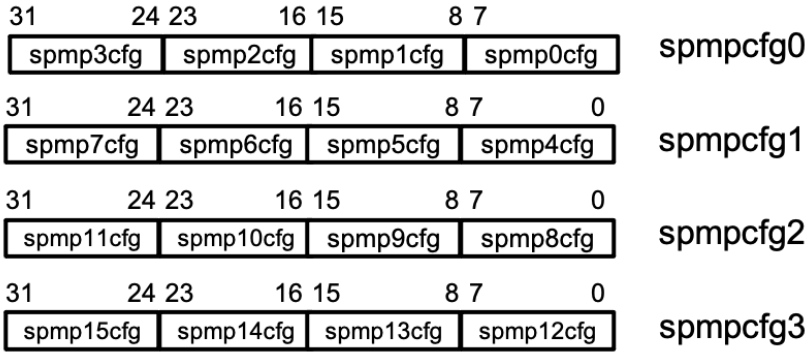

Figure 1. RV32 SPMP configuration CSR layout

Figure 2. RV64 SPMP configuration CSR layout

The SPMP address registers are CSRs named spmpaddr0-spmpaddr63. Each SPMP address register encodes bits 33-2 of 34-bit physical address for RV32, as shown in Figure 3. For RV64, each SPMP address encodes bits 55–2 of a 56-bit physical address, as shown in Figure 4. Fewer address bits may be implemented for specific reasons, e.g., systems with smaller physical address space. Implemented address bits must be contiguous and go from lower to higher bits.

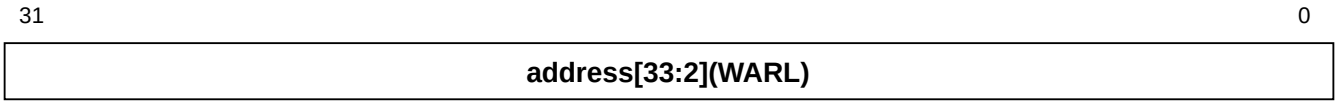

Figure 3. SPMP address register format, RV32

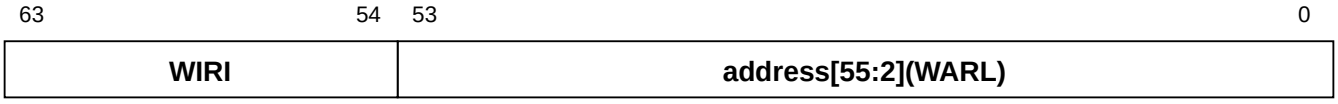

Figure 4. SPMP address register format, RV64

The layout of SPMP configuration registers is the same as PMP configuration registers, as shown in Figure 5. The register is WARL. The rules and encodings for permission are explained in section 2.4, which resembles the encoding of ePMP (except SPMP does not use locked rules).

1. The S bit marks a rule as **S-mode-only** when set and **U-mode-only** when unset.
2. Bit 5 and 6 are reserved for future use.
3. The A field will be described in the following sections (2.3).
4. The R/W/X bits control read, write, and instruction execution permissions.

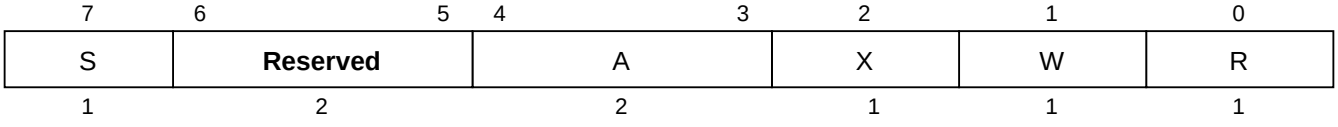

Figure 5. SPMP configuration register format

The number of SPMP entries: Implementations may implement zero, 16, or 64 SPMP entries. SPMP

CSRs are accessible to M-mode and S-mode. **The reset state**: On system reset, the A field of spmp[i]cfg should be zero.

SPMP CSRs should be allocated contiguously starting with the lowest CSR number.

2.3. Address Matching

The A field in an SPMP entry's configuration register encodes the address-matching mode of the associated SPMP address register. It is the same as PMP/ePMP.

Please refer to the "Address Matching" subsection of PMP in the riscv-privileged spec for detailed information.

2.4. Encoding of Permissions

SPMP has three kinds of rules: **S-mode-only**, **U-mode-only** and **Shared-Region** rules.

1. An **S-mode-only** rule is **enforced** on Supervisor mode and **denied** on User mode.
2. A **U-mode-only** rule is **enforced** on User modes and **denied/enforced** on Supervisor mode depending on the value of `sstatus.SUM` bit:
   - If `sstatus.SUM` is set, a U-mode-only rule is enforced without code execution permission on Supervisor mode to ensure supervisor mode execution protection.
   - If `sstatus.SUM` is unset, a U-mode-only rule is denied on Supervisor mode.
3. A **Shared-Region** rule is enforced on both Supervisor and User modes, with restrictions depending on the `spmpcfg.S` and `spmpcfg.X` bits:
   - If `spmpcfg.S` is not set, the region can be used for sharing data between S-mode and U-mode, yet not executable. S-mode has RW permission to that region, and U-mode has read-only permission if `spmpcfg.X` is not set or RW permission if `spmpcfg.X` is set.
   - If `spmpcfg.S` is set, the region can be used for sharing code between S-mode and U-mode, yet not writeable. S-mode and U-mode have execute permission to the region, and S-mode may also have read permission if `spmpcfg.X` is set.
   - The encoding `spmpcfg.SRWX=1111` can be used for sharing data between S-mode and U-mode, where both modes only have read-only permission to the region.
4. The encoding `spmpcfg.SRWX=1000` is reserved for future standard use.

The encoding and results are shown in the table:

	S=0			S=1		
spmpcfg	Smode	Smode	Umode	Smode	Smode	Umode
RWX	Sum=0	Sum=1	Sum=x	Sum=0	Sum=1	Sum=x
R - -	Deny	EnforceNoX	Enforce	Enforce	Enforce	Deny
R - X	Deny	EnforceNoX	Enforce	Enforce	Enforce	Deny
- - X	Deny	EnforceNoX	Enforce	Enforce	Enforce	Deny
- - -	Deny	EnforceNoX	Enforce	RSVD		
R W -	Deny	EnforceNoX	Enforce	Enforce	Enforce	Deny
R W X	Deny	EnforceNoX	Enforce	shr RO		
- W X	shr RW			shr RX		shr X
- W -	shr RW		shr RO	shr X		

Figure 6. SPMP Encoding Table

Deny: Access not allowed.

Enforce: The R/W/X permissions are enforced on accesses.

EnforceNoX: The R/W permissions are enforced on accesses, while the X bit is forced to be zero.

SHR: It is shared between S/U modes with X, RX, RW, or ReadOnly privileges.

RSVD: It is reserved for future use.

SUM bit: We re-use the sstatus.SUM (allow Supervisor User Memory access) bit of modifying the privilege with which S-mode loads and stores access to physical memory. The semantics of SUM in SPMP is consistent with those in Sv.

2.5. Priority and Matching Logic

M-mode accesses are always considered to pass SPMP checks. If PMP/ePMP is implemented, accesses succeed only if both PMP/ePMP and SPMP permission checks pass.

Like PMP entries, SPMP entries are also statically prioritized. The lowest-numbered SPMP entry that matches any byte of access (indicated by an address and the accessed length) determines whether that access is allowed or fails. The SPMP entry must match all bytes of access, or the access fails, irrespective of the S, R, W, and X bits.

On some implementations, misaligned loads, stores, and instruction fetches may also be decomposed into multiple accesses, some of which may succeed before an exception occurs. In particular, a portion of a misaligned store that passes the SPMP check may become visible, even if another portion fails the SPMP check. The same behavior may manifest for floating-point stores wider than XLEN bits (e.g., the FSD instruction in RV32D), even when the store address is naturally aligned.

1. If the privilege mode of the access is M, the access is `allowed`;
2. If the privilege mode of the access is S and no SPMP entry matches, the access is `allowed`;
3. If the privilege mode of the access is U and no SPMP entry matches, but at least one SPMP entry is implemented, the access is `denied`;
4. Otherwise, the access is checked according to the permission bits in the matching SPMP entry. It is allowed if it satisfies the permission checking with the S, R, W, or X bit corresponding to the access

type.

2.6. SPMP and Paging

The table below shows which mechanism to use. (Assume both paged virtual memory and SPMP are implemented.)

satp	Isolation mechanism
satp.mode == Bare	SPMP only
satp.mode != Bare	Paged Virtual Memory only

We do not allow both SPMP and paged virtual memory permissions to be actived at the same time now because: (1) It will introduce one more layer to check permission for each memory access. This issue will be more serious for a guest OS that may have host SPMP and guest SPMP. (2) Paged virtual memory can provide sufficient protection.

That means SPMP is enabled when `satp.mode==Bare` and SPMP is implemented.

Please refer to Table "Encoding of satp MODE field" in the riscv-privileged spec for detailed information on the satp.MODE field.

If page-based virtual memory is not implemented, or when it is disabled, memory accesses check the SPMP settings synchronously, so no fence is needed.

2.7. Exceptions

Failed accesses generate an exception. SPMP follows the strategy that uses different exception codes for different cases, i.e., load, store/AMO, instruction faults for memory load, memory store/AMO and instruction fetch, respectively.

The SPMP reuses exception codes of page fault for SPMP fault. Because page fault is typically delegated to S-mode, so does SPMP fault, we can benefit from reusing page fault. S-mode software(i.e., OS) can distinguish page fault from SPMP fault by checking satp.mode (as mentioned in 2.6, SPMP and paged virtual memory will not be activated simultaneously). **SPMP proposes to rename page fault to SPMP/page fault for clarity**.

Note that a single instruction may generate multiple accesses, which may not be mutually atomic.

Table of renamed exception codes:

Interrupt	Exception Code	Description
0	12	Instruction SPMP/page fault
0	13	Load SPMP/page fault
0	15	Store/AMO SPMP/page fault

Please refer to Table "Supervisor cause register (scause) values after trap" in the riscv-privileged spec for detailed information on exception codes.

Delegation: Unlike PMP, which uses access faults for violations, SPMP uses SPMP/page faults for violations. The benefit of using SPMP/page faults is that we can delegate the violations caused by SPMP to S-mode, while the access violations caused by PMP can still be handled by machine mode.

2.8. Context Switching Optimization

With SPMP, each context switch requires the OS to store 64 address registers and 8 configuration registers (RV64), which is costly and unnecessary. So the SPMP proposes an optimization to minimize the overhead caused by context switching.

We add two CSRs called ***spmpswitch0*** and ***spmpswitch1***, which are XLEN-bit read/write registers, as shown in Figure 7. For RV64, only ***spmpswitch0*** is used. Each bit of this register holds the on/off status of the corresponding SPMP entry. During the context switch, the OS can store and restore spmpswitch as part of the context. An SPMP entry is activated only when both corresponding bits in spmpswitch and A field of spmp[i]cfg are set. (i.e., spmpswitch[i] & spmp[i]cfg.A!=0)

Figure 7. SPMP domain switch register format (RV64)

If the `spmpswitch` is implemented, and `spmpcfg[i].A == TOR`, the entry matches any address y such that spmpaddr[i−1] ≤ y < spmpaddr[i] (irrespective of values of spmpcfg[i-1] and spmpswitch[i-1]).

2.9. Access Methods of SPMP CSRs

How SPMP CSRs are accessed depends on whether the `Sscsrind` extension is implemented or not.

Indirect CSR access: The SPMP supports indirect CSR access if the `Sscsrind` extension is implemented. The `Sscsrind` defines 1 select CSR (`siselect`) and 6 alias CSRs (`sireg[i]`). Each combination of `siselect` and `sireg[i]` represents an access to the corresponding SPMP CSR.

siselect number	indirect CSR access of sireg[i]
siselect#1	sireg[1-6] → spmpcfg[0-5]
siselect#2	sireg[1-6] → spmpcfg[6-11]
siselect#3	sireg[1-4] → spmpcfg[12-15]
siselect#4	sireg[1-6] → spmpaddr[0-5]
siselect#5	sireg[1-6] → spmpaddr[6-11]
siselect#6	sireg[1-6] → spmpaddr[12-17]
siselect#7	sireg[1-6] → spmpaddr[18-23]
siselect#8	sireg[1-6] → spmpaddr[24-29]
siselect#9	sireg[1-6] → spmpaddr[30-35]
siselect#10	sireg[1-6] → spmpaddr[36-41]
siselect#11	sireg[1-6] → spmpaddr[42-47]

siselect number	indirect CSR access of sireg[i]
siselect#12	sireg[1-6] → spmpaddr[48-53]
siselect#13	sireg[1-6] → spmpaddr[54-59]
siselect#14	sireg[1-4] → spmpaddr[60-63]
siselect#15	sireg[1-2] → spmpswitch[0-1]

Direct CSR access: SPMP CSRs can be accessed directly with corresponding CSR numbers if the `Sscsrind` extension is not implemented.

The specific value of `siselect#1-15` will be allocated after review by the Arch Review Committee.

Please refers to the specification of the `Sscsrind` extension for details of indirect CSR access. github.com/riscv/riscv-indirect-csr-access

Chapter 3. Summary of Hardware Changes

3.1. Sharing Registers and Comparators with PMP

Given that PMP and SPMP have similar layout of address/config registers and the same address matching logic. Reusing registers and comparators between PMP and SPMP is benefitial to reduce hardware cost.

Implementations can consider PMP/SPMP (PMPs for short) entries as a resource pool. A new M-mode CSR called `mpmpdeleg` is introduced to control which subset of the PMPs entries is delegated to S-mode (i.e., to be used as SPMP entries).

Specificly, an implementation with 64 PMPs entries in total, by setting `mpmpdeleg` to 32 at runtime, the PMPs[0]-PMPs[31] is used as PMP entries, while PMPs[32]-PMPs[63] is used as SPMP entries.

An implementation can still have a fixed number of PMP and SPMP entries by hardwiring the `mpmpdeleg` to some constant.

In the case where the Sscsrind extension is not implemented. S-mode software sees SPMP[mpmpdeleg]-SPMP[max_PMPs_nums - 1]. This is because keeping the SPMP index starting at 0 adds extra overhead to the address decoding path.

3.2. New CSRs and Exception Code

Item	Changes
CSR for delegating PMPs entries	1 new CSR
CSRs for SPMP address	64 new CSRs
CSRs for SPMP configuration	16 new CSRs for RV32 and 8 for RV64
CSR for Domain switch	2 new CSRs for RV32 and 1 for RV64
Renamed exception code	***Instruction page fault*** renamed to ***Instruction SPMP/page fault*** ***Load page fault*** renamed to ***Load SPMP/page fault*** ***Store/AMO page fault*** renamed to ***Store/AMO SPMP/page fault***

Chapter 4. Interaction with other proposals

This section discusses how SPMP interacts with other proposals.

RISC-V PMP enhancements: SPMP is compatible with the ePMP proposal and uses almost the same encoding as ePMP.

J-extension pointer masking proposal: When both PM and SPMP are used, SPMP checking should be performed using the actual addresses generated by PM (pointer masking).

Hypervisor extension: There are extensions (vspmp and hgpmp) in development to support SPMP in guest OS and hypervisors.