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

Editor - Dong Du, RISC-V SPMP Task Group

Version 0.9.2, 5/2024: 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

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, Editor <bichengyang@sjtu.edu.cn>
- 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
- José Martins
- Thomas Roecker
- Sandro Pinto
- Rongchuan Liu
- Noureddine Ait Said

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) and automotive devices. However, page-based virtual memory (MMU) is usually undesired in order to meet resource and latency constraints. It is hard to isolate the S-mode OSes (e.g., RTOS) and user-mode applications for such devices. 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.

If PMP/ePMP is implemented, accesses succeed only if both PMP/ePMP and SPMP permission checks pass. The implementation can perform SPMP checks in parallel with PMA and PMP. The SPMP exception reports have higher priority than PMP or PMA exceptions (i.e., if the access violates both SPMP and PMP/PMA, the SPMP exception will be reported).

SPMP checks will be applied to all accesses whose effective privilege mode is S or U, including instruction fetches and data accesses in S and U mode, and data accesses in M-mode when the MPRV bit in mstatus is set and the MPP field in mstatus contains S or U.

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. The `sstatus.SUM` (permit Supervisor User Memory access) bit must be **writeable**.

    The Privileged Architecture specification states the following

    > `SUM` has no effect when page-based virtual memory is not in effect, nor when executing in U-mode. `SUM` is read-only 0 if `satp.MODE` is read-only 0.
    >
    > — Supervisor-Level ISA, Version 1.13 >> "Memory Privilege in \`sstatus\` Register"

    In SPMP, this bit modifies the privilege with which S-mode loads and stores access to physical memory, hence the need to make it writeable.

3. The `sstatus.MXR` (Make eXecutable Readable) bit must be **writeable**.

    The Privileged Architecture specification states that

    > `MXR` has no effect when page-based virtual memory is not in effect.
    >
    > — Machine-Level ISA, Version 1.13 >> "Memory Privilege in \`mstatus\` Register"

In SPMP, the `MXR` bit modifies the privilege with which loads access physical memory. Its semantics are consistent with those of the Machine-Level ISA.

In SPMP, this bit is made writeable to support M-mode emulation handlers where instructions are read with `MXR=1` and `MPRV=1`.

2.2. Supervisor Security Configuration (sseccfg) CSR

Supervisor Security Configuration (sseccfg) is a new Supervisor mode CSR used for configuring SPMP features. All sseccfg fields defined on this proposal are WARL, and the remaining bits are reserved for future standard use and should always read zero. This CSR has two fields:

- Bit 0. `sseccfg.SMWP` (Supervisor Memory Access Whitelist Policy): When set, this bit changes the default SPMP policy for S-Mode when accessing memory regions that don't have a matching rule to denied instead of ignored. This bit must reset to 0.
- Bit 1. `sseccfg.SMAL` (SPMP Match-Any Logic): When set, this bit changes the entry matching logic for SPMP from the default priority-based matching, where the lowest-numbered SPMP entry that matches any byte of the access determines the permissions for that access, to a match-any logic where the final permissions for the access are the union of the permissions of any entry that matches any byte of the access. This bit must reset to 0. If match-any logic is not implemented this bit should always read as zero.

For RV64 `sseccfg` is 64 bits wide, while for RV32 `sseccfg` is divided into `sseccfg` (lower 32 bits) and `sseccfgh` (upper 32 bits).

The `sseccfg` register must be cleared on reset.

2.3. 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 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. Figure 1 and Figure 2 demonstrate the first 16 entries of SPMP. The layout of the rest entries is similar.

> The terms, entry and rule, are similar to ePMP.
>
> The implementation should decode all SPMP CSRs, and it can modify the number of **writable SPMP entries** while the remaining SPMP CSRs are read-only zero.
>
> The lowest-numbered SPMP entries must be implemented first.

31 24	23 16	15 8	7 0	
spmp3cfg	spmp2cfg	spmp1cfg	spmp0cfg	spmpcfg0
8	8	8	8	

31 24	23 16	15 8	7 0	
spmp7cfg	spmp6cfg	spmp5cfg	spmp4cfg	spmpcfg1
8	8	8	8	

⋮

31 24	23 16	15 8	7 0	
spmp63cfg	spmp62cfg	spmp61cfg	spmp60cfg	spmpcfg15
8	8	8	8	

Figure 1. RV32 SPMP configuration CSR layout.

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
spmp7cfg	spmp6cfg	spmp5cfg	spmp4cfg	spmp3cfg	spmp2cfg	spmp1cfg	spmp0cfg	spmpcfg0
8	8	8	8	8	8	8	8	

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
spmp15cfg	spmp14cfg	spmp13cfg	spmp12cfg	spmp11cfg	spmp10cfg	spmp9cfg	spmp8cfg	spmpcfg2
8	8	8	8	8	8	8	8	

⋮

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
spmp63cfg	spmp62cfg	spmp61cfg	spmp60cfg	spmp59cfg	spmp58cfg	spmp57cfg	spmp56cfg	spmpcfg14
8	8	8	8	8	8	8	8	

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. The number of address bits should be the same for all **writable SPMP entries**. Implemented address bits must extend to the LSB format, except as otherwise permitted by granularity rules. See the Privileged Architecture specification, Section 3.7: Physical Memory Protection, Address Matching.

31 0
address[33:2] **(WARL)**
32

Figure 3. SPMP address register format, RV32.

63 54	53 0
WARL	address[55:2] **(WARL)**
10	54

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.

7	6 5	4 3	2	1	0
S **(WARL)**	Reserved	A **(WARL)**	X **(WARL)**	W **(WARL)**	R **(WARL)**
1	2	2	1	1	1

Figure 5. SPMP configuration register format.

2.4. 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.

> Software may determine the SPMP granularity by writing zero to `spmp0cfg`, then writing all ones to `spmpaddr0`, then reading back `spmpaddr0`. If G is the index of the least-significant bit set, the SPMP granularity is 2^{G+2}

2.5. 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` is used for backward compatibility, where both S-mode and U-mode have RWX permissions.

4. The encoding `spmpcfg.SRWX=1000` is reserved for future standard use.

The encoding and results are shown in the Figure 6:

	spmpcfg.S=0			spmpcfg.S=1		
spmpcfg	S mode Access	S mode Access	U mode Access	S mode Access	S mode Access	U mode Access
RWX	sstatus.SUM=0	sstatus.SUM=1	sstatus.SUM=x	sstatus.SUM=0	sstatus.SUM=1	sstatus.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		
RW -	Deny	EnforceNoX	Enforce	Enforce	Enforce	Deny
RWX	Deny	EnforceNoX	Enforce	Enforce	Enforce	Deny
- WX	SHR RW			SHR RX		SHR X
- W -	SHR RW		SHR RO	SHR X		

Figure 6. SPMP Encoding Table

Deny: Access fails.

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: The SPMP uses the `sstatus.SUM` (permit Supervisor User Memory access) bit to modify the privilege with which S-mode loads and stores access to physical memory. The semantics of `sstatus.SUM` in SPMP are consistent with those of the Machine-Level ISA (see the "Memory Privilege in mstatus Register" subsection).

2.6. Matching Logic

By default, when `sseccfg.SMAL` is clear:

- SPMP entries are statically prioritized, similar to PMP entries
- 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 denied
- The SPMP entry must match all bytes of access, or the access fails
- This matching is done irrespective of the S, R, W, and X bits

When `sseccfg.SMAL` is set:

- The union of permissions of any entries matching the access determines whether the access is allowed or denied

- All SPMP entries must match all bytes of the access, or the access fails
- This matching is done 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 stores wider than XLEN bits (e.g., the FSD instruction in RV32D), even when the store address is naturally aligned.

1. If the effective privilege mode of the access is M, the access is `allowed`;
2. If the effective privilege mode of the access is S and no SPMP entry matches, if `sseccfg.SMWP` is clear the access is `allowed`, otherwise if `sseccfg.SMWP` is set, the access is denied;
3. If the effective 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 SRWX encoding corresponding to the access type.

2.7. 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

SPMP and paged virtual memory cannot be active simultaneously for two reasons:

1. An additional permission check layer would be introduced for each memory access.
2. Sufficient protection is provided by paged virtual memory.

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 implemented, an `SFENCE.VMA` instruction with `rs1=x0` and `rs2=x0` is needed after writing the SPMP CSRs. If page-based virtual memory is not implemented, memory accesses check the SPMP settings synchronously, so no fence is needed. Please refer to hypervisor extension for additional synchronization requirements when hypervisor is implemented.

2.8. Exceptions

When an access fails, SPMP generates an exception based on the access type (i.e., load accesses, store/AMO accesses, and instruction fetches). Each exception has a different code.

The SPMP reuses page fault exception codes for SPMP faults since page faults are typically delegated to S-mode. S-mode software (i.e., OS) can distinguish between SPMP and page faults by checking `satp.mode`, since SPMP and paged virtual memory cannot be active simultaneously (as described in Section 2.7). **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 the violations caused by SPMP can be delegated to S-mode, while the access violations caused by PMP can still be handled by machine mode.

2.9. Context Switching Optimization

Context switching with SPMP requires storing 64 address and 8 configuration registers (RV64), creating significant overhead. To optimize this:

- In RV32: two XLEN-bit read/write CSRs called `spmpswitch0` and `spmpswitch1` are added, as depicted in Figure 7.
- In RV64: one XLEN-bit read/write CSR called `spmpswitch0` is added, as depicted in Figure 8.

Each bit controls the activation of its corresponding SPMP entry. An entry is active only when both its `spmpswitch[i]` bit and `spmp[i]cfg.A` field are set, i.e., `spmpswitch[i] & spmp[i]cfg.A!=0`.

31 … 0
WARL

Figure 7. SPMP domain switch registers (`spmpswitch0` and `spmpswitch1`), RV32.

63 … 0
WARL

Figure 8. SPMP domain switch register (`spmpswitch0`), RV64.

When `spmpswitch` is implemented and `spmpcfg[i].A == TOR`, an entry matches any address y where:

- `spmpaddr[i-1]` $\le y <$ `spmpaddr[i]`
- This matching occurs regardless of `spmpcfg[i-1]` and `spmpswitch[i-1]` values

The `spmpswitch` registers must be cleared on reset.

2.10. 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#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 refer to the `Sscsrind` extension specification for details on indirect CSR accesses: github.com/riscv/riscv-indirect-csr-access

Chapter 3. Extension of Resource Sharing

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 may be benefitial (in some cases) to save hardware resources. This section introduces the resource sharing extension that can support dynamic reallocation of hardware resource between PMP/ePMP and SPMP. Notably, **this extension is not mandatory** and a specific implementation can still statically implement the numbers of regions in PMP/ePMP and SPMP.

Implementations can consider PMP/SPMP entries as a resource pool (called PMP Resource). Specifically, each PMP Resource entry consists of an address CSR, a configuration CSR, and associated micro-architecture state. A new M-mode CSR called `mpmpimppart` is introduced to control the sharing of PMP Resource between PMP and SPMP.

The 16-bit CSR shown in Figure 9 has two fields:

1. `pmpimp`: 7-bit, allowing a value of 0—64 to specify the number of PMP entries
2. `spmpimp`: 7-bit, allowing a value of 0—64 to specify the number of SPMP entries

The above two fields allow each of the PMP and SPMP to be of different entries, giving extra flexibility.

15 14	13 7	6 0
reserved	pmpimp	spmpimp

Figure 9. mpmpimppart register format

Constraints:

1. The values of `pmpimp` and `spmpimp` in `mpmpimppart`, under RV32, can only be a multiple of 4; under RV64, it can only be a multiple of 8. This design avoids sharing the same configuration CSR between S-mode and M-mode.
2. The values of `pmpimp` and `spmpimp` in `mpmpimppart` cannot be larger than 64 (the maximum number of supported PMP/SPMP entries).
3. The sum of `pmpimp` and `spmpimp` cannot be larger than the total number of PMP Resources.
4. Not all physical address bits may be implemented, so the `mpmpimppart` is WARL. This can be utilized for feature discovery.

Addressing:

Notably, the mapping between PMP Resources and PMP/SPMP entries is defined by the specific implementation. PMP entries will be supported contiguously, beginning with the lowest CSR number. Unlike PMP, SPMP entries **may not** start at the lowest CSR number and might include non-contiguous valid entries. For instance, given an implementation with a total of 64 PMP Resource entries, if both `pmpimp` and `spmpimp` are set to 32 during runtime, `PMPResource[0]` to `PMPResource[31]` would map to `PMP[0]` to `PMP[31]`. The remaining entries, `PMPResource[32]` to `PMPResource[63]`, could be mapped as either `SPMP[0]` to `SPMP[31]` or `SPMP[32]` to `SPMP[63]`, illustrating the flexibility in resource sharing design and implementation. If the SPMP entry with lowest CSR number is configured with TOR address-matching mode, zero is used for the lower bound.

Another example is that, an implementation with a total of 96 PMP Resource entries, if `pmpimp` is set to 48 and `spmpimp` is set to 48 during runtime, `PMPResource[0]` to `PMPResource[47]` would map to `PMP[0]` to `PMP[47]`. `PMPResource[48]` to `PMPResource[63]` may map to `SPMP[48]` to `SPMP[63]`, and `PMPResource[64]` to `PMPResource[95]` may map to `SPMP[0]` to `SPMP[31]`. `SPMP[32]` to `SPMP[47]` are not implemented (from software perspective) in this case.

Re-configuration:

M-mode software can re-configure the entries for PMP and SPMP by modifying the `mpmpimppart` CSR. A re-configuration will be ignored if it will change the PMP resource of a locked PMP entry to an SPMP entry.

> Different mapping implementations have their own trade-offs. For example, mapping `PMPResource[32]` to `PMPResource[63]` onto `SPMP[0]` to `SPMP[31]` could necessitate additional hardware resources to translate the SPMP CSR index to the corresponding `PMPResource`. This might look something like the formula `pmp-resource-index = spmp-index + pmpimp`. Another approach is the non-zero-start mapping. As an example, `PMPResource[32]` to `PMPResource[63]` could be mapped to `SPMP[32]` to `SPMP[63]`, while `SPMP[0]` to `SPMP[31]` are not implemented. In this scenario, the SPMP CSR index mirrors that of the `PMPResources` (when it doesn't exceed 64). A potential drawback is that software must be able to handle scenarios where SPMP doesn't begin at the lowest CSR number. Nevertheless, this is generally acceptable since S-mode software commonly inspects the available SPMP entries by enumerating CSRs at boot time and utilizes the SPMP CSR accordingly based on this probe. Additionally, this design choice becomes even more rational when the Sscsrind extension is implemented and in use. PMP Resouce Sharing extension now supports both mapping methods (and others), and the software only needs to check the usable SPMP entries before using them.

Chapter 4. Summary of Hardware Changes

Item	Changes
CSR for SPMP control	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
CSR for resource sharing	1 new CSR
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 5. Interaction with other proposals

This section discusses how SPMP interacts with other proposals.

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 separate extensions (vspmp and hpmp) in development to support SPMP in guest OS and hypervisors.

Smstateen extension: SPMP adds readable and writable states in S-mode, which can be abused as a covert channel if the OS/hypervisor is not aware of SPMP (thus the states won't be context-switched). It is desired that SPMP occupies a bit in mstateen register of Smstateen extension, which can control supervisor access to SPMP states.