Nuclei™ TEE Architecture

Copyright Notice

Contact Information

Should you have any problems with the information contained herein, please contact Nuclei System Technology by email support@nucleisys.com, or visit “Nuclei User Center” website http://user.nucleisys.com for supports or online discussion.

Revision History

Rev.	Revision Date	Revised Content
1.0.0	2019/11/12	1. Initial Release.
1.0.1	2021/02/01	1. Add sdcause in 2.2. 2. modify some typos related with user mode and supervisor mode

Table of Contents

List of Tables

List of Figures

1.TEE Introduction

Nuclei provides the TEE (Trusted Execution Environment) to achieve better isolation for implementing supervisor-level interrupt/exception handling and sPMP.

With TEE, it is flexible to isolate machine-level interrupt/exception from lower privileged-level interrupt/exception.

When the TEE is configured, and the outer execution environment has delegated specified interrupts and exceptions to supervisor-level, then hardware can transfer control directly to a supervisor-level trap handler without invoking the outer execution environment. Since then, users have more flexibility to define the behavior of hardware when an interrupt or exception is taken. Users can delegate the specified exceptions through the TEE. As for interrupts, with the TEE, users can delegate the designated interrupts only when ECLIC is configured, which is described in Nuclei ISA Spec. The NMI (Non-maskable-interrupt) cannot be trapped to the supervisor-mode or user-mode for any configuration.

Moreover, with TEE, it enables S-mode OS to limit the physical addresses accessible by U-mode software.

When the TEE is configured, sPMP entries can be set to achieve better isolation between OS(running on S-mode) from user software(running on U-mode), and to achieve better scalability of PMP-based TEE/enclave with smaller TCB (Trusted Computing Base).

2.TEE Related CSRs

2.1. Added CSRs List For The TEE

Based on the basic CSRs, the TEE has added some CSRs as shown in Table 2 -1. These include the RISC-V standard CSRs for the S-mode, and some Nuclei customized CSRs.

Table 2-1 Added CSRs for the TEE

CSR Type	Number	Privilege	Name	Description
RISC-V Standard CSR	0x100	SRW	sstatus	Supervisor status register
	0x104	SRW	sie	Supervisor interrupt-enable register
	0x105	SRW	stvec	Supervisor trap handler base address
	0x106	SRW	scounteren	Supervisor counter enable
	0x107	SRW	stvt	Supervisor Trap-handler vector table base address

	0x140	SRW	sscratch	Scratch register for supervisor trap handlers
	0x141	SRW	sepc	Supervisor exception program counter
	0x142	SRW	scause	Supervisor trap cause
	0x143	SRW	stval	Supervisor bad address or instruction
	0x144	SRW	sip	Supervisor interrupt pending
	0x145	SRW	snxti	Supervisor interrupt handler address and enable modifier
	0x146	SRW	sintstatus	Supervisor current interrupt levels
	0x148	SRW	sscratchcsw	Scratch swap register for multiple privilege modes
	0x149	SRW	sscratchcswl	Scratch swap register for supervisor interrupt levels
	0x180	SRW	satp	Supervisor address translation and protection
	0x302	MRW	medeleg	Machine exception delegation register
	0x303	MRW	mideleg	Machine interrupt delegation register
Nuclei Customized CSRs	0x1A0	SRW	spmpcfg0	Supervisor physical memory protection configuration
	0x1A1	SRW	spmpcfg1	Supervisor physical memory protection configuration
	0x1A2	SRW	spmpcfg2	Supervisor physical memory protection configuration
	0x1A3	SRW	spmpcfg3	Supervisor physical memory protection configuration
	0x1B0	SRW	spmpaddr0	Supervisor physical memory protection address register
	0x1B1	SRW	spmpaddr1	Supervisor physical memory protection address register
	…	SRW	…	…
	(0x1B0+n)	SRW	spmpaddrn	Supervisor physical memory protection address register
	0x947	SRW	jalsnxti	Jumping to next supervisor interrupt handler address and interrupt-enable register
	0948	SRW	stvt2	ECLIC non-vectored supervisor interrupt handler address register
	0x949	SRW	pushscause	Push scause to stack
	0x94a	SRW	pushsepc	Push sepc to stack

2.2. RISC-V Standard CSR For TEE

2.2.1.sstatus

The *sstatus* register is a 32-bit read/write register formatted as shown in Figure 2-1. The
sstatus register keeps track of and controls the hart's current operating state.

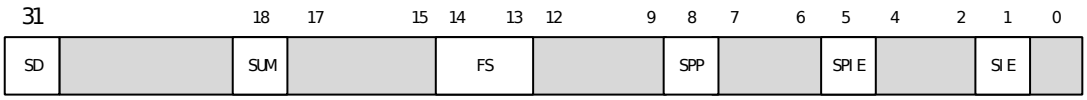

Figure 2-1 sstatus register

The supervisor interrupt-enable bit SIE indicates supervisor-level interrupts are disabled when it is clear. The value of SIE is preserved in SPIE when a supervisor-level trap is taken, and the value of SIE is set to zero to provide atomicity for the supervisor-level trap handler.

The SUM (permit Supervisor User Memory access) bit modifes the privilege with which S-mode
loads and stores access user mode memory, see section 6.2.5. for more details about SUM bit .

The SRET instruction is used to return from traps in S-mode, an instruction unique to S-mode. SRET sets PC to sepc, restores interrupt-enable setting by copying SPIE to SIE, and sets SPIE bit.

When the TEE is configured, the *mstatus* register mirrors the SIE and SPIE fields in bit 1 and bit 5, the same location with *sstatus* as shown in Figure 2 -2.

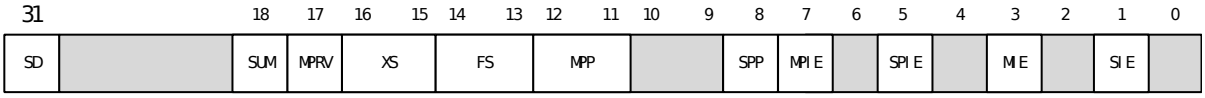

Figure 2-2 mstatus register

2.2.2.sie

The *sie* register has **NO** effect when interrupt handling mode is ECLIC, and return data are all zeros while reading the register.

2.2.3.stvec

The s*tvec* register is a 32-bit read/write register formatted as shown in Figure 2 -3. It functions in an analogous way to the *mtvec* register defined in M-mode.

The *stvec* register holds S-mode exception and interrupt configuration, consisting of exception and CLIC non-vector base address (BASE).

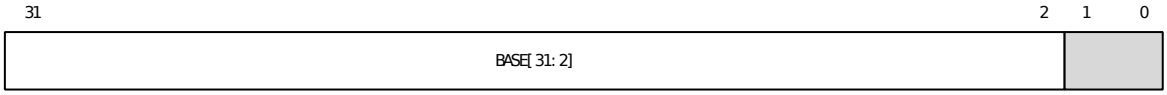

Figure 2-3 stvec register

2.2.4.scounteren

The counter-enable register *scounteren* controls the availability of the hardware performance monitoring counters to U-mode.

When the CY, TM or IR bit in the *scounteren* register is clear, attempts to read the cycle,
time or instret register while executing in U-mode will cause an illegal instruction exception. When one of these bits is set, access to the corresponding register is permitted.

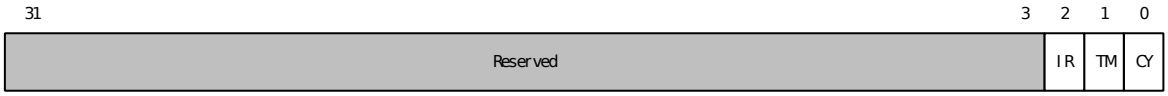

Figure 2-4 scounter register

2.2.5.stvt

The s*tvt* register is a 32-bit read/write register formatted as shown in Figure 2 -5. It functions in an analogous way to the *mtvt* register defined in M-mode.

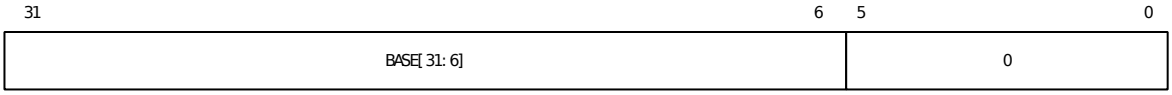

Figure 2-5 stvt register

The *stvt* register holds the base address of ECLIC S-mode vector interrupts, and the base address is aligned at least 64-byte boundary. In order to improve the performance and reduce the gate count, the alignment of the base address in *stvt* is determined by the actual number of interrupts, which is shown in Table 2 -2.

Table 2-2 Alignment of stvt base address

interrupt number	alignment
0 to 16	64-byte
17 to 32	128-byte
33 to 64	256-byte
65 to 128	512-byte
129 to 256	1KB
257 to 512	2KB
513 to 1024	4KB

2.2.6.sscratch

The *sscratch* register is a 32-bit read/write register dedicated for use by supervisor mode. Typically, it is used to hold a pointer to a user mode hart-local context space and swapped with a supervisor register upon entry to a S-mode trap handler.

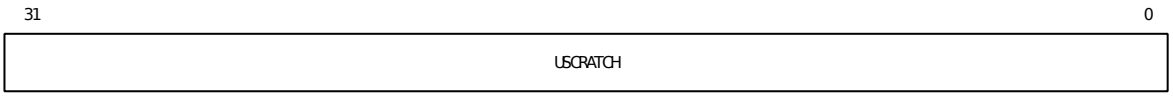

Figure 2-6 sscratch register

2.2.7.sepc

The *sepc* register is a 32-bit read/write register formatted as shown in Figure 2 - 7. It functions in an analogous way to the *mepc* register defined in M-mode.

When an exception or interrupt is taken into S-mode, *sepc* is written with the virtual address of the instruction that encountered the exception or interrupt. Otherwise, *sepc* is never written by the implementation, though it may be explicitly written by software.

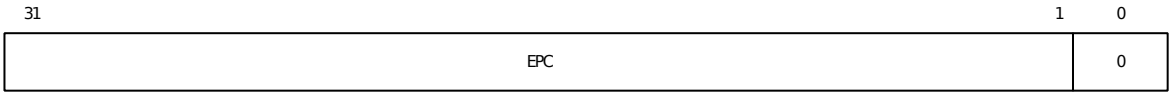

Figure 2-7 sepc register

2.2.8.scause

The *scause* register is a 32-bit read/write register formatted as shown in Figure 2 -8. It functions in an analogous way to the *mcause* register defined in M-mode.

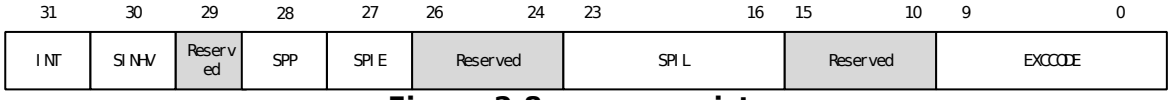

Figure 2-8 scause register

Table 2-3 scause register description

Field Name	Bits	Description
INT	31	The bit is set if the trap was caused by an interrupt.
SINHV	30	The bit indicates that the interrupt is reading the interrupt vector table.
Reserved	29	0
SPP	28	The bit indicates privilege mode before S-mode exception or interrupt is taken. It mirrors *sstatus*.spp
SPIE	27	The bit holds the SIE value before S-mode exception or interrupt is taken. It mirrors *sstatus*.spie
Reserved	26 : 24	0
SPIL	23 : 16	The bits hold the interrupt level before S-mode interrupt is taken.
Reserved	15 : 10	0
EXCCODE	9 : 0	The bits hold the code of last exception or interrupt

2.2.9.sdcause

Since there might be some exceptions share the same scause.EXCCODE value. To further record the differences, Nuclei processor core customized CSR sdcause register to record the detailed information about the exception.

Table 2-4 sdcause register description

Field	Bit	Description
Reserved	31 : 2	Reserved 0
sdcause	2:0	Further record the detailed information about the exception. When scause.EXCCODE = 1 (Instruction access fault) ■ 0: Reserved ■ 1: PMP permission violation ■ 2: Bus error ■ 3-7: Reserved When scause.EXCCODE = 5 (Load access fault) ■ 0: Reserved ■ 1: PMP permission violation ■ 2: Bus error ■ 3: NICE extended long pipeline instruction return error. Note: although this error ideally is nothing to do with the Load access fault, but they just shared the same scause.EXCCODE to simplify the hardware implementation ■ 4-7: Reserved When scause.EXCCODE = 7 (Store/AMO access fault) ■ 0: Reserved ■ 1: PMP permission violation ■ 2: Bus error ■ 3-7: Reserved When scause.EXCCODE = 12 (Instruction page fault) ■ 0-4: Reserved ■ 5: Page fault ■ 6. SPMP permission violation ■ 7: Reserved When scause.EXCCODE = 13 (Load page fault) ■ 0-4: Reserved ■ 5: Page fault ■ 6. SPMP permission violation ■ 7: Reserved When scause.EXCCODE = 15 (Store/AMO page fault) ■ 0-4: Reserved ■ 5: Page fault ■ 6. SPMP permission violation ■ 7: Reserved

2.2.10. stval

The *stval* register is a 32-bit read-write register formatted as shown in Figure 2 - 9. When a trap is taken into S-mode, *stval* is written with exception-specific information to assist software in handling the trap. Otherwise, *stval* is never written by the implementation, though it may be explicitly written by software.

When a hardware breakpoint is triggered, or an instruction-fetch, load, or store address-misaligned, access exception occurs, *stval* is written with the faulting effective address.

On an illegal instruction trap, *stval* is written with the faulting instruction code.

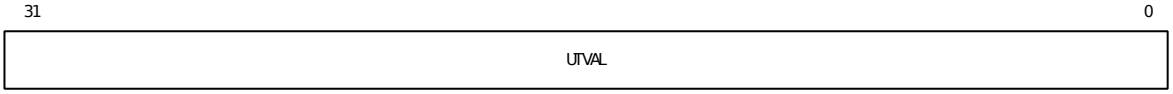

Figure 2-9 stval register

2.2.11. sip

The *sip* register has **NO** effect when interrupt handling mode is ECLIC, and return data are all zeros while reading the register.

2.2.12. snxti

The *snxti* register is a 32-bit read/write register formatted as shown in Figure 2 - 10. It functions in an analogous way to the *mnxti* register defined in M-mode.

The *snxti* CSR can be used by software to service the next S-mode horizontal or lower level interrupt when it has a higher level than the saved interrupt context (held in *scause*.*spil*), without incurring the full cost of an interrupt pipeline flush and context save/restore.

Note: If the next interrupt is an M-mode one while the execution privilege mode is S-mode, the processor will take the next interrupt directly in a nested way. In other hand, if the processor is running in M-mode, then a S-mode interrupt will not be taken.

The *snxti* CSR is designed to be accessed using CSRRSI/CSRRCI instructions, where the value read is a pointer to an entry in the S-mode interrupt handler table and the write back updates the S-mode interrupt-enable status.

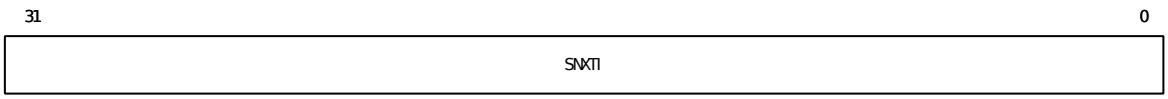

Figure 2-10 snxti register

2.2.13. sintstatus

The *sintstatus* register is a 32-bit read-only register formatted as shown in Figure 2 -11. It functions in an analogous way to the *mintstatus* register defined in M-mode.

The *sintstatus* register holds the active interrupt level for S-mode. The SIL field is read-only.

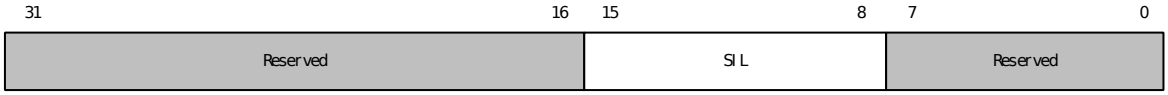

Figure 2-11 sintstatus register

When the TEE is configured, the *mintstatus* register mirrors the SIL field in bit 8-15, the same location with *sintstatus* as shown in Figure 2 -12

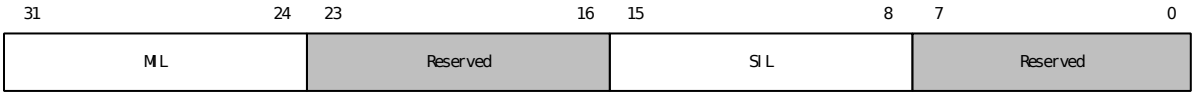

Figure 2-12 mintstatus register

2.2.14. sscratchcsw

The *sscratchcsw* register is a 32-bit read/write register formatted as shown in Figure 2 -13. It functions in an analogous way to the *mscratchcsw* register defined in M-mode.

Figure 2-13 sscratchcsw register

To accelerate interrupt handling with multiple privilege modes, sscratchcsw is defined for supervisor mode to support conditional swapping of the sscratch register when transitioning between supervisor mode and user mode.

```
csrrw rd, sscratchcsw, rs1
  // Pseudocode operation.
  if (scause.spp != S-mode) then {
      t = rs1; rd = sscratch; sscratch = t;
  } else {
      rd = rs1; // sscratch unchanged.
  }
  // Usual use: csrrw sp, sscratchcswl, sp
```

2.2.15. sscratchcswl

The *sscratchcswl* register is a 32-bit read/write register formatted as shown in Figure 2 -14. It functions in an analogous way to the *mscratchcswl* register defined in M-mode.

Figure 2-14 sscratchcswl register

Within U-mode, *sscratchcswl* is useful to separate interrupt handler tasks from application tasks to enhance robustness, reduce space usage, and aid in system debugging. Interrupt handler tasks only have non-zero interrupt levels, while application tasks have an interrupt level of zero.

The *sscratchcswl* CSR is added to support faster swapping of the stack pointer between S-mode interrupt and non-interrupt code running in the same privilege mode.

```
csrrw rd, sscratchcswl, rs1
  // Pseudocode operation.
  if ( (scause.spil==0) != (sintstatus.sil==0) ) then {
      t = rs1; rd = sscratch; sscratch = t;
  } else {
      rd = rs1; // sscratch unchanged.
  }
  // Usual use: csrrw sp, sscratchcswl, sp
```

2.2.16. satp

The *s*atp CSR provides the Supervisor Address Translation and Protection.

2.2.17. medeleg

The *medeleg* register is a 32-bit read/write register formatted as shown in Figure 2 -15. To increase performance, the *medeleg* register contains individual read/write bits to indicate that certain exceptions should be processed directly by a lower privilege level.

By default, all exceptions at any privilege level are handled in machine mode. When N-Extension is configured, setting a bit in *medeleg* will delegate the corresponding exception in S/U-mode to the S-mode exception handler.

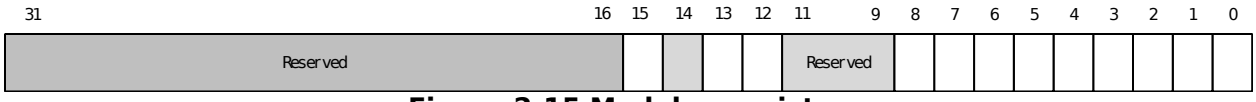

Figure 2-15 Medeleg register

Table 2-5 Medeleg register description

Field Name	Bits	Description

Reserved	31 : 16	0
	15	Delegate Store/AMO Page Fault Exception
Reserved	14	0
	13	Delegate Load Page Fault Exception
	12	Delegate Instruction Page Fault Exception
	11	Not used, tie to 0
Reserved	10	0
	9	Not used, tie to 0
	8	Delegate Environment Call from U-Mode
	7	Delegate Store/AMO Access Fault Exception
	6	Delegate Store/AMO Address Misaligned Exception
	5	Delegate Load Access Fault Exception
	4	Delegate Load Access Misaligned Exception
	3	Delegate Breakpoint Exception
	2	Delegate Illegal Instruction Exceptio
	1	Delegate Instruction Access Fault Exception
	0	Delegate Instruction Access Misaligned Exception

2.2.18. mideleg

The *mideleg* register has **NO** effect when interrupt handing mode is ECLIC, and return data are all zeros while reading the register.

2.3. Nuclei Customized CSR For TEE

2.3.1.spmpcfg<x>

The *spmpcfg<x>* register is a 32-bit read/write register formatted as shown in Figure 6 -23 spmpcfg<x> layout. It functions in an analogous way to the *pmpcfg<x>* register defined in M-mode except that: a new bit called U bit is defined in *spmpcfg<x>* to support SMAP(Supervisor Memory Access Prevention) and SMEP(Supervisor Memory Execution Prevention).

2.3.2.spmpaddr<x>

The *spmpaddr<x>* register is a 32-bit read/write register formatted as shown in Figure 6 -24. It functions in an analogous way to the *pmpaddr<x>* register defined in M-mode.

2.3.3.jalsnxti

The *jalsnxti* register is a 32-bit read/write register formatted as shown in Figure 2 -16. It functions in an analogous way to the *jalmnxti* register defined in M-mode.

The *jalsnxti* register is designed to speed up the S-mode interrupt handling comparing to using *snxti*. In addition to enabling the interrupt, accessing this CSR by ‘csrrw ra, jalsnxti, ra’ will directly jump to the next S-mode horizontal or lower

level interrupt entry address for the same privilege mode, at the same time, save the return address which is the pc of the executing instruction, to the ra register. For more details please refer to chapter 3.4.3..

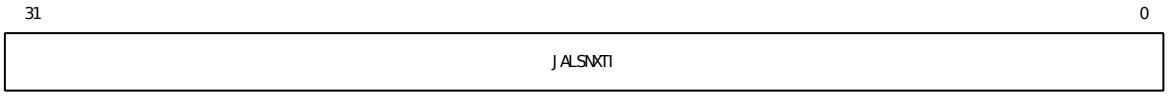

Figure 2-16 jalsnxti register

2.3.4.stvt2

The *stvt2* register is a 32-bit read/write register formatted as shown in Figure 2 - 17. It functions in an analogous way to the *mtvt2* register defined in M-mode.

The *stvt2* register holds S-mode ECLIC configuration, consisting of ECLIC non-vector base address(BASE) and ECLIC enable(STVT2EN). The ECLIC non-vector base address is aligned on a 4-byte boundary

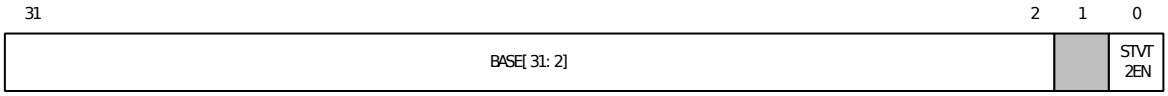

Figure 2-17 stvt2 register

Table 2-6 stvt2 register description

Field Name	Bits	Description
BASE	31:2	The base address of ECLIC non-vector interrupt in U-mode
Reserved	1	0
STVT2EN	0	Setting this bit will enable ECLIC mode , that means non-vector interrupt base address in S-mode is specified by stvt2 instead of stvec

2.3.5.pushscause

The *pushscause* register is a 32-bit read/write register formatted as shown in Figure 2 -18. It functions in an analogous way to the *pushmcause* register defined in M-mode.

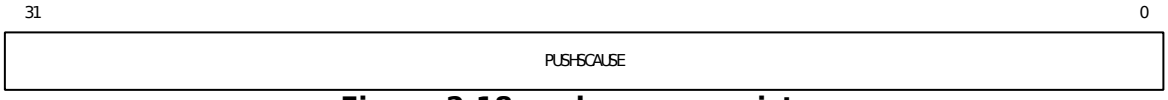

Figure 2-18 pushscause register

The *pushscause* register is a function CSR and it can't be read or written as a normal CSR. Any CSR instruction except for csrrwi accessing this register will raise an illegal instruction exception.

The *pushscause* register is designed to speed up the context saving of S-mode

interrupt. Using csrrwi instruction, the data of *scause* will be pushed to memory address = sp + imm*4.

```
        csrrwi x0, pushscause ,2 // push scause to memory address = sp + 8
```

2.3.6.pushsepc

The *pushsepc* register is a 32-bit read/write register formatted as shown in Figure 2 -19. It functions in an analogous way to the *pushmepc* register defined in M-mode.

Figure 2-19 pushsepc register

The *pushsepc* register is a function CSR and it can't be read or written as a normal CSR. Any CSR instruction except for csrrwi accessing this register will raise an illegal instruction exception.

The *pushsepc* register is designed to speed up the context saving of S-mode interrupt. Using csrrwi instruction, the data of *sepc will be pushed* to memory address = sp + imm*4.

```
        csrrwi x0, pushsepc ,2 // push sepc to memory address = sp + 8
```

3.TEE Interrupt Operation

This chapter describes the behavior of TEE interrupts, especially for supervisor-level interrupts. A supervisor-level interrupt is taken only when the current mode is supervisor mode and the s*tatus*.SIE field is set or when the current mode is user mode. Several main changes in the TEE will be mentioned in the following sections.

3.1. ECLIC Memory Map

The ECLIC memory map has been changed to support the TEE. The ECLIC memory map can support up to 1,024 total interrupt inputs, and an M-mode ECLIC Region and a S-mode ECLIC Region. Table 3 -7 describes the detailed memory map scheme.

Table 3-7 ECLIC Modified Memory Map

Offset	R/W	Name	Width (bits)	Description
0x0000	RW	cliccfg	8	
0x0004	R	clicinfo	32	
0x0008	RW	mintthresh	32	
0x1000+4*i	RW	clicintip[i]	8	M-mode ECLIC Region
0x1001+4*i	RW	clicintie[i]	8	
0x1002+4*i	RW	clicintattr[i]	8	
0x1003+4*i	RW	clicintctl[i]	8	
0x2008	RW	sintthresh	32	
0x3000+4*i	RW	clicintip[i]	8	S-mode ECLIC Region
0x3001+4*i	RW	clicintie[i]	8	
0x3002+4*i	RW	clicintattr[i]	8	
0x3003+4*i	RW	clicintctl[i]	8	
Note: ■ The above "i" indicates the interrupt ID, an interrupt i has its own corresponding clicintip[i], clicintie[i], clicintattr[i], and clicintctl[i] registers. ■ Aligned byte, half-word, and word accesses to ECLIC registers are supported. ■ The above "R" means read-only, and any write to this read-only register will be ignored without bus error. ■ If an input i is not present in the hardware, the corresponding clicintip[i], clicintie[i], clicintctl[i] memory locations appear hardwired to zero.				

3.1.1.M-mode ECLIC Region

This region is designed to support M-mode access. If an input i is not present in the hardware, the corresponding *clicintip[i]*, *clicintie[i]*, *clicintctl[i]* memory locations appear hardwired to zero.

3.1.2.S-mode ECLIC Region

Supervisor-mode ECLIC regions only expose interrupts that have been configured to be supervisor-accessible via the M-mode CLIC region. System software must configure PMP and sPMP permissions to make sure this region can only be accessed from appropriate Supervisor-mode codes.

Any interrupt *i* that is not accessible to S-mode appears as hard-wired zeros in *clicintip[i]*, *clicintie[i]*, and *clicintctl[i]*.

If *clicintattr [i]* is set to S-mode (bit 7 is clear, and bit 6 is set), then interrupt *i* is visible in the S-mode region except that only the low 6 bits of *clicintattr [i]* can be written via the S-mode memory region.

3.1.3.ECLIC Modified Memory Mapped Registers

Following only describes the ECLIC modified memory mapped registers.

3.1.3.1 cliccfg

This *cliccfg* register is a global configuration register. Table 3 -8 describes the bit assignments of this register.

Table 3-8 cliccfg bit assignments

Field	Bits	R/W	Reset Value	Description
Reserved	7	R	N/A	Reserved, ties to 0.
nmbits	6:5	R	N/A	nmbits ties to 1, indicates supervisor-level interrupt supporting.
nlbits	4:1	RW	0	nlbits specifies interrupt level and priority.
Reserved	0	R	N/A	Reserved, ties to 1.

3.1.3.2 mintthresh

The *mintthresh* register holds the current threshold level for each privilege mode (i.e., mth, sth). Table 3 -9 describes the bit assignments of this register.

Table 3-9 mintthresh bit assignments

Field	Bits	R/W	Reset Value	Description
mth	31:24	RW	0	Interrupt-level threshold for M-mode.
Reserved	23:16	R	N/A	Reserved, ties to 0.
sth	15:8	RW	0	Interrupt-level threshold for S-mode.
Reserved	7:0	R	N/A	Reserved, ties to 0.

3.1.3.3 sintthresh

The *sintthresh* register holds the current threshold level for Supervisor mode. Table 3 -10 describes the bit assignments of this register.

Table 3-10 sintthresh bit assignments

Field	Bits	R/W	Reset Value	Description
Reserved	31:24	R	N/A	Reserved, ties to 0.

Reserved	23:16	R	N/A	Reserved, ties to 0.
sth	15:8	RW	0	Interrupt-level threshold for S-mode.
Reserved	7:0	R	N/A	Reserved, ties to 0.

Note: here sth is a mirror to mintthresh.sth and will be updated synchronously.

3.1.3.4 clicintattr[i]

This register specifies various attributes for each interrupt. Table 3 -11 describes the bit assignments of this register.

Table 3-11 clicintattr [i] bit assignments

Field	Bits	R/W	Reset Value	Description
mode	7:6	M-mode: RW S-mode: R	3	Specifies in which privilege mode the interrupt should be taken: 3: Machine Mode 1: Supervisor Mode Note: M-mode can Read/Write this field, but S-mode can only Read this field.
Reserved	5:3	R	N/A	Reserved, ties to 0.
trig	2:1	RW	0	Specifies the trigger type and edge polarity for each interrupt input.
shv	0	RW	0	Specifies hardware vectoring or non-vectoring mode.

3.2. ECLIC CSRs

Supervisor-mode related CSRs are added to support supervisor-level interrupt, please refer to TEE Related CSRs in Chapter 2 for more details.

3.3. ECLIC Interrupt Arbitration

As mentioned above, the *clicintattr[i]*.mode field is implemented to specify in which privilege mode the interrupt should be taken. In order to support the TEE, the arbitration logic needs to take the *clicintattr[i]*.mode into the consideration. Hence, the winner of the arbitration is determined by interrupts' mode, level and priority. In the modified arbitration logic, privilege mode priors to level and priority, and level priors to priority. Figure 3 -20 shows the modified arbitration scheme. For each interrupt *i*, its privilege mode, level and priority are combined to represent an unsigned integer number. The interrupt that has the greatest number wins the arbitration.

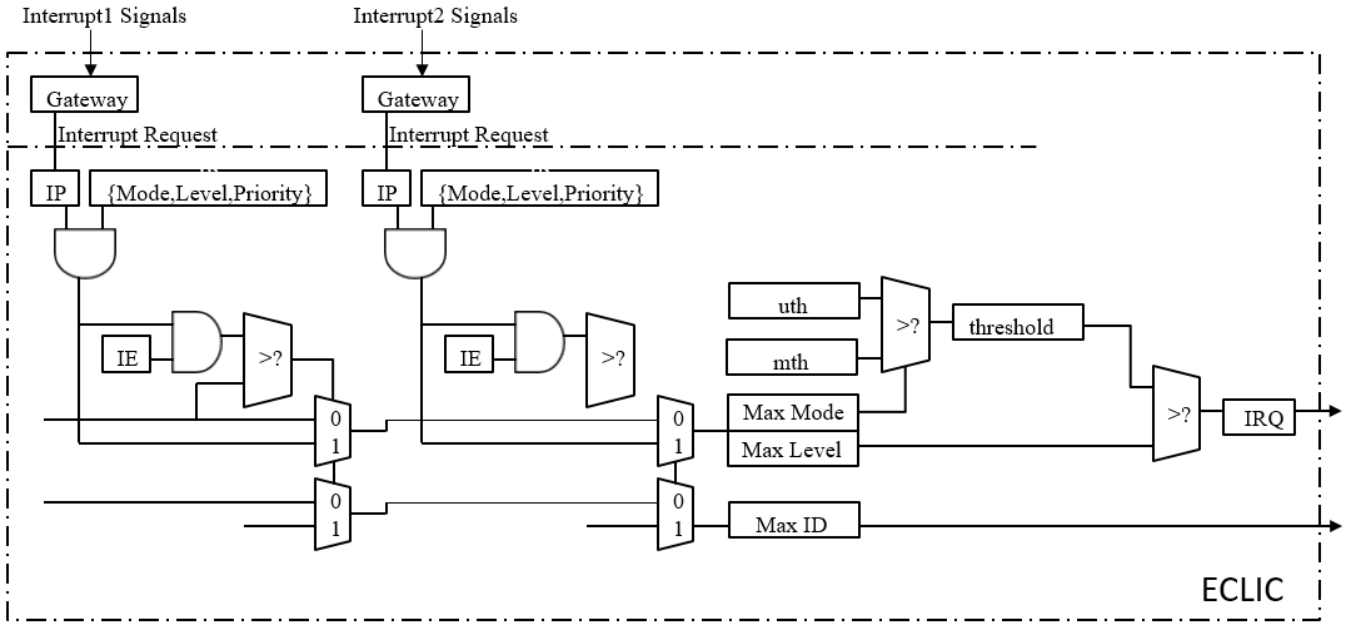

Figure 3-20 Modified arbitration scheme

3.4. Supervisor-level Interrupt Flow

This section describes the overall behavior when a supervisor-level interrupt is taken.

3.4.1.Enter A Supervisor-level Interrupt Handler

If a supervisor-level interrupt wins the arbitration and is taken(it can only happen in supervisor/user mode), the following listed hardware behaviors will happen at the same time:

- Stop executing the current program, and jump to a new PC, then execute.
- Update the following listed CSRs at the same time:
  sepc
  sstatus
  scause
  sintstatus
- The privilege mode changes to supervisor mode.
- Figure 3 -21 shows the general flow of entering a supervisor-level interrupt handler.

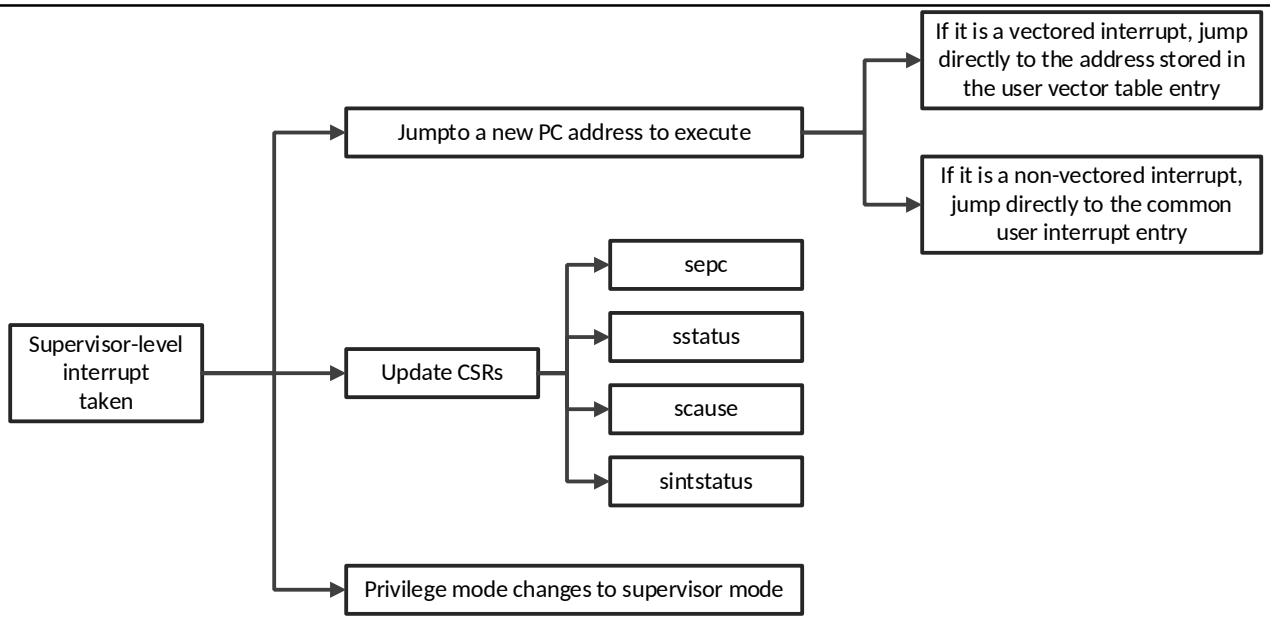

Figure 3-21 General flow of entering an supervisor-level interrupt handler

3.4.1.1 Jump To A New PC To Execute

Each interrupt can be configured to vectored or non-vectored (through *clicintattr[i]*.shv).

- If the interrput is vectored, the core will jump directly to the address stored in the supervisor vector table entry, which is specified by *stvt*.
- If the interrput is non-vectored, the core will jump directly to the common supervisor interrupt entry, which is specified by *stvt2*.

3.4.1.2 Update CSR Register sepc

When a supervisor-level interrupt is taken, the sepc will be updated to the interrupted PC. In this way, once returning from this supervisor-level interrupt handler, the core can restore the PC from *spec*.

3.4.1.3 Update CSR Register scause

The *scause* will be updated as follows:

- The *scause*.EXCCODE field will be updated to indicate the current interrupt ID.
- The *scause*.SPIL field will be updated to indicate the interrupted interrupt level(ie, *sintstatus*.SIL). In this way, once returning from supervisor-level interrupt handler, *sintstatus*.SIL can restore is interrupted value from *scause*.SPIL.
- If the interrupt is vectored, the scause.SINHV field will be updated to indicate

its hardware vectoring state.

3.4.1.4 Update CSR Register *sstatus*

The *sstatus* will be updated as follows:

- The *sstatus*.SPIE field will be updated to the value of *sstatus*.SIE field, while *sstatus*.SIE field will be updated to zero.
- Note: The *scause*.SPIE field mirrors the *sstatus*.SPIE field, and is aliased into *scause* to reduce context save/restore code.

3.4.1.5 Update CSR Register sintstatus

The *sintstatus* will be updated as follows:

- The *sintstatus*.SIL field will be updated to hold the active interrupt level.

3.4.1.6 Privilege Mode Changes to Supervisors Mode

A supervisor-level interrupt can be taken from supervisor/user mode, after being taken the privilege mode will change to supervisor mode.

3.4.2.Return From A Supervisor-level Interrupt Handler

The regular sret instructions are used to return from a supervisor-level interrupt handler. After the execution of sret, the following listed hardware behaviors will happen at the same time.

- Stop executing the current program, and jump to a new PC stored in sepc to execute.
- Update the following listed CSRs at the same time:
  sstatus
  scause
  sintstatus
- The privilege mode changes to the previous interrupted mode.
- Figure 3 -22 shows the general procedure.

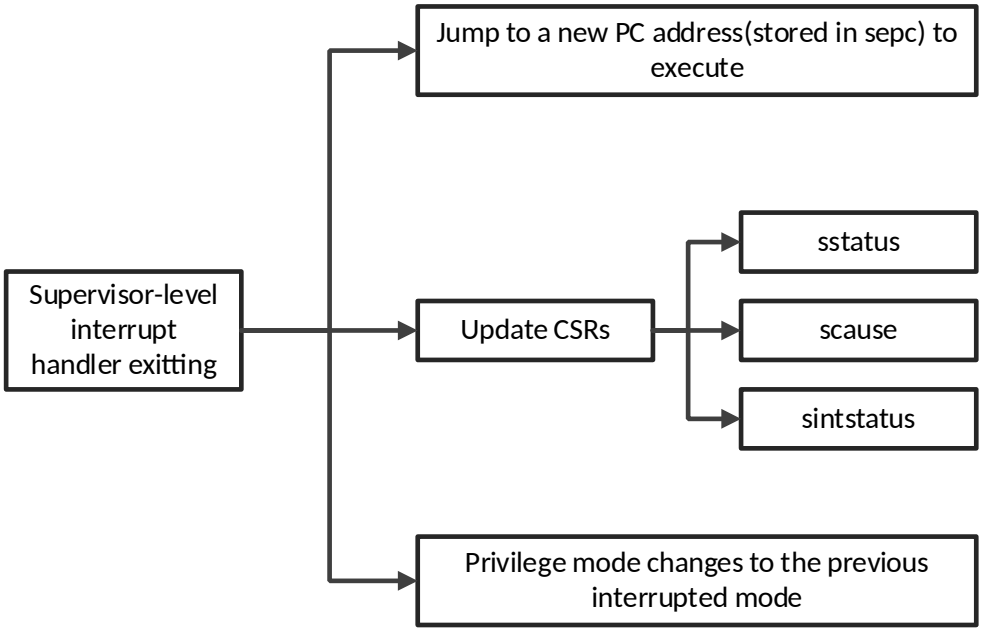

Figure 3-22 General flow of returning from an supervisor-level interrupt handler

3.4.2.1 Jump To A New PC To Execute

The core will jump to a new address stored in spec to execute, after executing the sret instruction to return from a supervisor-level interrupt handler.

3.4.2.2 Update CSR Register *sstatus*

After the execution of sret, the CSR register *sstatus* will be updated as follows.

- The *sstatus*.SIE field will be updated to the value of *sstatus*.SPIE field.
- The *sstatus*.SPIE field will be set to 1.

3.4.2.3 Update CSR Register scause

After the execution of sret, the CSR register *scause* will be updated as follows.

- Note: The *scause*.SPIE field mirrors the *sstatus*.SPIE field, and is aliased into scause to reduce context save/restore code.

3.4.2.4 Update CSR Register sintstatus

After the execution of sret, the CSR register *sintstatus* will be updated as follows.

- The *sintstatus*.SIL field will be updated to the value of the *scause*.SPIL filed.

3.4.2.5 Privilege Mode Changes to the previous interrupted mode

The privilege mode will change to the previous interrupted mode (which is stored in sstatus.SPP) when returning from this interrupt handler.

3.4.3.Supervisor-level Non-vector Mode

Supervisor-level non-vector mode interrupt flow is the same as machine-level non-vector mode flow, except:

- The CSRs that should be saved and restored during interrupt processing are different.
  During supervisor-level interrupt processing, *sepc* and *scause* should be saved and restored.
- The instruction used for tail-chaining is different.
  For supervisor-level non-vector mode, the instruction “csrrw ra, jalsnxti, ra” is used to realize supervisor-level interrupt tail-chaining.

3.4.4.Supervisor-level Vector Mode

Supervisor-level vector mode interrupt flow is the same as machine-level vector mode flow, except:

- The CSRs that should be saved and restored during interrupt processing are different.
  During supervisor-level interrupt processing, *sepc* and *scause* should be saved and restored.
- The "interrupt attribute" for C is different.
  For a supervisor-level vector mode handler, the "interrupt attribute" for C has the following syntax:

```
void __attribute__ ((interrupt(“supervisor”)))
  foo (void)
  {
    extern volatile int INTERRUPT_FLAG;
    INTERRUPT_FLAG = 0;
    extern volatile int COUNTER;
  #ifdef __riscv_atomic
    __atomic_fetch_add (&COUNTER, 1, __ATOMIC_RELAXED);
  #else
    COUNTER++;
  #endif
  }
```

3.5. Nesting Between Privilege Modes

A machine-level interrupt can pre-empt a supervisor-level interrupt which has already been taken in supervisor mode, while a supervisor-level interrupt can never pre-empt a machine-level interrupt. Besides, a supervisor-level interrupt occurring in machine mode can not be taken.

4.TEE Exception Operation

This chapter describes the behavior of TEE exception, especially for S-mode exception. The S-mode exception is taken only when the current mode is supervisor mode and the corresponding bit in medeleg is set or the current mode is user mode. On the other hand, the medeleg register will be ignored when the processor is in machine mode, which means any exception happens in machine mode will never be delegated.

4.1. TEE Exception Mask

For a processor that supports the TEE, unlike interrupts, exceptions can't be masked in any privilege mode at any time. Once there is an exception, the processor will take it in either machine or supervisor mode.

4.2. TEE Exception Priority

The priority of exceptions taken in same privilege mode is defined by exception code, that is, the exception with smaller code has a higher priority. If a S-mode exception and an M-mode exception occur at the same time in supervisor mode, the M-mode exception has a higher priority.

4.3. S-mode Exception Taken

When entering a S-mode exception, the hardware behavior can be briefly described as follows.

- The PC is changed to the address defined in the *stvec* register.
- *scause* is updated to reflect the type of exception.
- *sepc* is updated to save the pc address where the exception happens.
- *stval* is updated to record the memory access address or instruction code.
- *sstatus.sie* is copied to *sstatus.spie* and *sstatus.sie* is cleared.

4.4. S-mode Exception Return

When returning from a S-mode exception trap handler, the hardware behavior can be briefly described as follows.

- The pc is restored from the address in sepc register.
- *sstatus*.sie is copied to *sstatus*.sipe and spie is set.

4.5. S-mode Exception Nesting

Nuclei TEE does not support S-mode exception nesting with another S-mode exception. The behavior is unpredictable.

However, M-mode exception or NMI taken in S-mode exception is controllable, as the S-mode trap CSRs won't be updated.

5.TEE Low-Power Mechanism

5.1. TEE WFI Mechanism

There are some changes for WFI mechanism in the TEE. Due to the addition of S-mode interrupt, WFI can be divided into U-mode WFI, S-mode WFI and M-mode WFI.

U-mode WFI means that the processor executes wfi instruction in user mode, then the processor enters the User-Level Low-Power mode.

- The processor will be waked up if a S-mode interrupt wins the interrupt arbitration, and the processor will be trapped to the corresponding handler no matter *sstatus*.sie is set or not.
- The processor will be waked up if an M-mode interrupt wins the interrupt arbitration, and the processor will be trapped to the corresponding handler no matter *mstatus*.mie is set or not.

S-mode WFI means that the processor executes wfi instruction in supervisor mode, then the processor enters the Supervisor-Level Low-Power mode.

- The processor will be waked up if a S-mode interrupt wins the interrupt arbitration, and whether the processor will resume execution or be trapped to the interrupt handler depends on *sstatus*.sie.
- The processor will be waked up if an M-mode interrupt wins the interrupt arbitration, and the processor will be trapped to the corresponding handler no

matter *mstatus*.mie is set or not.

M-mode WFI means that the processor executes wfi instruction in machined mode, then the processor enters the Machine-Level Low-Power mode.

- Any S-mode interrupts cannot win the interrupt arbitration when the processor stays in the Machine-Level Low-Power mode, so the processor will still stay in Low-Power mode when it encounters a S-mode interrupt.
- The processor will be waked up if an M-mode interrupt wins the interrupt arbitration, and whether the processor will resume execution or be trapped to the interrupt handler depends on *mstatus*.mie.

In addition, NMI and debug request can also wake up U-mode WFI, S-mode WFI and M-mode WFI.

5.2. TEE WFE Mechanism

The WFE mechanism for a core with the TEE remains unchanged. No matter what the privilege mode is, setting wfe register and executing the wfi instruction will take the processor to Low-Power mode. The processor could be waked up by event, NMI, and debug request.

6.TEE Physical Memory Protection Mechanism

In addition to PMP, sPMP mechanism is implemented to achieve better isolation in TEE.

6.1. TEE PMP Mechanism

The PMP mechanism in TEE remains unchanged.

6.2. TEE sPMP Mechanism

The sPMP mechanism provides supervisor-mode control registers to allow physical memory access privileges (read, write, execute) to be specified for each physical memory region. The sPMP values are checked after the physical address to be accessed pass PMP checks described in the RISC-V privileged spec.

sPMP checks are applied to all accesses when the hart is running in U modes, and for loads
and stores when the MPRV bit is set in the mstatus register and the MPP field in the mstatus
register contains U. Optionally, sPMP checks can also apply to S-mode accesses, in which case
the sPMP values are locked to S-mode software, so that S-mode cannot change their values.
Unlike PMP registers, sPMP registers can always be modified by M-mode software even when
they are locked. sPMP registers can grant permissions to U-mode, which has none by default,
and revoke permissions from S-mode, which has full permissions by default.

6.2.1.sPMP CSR registers

6.2.1.1 sPMP CSR register list

Following Table 6 -12 lists the sPMP CSR registers. 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 16 sPMP entries are supported.

Table 6-12 sPMP CSR Registers

CSR Type	Number	Privilege	Name	Description
sPMP Related CSR Registers	0x1A0	SRW	spmpcfg0	Supervisor physical memory protection configuration
	0x1A1	SRW	spmpcfg1	Supervisor physical memory protection configuration
	0x1A2	SRW	spmpcfg2	Supervisor physical memory protection configuration
	0x1A3	SRW	spmpcfg3	Supervisor physical memory protection configuration
	0x1B0	SRW	spmpaddr0	Supervisor physical memory protection address register
	0x1B1	SRW	spmpaddr1	Supervisor physical memory protection address register
	…	SRW	…	…
	(0x1B0+n)	SRW	spmpaddrn	Supervisor physical memory protection address register

6.2.1.2 spmpcfg<x>

The sPMP configuration registers are packed into CSRs in the same way as PMP does. The
spmpcfg0-spmpcfg3 hold the configurations spmp0cfg-spmp15cfg for the 16 sPMP entries, as shown in Figure 6 -23.

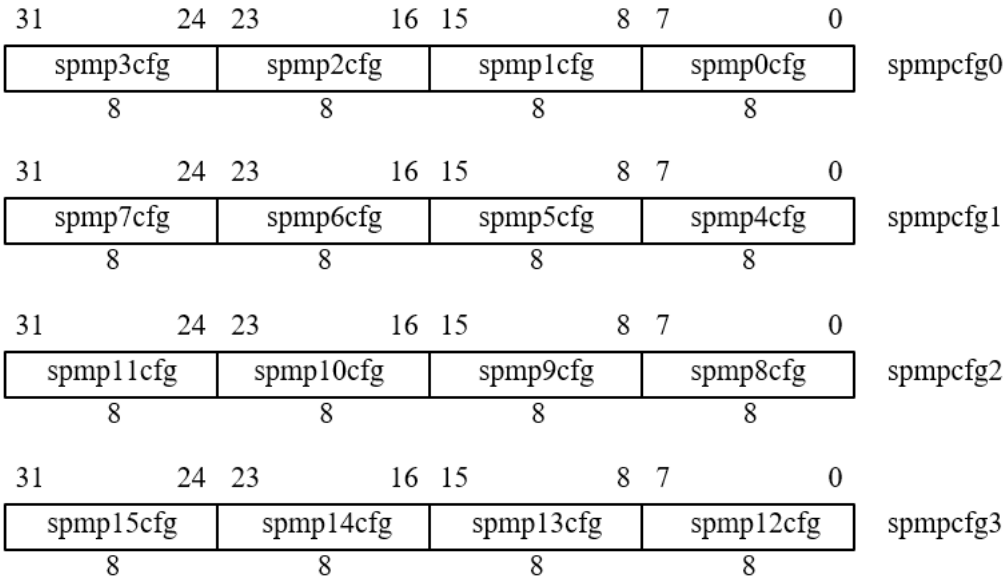

Figure 6-23 spmpcfg<x> layout

The format of sPMP configuration registers is the same as PMP configuration registers, as is
shown in.Table 6 -13

The R, W, and X bits, when set, indicate that the sPMP entry permits read, write and instruction execution, respectively. When one of these bits is clear, the corresponding access type is denied.

The U bit represents that the sPMP entry is for user mode, and will be used to enforce SMAP and SMEP (described in section 6.2.5.).

The A bits encodes the address-matching mode of the associated sPMP entry. The encoding of A field is the same as PMP's (TOR is not supported).

The remaining L field will be described in the following section 6.2.2.。

Table 6-13 spmp<x>cfg format

Field	Bits	Reset Value	Description
L	7	0	Indicates the sPMP entry is locked.
U	6	0	Indicates the sPMP entry is for user mode.
Reserved	5	N/A	Reserved, ties to 0.
A	4:3	0	Encodes the sPMP entry's address-matching mode.
X	2	0	Indicates the sPMP entry permits execution.
W	1	0	Indicates the sPMP entry permits write.
R	0	0	Indicates the sPMP entry permits read.

6.2.1.1 spmpaddr<x>

The sPMP address registers are CSRs named spmpaddr0-spmpaddr15. Each sPMP address
register encodes as shown in Figure 6 -24.

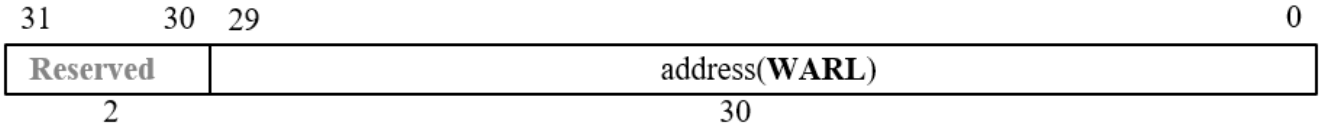

Figure 6-24 spmpaddr<x> format

6.2.2.sPMP Locking and Privilege Mode

The L bit indicates that the sPMP is locked to S-mode, i.e., S-mode writes to the configuration
register and associated address registers are ignored. Locked sPMP entries can only be

unlocked by M-mode or by a system reset. If sPMP entry i is locked, writes to the spmp<x>cfg and pmpaddr<x> are ignored. In addition to locking the sPMP entry, the L bit indicates whether the R/W/X permissions are enforced on S-mode accesses. When the L bit is set, these permissions are enforced for both user and supervisor modes (M-mode accesses are not affected). When the L bit is clear, any S-mode access matching the sPMP entry will succeed; the R/W/X permissions apply only to U modes.

6.2.3.sPMP Exception

Failed accesses generate a load, store, or instruction page fault (these exceptions should be delegated to S-mode software when happen).

6.2.4.sPMP Priority and Matching Logic

The sPMP checks only take effect after the memory access passes the PMP permission
checks. An M-mode access will not be checked by sPMP property.

Like PMP entries, sPMP entries are also statically prioritized. The lowest-numbered sPMP entry
that matches any byte of an access determines whether that access succeeds or fails. The
matching sPMP entry must match all bytes of an access, or the access fails, irrespective of the
L, R, W, and X bits.

If a sPMP entry matches all bytes of an access, then the L, R, W and X bits determine whether
the access succeeds or fails. If the privilege mode of the access is M, the access succeeds.
Otherwise, if the L bit is set or the privilege mode of the access is U, then the access succeeds
only if the R, W, or X bit corresponding to the access type is set.

If no sPMP entry matches an S-mode access (i.e., there is no such a sPMP entry whose L bit is
set and region contains the memory access), the access succeeds, otherwise the access is
checked according to the permission bits in sPMP entry. If no sPMP entry matches an U-mode
access, but at least one sPMP entry is implemented, the access fails.

6.2.5.SMAP and SMEP with sPMP

For SMAP, the SUM (permit Supervisor User Memory access) bit in the status register is leveraged to indicate the privilege with which S-mode loads, stores, and instruction fetches access physical memory.

- When SUM=0, S-mode physical memory accesses to memory that are accessible by
  U-mode (U=1 in Table 6 -13) will fault.
- When SUM=1, these accesses are permitted.

The SUM can take effect even when page-based virtual memory is not in effect.

For SMEP, it is not allowed for S-mode to execute codes in physical memory that are for
U-mode (U=1 in Table 6 -13).

7. TEE Configuration

N<xxx>_CFG_HAS_TEE is a global option to configure the implementation of TEE.