RISC-V S-mode Memory Protection Unit (SMPU)

Dong Du, Xu Lu, Bicheng Yang, Wenhao Li, Yubin Xia, Nick Kossifidis, Joe Xie, Paul Ku, Bill Huffman, Jonathan Behrens, Allen Baum, Robin Zheng, Zeyu Mi, RISC-V TEE Task Group

Version 0.9.0, 11/2021: This document is in the Stable state. Assume anything could still change, but limited change should be expected. For more information see: https://riscv.org/spec-state

Table of Contents

Chapter 1. Motivation

We propose SMPU (RISC-V S-mode Memory Protection Unit) to provide isolation when paged virtual memory system is not available.

RISC-V based processors recently stimulate great interest in the emerging internet of things (IoT). However, as the paged virtual memory system is usually not available 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, it is desirable to enable S-mode OS to limit the physical addresses accessible by U-mode software on a hart.

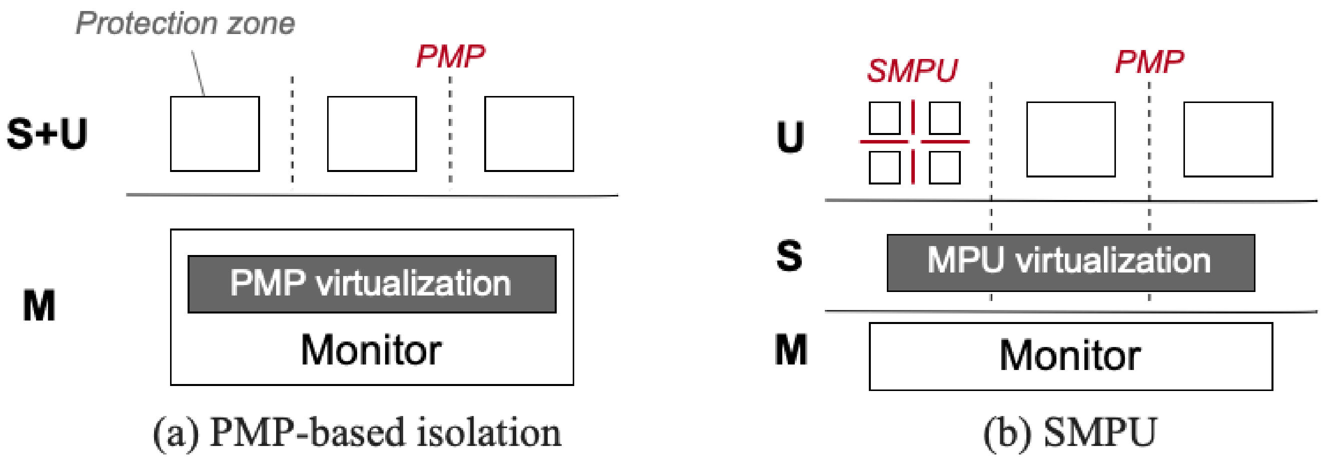

Figure 1. Comparison of PMP and SMPU

Chapter 2. S-mode Memory Protection Unit (SMPU)

An optional RISC-V S-mode Memory Protection Unit (SMPU) 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 SMPU is checked before the PMA checks and PMP checks, the same as paged virtual memory system (abbr. Sv).

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

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

> We use the term, Sv, to represent paged virtual memory system.

2.1. Requirements

1) S mode should be implemented

2.2. S-mode Memory Protection Unit CSRs

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

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

The SMPU configuration registers are packed into CSRs in the same way as PMP does. For RV32, 16 CSRs, smpucfg0-smpucfg15, hold the configurations smpu0cfg-smpu63cfg for the 64 SMPU entries. For RV64, even numbered CSRs (i.e., smpucfg0, smpucfg2, ..., smpucfg14) hold the configurations for the 64 SMPU entries; odd numbered CSRs (e.g., smpucfg1) are illegal. Figure 2 and 3 demonstrate the first 16 entries of SMPU, the layout of rest entries is similar.

31 24	23 16	15 8	7 0	
smpu3cfg	smpu2cfg	smpu1cfg	smpu0cfg	smpucfg0

31 24	23 16	15 8	7	
smpu7cfg	smpu6cfg	smpu5cfg	smpu4cfg	smpucfg1

31 24	23 16	15 8	7	
smpu11cfg	smpu10cfg	smpu9cfg	smpu8cfg	smpucfg2

31 24	23 16	15 8	7	
smpu15cfg	smpu14cfg	smpu13cfg	smpu12cfg	smpucfg3

Figure 2. RV32 SMPU configuration CSR layout

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
smpu7cfg	smpu6cfg	smpu5cfg	smpu4cfg	smpu3cfg	smpu2cfg	smpu1cfg	smpu0cfg	smpucfg0

63 56	55 48	47 40	39 32	31 24	23 16	15 8	7 0	
smpu15cfg	smpu14cfg	smpu13cfg	smpu12cfg	smpu11cfg	smpu10cfg	smpu9cfg	smpu8cfg	smpucfg2

Figure 3. RV64 SMPU configuration CSR layout

The SMPU address registers are CSRs named smpuaddr0-smpuaddr63. Each SMPU address register encodes bits 33-2 of 34-bit physical address for RV32, as shown in Figure 4. For RV64, each SMPU address encodes bits 55–2 of a 56-bit physical address, as shown in Figure 5. Fewer address bits may be implemented for specific reasons, e.g., systems have a smaller physical address space. Implemented address bits must be contiguous and have to go from lower to higher bits.

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

Figure 4. SMPU address register format, RV32

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

Figure 5. SMPU address register format, RV64

The layout of SMPU configuration registers is the same as PMP configuration registers, as is shown in Figure 6. The whole register is WARL.

1. The S bit marks a rule as **S-mode-only** when set and **U-mode-only** when unset. The encoding of `smpucfg.RW=01`, and the encoding `smpucfg.SRWX=1111`, now encode a Shared-Region. The rules and encodings for permission are explained in section 2.4, which resembles the encoding of ePMP (except SMPU doesn't use locked rules).
2. Bit 5 and 6 are reserved for future use.

3. The A bit 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	**Reserved**	A	X	W	R
1	2	2	1	1	1

Figure 6. SMPU configuration register format

The number of SMPU entries: The proposal allows 64 SMPU entries, which can provide 64 isolated regions concurrently. To provide more isolation regions, the software in S-mode (usually an OS) can virtualize more isolated regions and schedule them by switching the values in SMPU entries.

The reset state: On system reset, the A field of smpu[i]cfg should be zero.

2.3. Address Matching

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

2.4. Encoding of Permissions

SMPU has three kinds of rules: **U-mode-only**, **S-mode-only**, and **Shared-Region** rules. The S bit marks a rule as **S-mode-only** when set and **U-mode-only** when unset. The encoding `smpucfg.RW=01` and `smpucfg.SRWX=1111` encode a Shared-Region and `smpucfg.SRWX=1000` is reserved for future standard use.

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 yet without code execution permission on Supervisor mode in order to ensure SMEP.
   - 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 `smpucfg.S` and `smpucfg.X` bits:
   - If `smpucfg.S` is not set the region can be used for sharing data between S-mode and U-mode so is not executable. S-mode has RW access to that region and U-mode has read-only access if `smpucfg.X` is not set, or RW access if `smpucfg.X` is set.
   - If `smpucfg.S` is set the region can be used for sharing code between S-mode and U-mode so is not writeable. Both S-mode and U-mode have execute access on the region and S-mode may also have read access if `smpucfg.X` is set.
   - The encoding `smpucfg.SRWX=1111` can be used for sharing data between S-mode and U mode, where both modes only have read-only access to the region.

The encoding and results are shown in the table:

Bits on *smpucfg* register				Result		
S	R	W	X	S Mode		U Mode
				SUM=0	SUM=1	SUM=0/1
0	0	0	0	Inaccessible region (Access Exception)		
0	0	0	1	Access Exception		Execute-only region
0	0	1	0	Shared data region: Read/write on S mode, read-only on U mode		
0	0	1	1	Shared data region: Read/write for both S and U mode		
0	1	0	0	Access Exception	Read-only region	Read-only region
0	1	0	1	Access Exception	Read-only region	Read/Execute region
0	1	1	0	Access Exception	Read/Write region	Read/Write region
0	1	1	1	Access Exception	Read/Write region	Read/Write/Execute region
1	0	0	0	Reserved		
1	0	0	1	Execute-only region		Access Exception
1	0	1	0	Shared code region: Execute only on both S and U mode		
1	0	1	1	Shared code region: Execute only on U mode, read/execute on S mode		
1	1	0	0	Read-only region		Access Exception
1	1	0	1	Read/Execute region		Access Exception
1	1	1	0	Read/Write region		Access Exception
1	1	1	1	Shared data region: Read only on both S and U mode		

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

2.5. Priority and Matching Logic

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

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

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

type.

2.6. SMPU and Paging (Sv)

The table below shows which mechanism to use. (Assume both Sv and SMPU are implemented.)

satp	Isolation mechanism
satp.mode == Bare	SMPU only
satp.mode != Bare	Sv only

We do not allow both SMPU and Sv permissions active 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 guest OS which may have host SMPU and guest SMPU. (2) Sv can provide sufficient protection.

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

> If Sv is not implemented, or when it is disabled, memory accesses check the SMPU settings synchronously, so no fence is needed.

2.7. Exceptions

Failed accesses generate an exception. SMPU 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 SMPU reuses exception codes of page fault for SMPU fault. This is because page fault is typically delegated to S-mode, and so does SMPU, so we can benefit from reusing page fault. S-mode software(i.e., OS) can distinguish page fault from SMPU fault by checking satp.mode (as mentioned in 2.6, SMPU and Sv will not be activated simultaneously). The **SMPU is proposing to rename page fault to SMPU/Sv 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 SMPU/Sv fault
0	13	Load SMPU/Sv fault
0	15	Store/AMO SMPU/Sv fault

> You can refer to the Table 3.6 in riscv-privileged spec.

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

2.8. Context Switching Optimization

With SMPU, each context switch requires the OS to store 64 address registers and 8 configuration registers

(RV64), which is costly and unnecessary. So the SMPU is proposing an optimization to minimize the overhead caused by context switching.

We add two CSRs called ***smpuswitch0*** and ***smpuswitch1***, which are XLEN-bit read/write registers, formatted as shown in Figure 7. For RV64, only ***smpuswitch0*** is used. Each bit of this register holds on/off status of the corresponding SMPU entry respectively. During context switch, the OS can simply store and restore smpuswitch as part of the context. An SMPU entry is activated only when both corresponding bits in smpuswitch and A field of smpuicfg are set. (i.e., smpuswitch[i] & smpu[i]cfg.A)

Figure 7. SMPU domain switch register format (RV64)

Chapter 3. Summary of Hardware Changes

Item	Changes
CSRs for SMPU address	64 new CSRs
CSRs for SMPU 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 SMPU/Sv fault*** ***Load page fault*** renamed to ***Load SMPU/Sv fault*** ***Store/AMO page fault*** renamed to ***Store/AMO SMPU/Sv fault***

Chapter 4. Interaction with hypervisor extension

To support both SMPU and hypervisor extension (version: 1.0.0-rc), there are some further changes.

4.1. vSMPU extension

This extension describes how SMPU is used in a guest VM.

1. A set of vSMPU CSRs for the VS-mode are required, including 64 vSMPU address registers and 16 configuration registers. When V=1, vSMPU CSR substitutes for the usual SMPU CSR, so instructions that normally read or modify SMPU CSR actually access vSMPU CSR instead. This is consistent with the paging in VS-mode (i.e., vsatp).
2. For HLV, HLVX, and HSV instructions, the hardware should perform vSMPU checking before G-stage address translation (or hgMPU protection when hgatp in BARE mode).
3. The vSMPU checking is performed in the guest physical addresses, before G-stage address translation (or hgMPU protection when hgatp in BARE mode).

4.2. hgMPU extension

This extension describes how SMPU is used for protecting a hypervisor from guests (only enabled when hgatp is set to BARE mode).

1. When hgMPU is enabled, all guest memory accesses will be checked by hgMPU; while hypervisor (in HS mode) and HU mode applications will not be affected.
2. A set of hgMPU CSRs for the HS-mode are required, including 64 hgMPUaddr address registers and 16 hgMPUcfg configuration registers. When V=1, and hgatp.MODE=Bare, hgMPU is used to provide isolation between hypervisor and guest VMs.
3. XLEN-bit read/write hgmpuswitch0 and hgmpuswitch1 CSRs are also provided in hgMPU, which are identical to smpuswitch0 and smpuswitch1 shown in Figure 7. Only hgmpuswitch0 is used for RV64. During context switch, the hypervisor can simply store and restore hgmpuswitch (we use hgmpuswitch to represent either hgmpuswitch0 or hgmpuswitch1) as part of the context. An hgMPU entry is activated only when both corresponding bits in hgmpuswitch and A field of hgmpuicfg are set. (i.e., hgmpuswitch[i] & hgmpuicfg.A)
4. The hgMPU checking is performed after the guest address translation (or vSMPU checking), before PMP checking.

As hgMPU does not apply on hypervisor, the encodings of configuration registers are simplified as the following table.

The encodings of hgmpucfg are shown in the table:

Bits on *hgmpucfg* register				**Result**
S	R	W	X	V Mode (VS + VU)
0	0	0	0	Inaccessible region (Access Exception)
0	0	0	1	Execute-only region
0	1	0	0	Read-only region
0	1	0	1	Read/Execute region

Bits on *hgmpucfg* register				Result
0	1	1	0	Read/Write region
0	1	1	1	Read/Write/Execute region
Others				Reserved

Chapter 5. Interaction with other proposals

This section discusses how SMPU interacts with other proposals.

RISC-V PMP enhancements (version: 0.9.4): SMPU is compatible with ePMP proposal, and uses almost the same encoding as ePMP.

J-extension pointer masking proposal (version: v0.1-draft): When both PM and SMPU are used, SMPU checking should be performed using the actual addresses generated by PM (pointer masking).